

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

Vít Novotný  
witiko@mail.muni.cz

Version 2.9.0  
September 14, 2020

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	33
1.1	Feedback . . . . .	2	2.4	ConT <sub>E</sub> Xt Interface . . . . .	43
1.2	Acknowledgements . . . . .	2	<b>3</b>	<b>Implementation</b>	<b>44</b>
1.3	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	44
<b>2</b>	<b>Interfaces</b>	<b>5</b>	3.2	Plain T <sub>E</sub> X Implementation	138
2.1	Lua Interface . . . . .	5	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	149
2.2	Plain T <sub>E</sub> X Interface . . . . .	18	3.4	ConT <sub>E</sub> Xt Implementation	160

## 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 converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. ;-)

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the 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   = "2.9.0",
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               "2016-2020 Vít Novotný"},
7   license   = "LPPL 1.3"
8 }
9
```

---

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

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

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

```
10 if not modules then modules = { } end
11 modules['markdown'] = metadata
```

## 1.1 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 on the T<sub>E</sub>X- $\LaTeX$  Stack Exchange.<sup>5</sup>

## 1.2 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.

Funding by the the Faculty of Informatics at the Masaryk University in Brno [1] is gratefully acknowledged.

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

## 1.3 Requirements

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

### 1.3.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaT<sub>E</sub>X engine:

**LPeg  $\geq$  0.10** A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg  $\geq$  0.10 is included in LuaT<sub>E</sub>X  $\geq$  0.72.0 (T<sub>E</sub>XLive  $\geq$  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 LuaT<sub>E</sub>X (T<sub>E</sub>XLive  $\geq$  2008).

---

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

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

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

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

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

All the abovelisted modules are statically linked into the current version of the LuaTeX engine [2, Section 3.3].

### 1.3.2 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 1.3.1), and the following Lua module:

**Lua File System** A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain 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 [2, Section 3.2].

The Lua File System module is statically linked into the LuaTeX engine [2, 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.

### 1.3.3 L<sup>A</sup>TeX Requirements

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

```
15 \NeedsTeXFormat{LaTeX2e}%
```

all the plain TeX prerequisites (see Section 1.3.2), and the following L<sup>A</sup>TeX 2<sub>ε</sub> packages:

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

16 `\RequirePackage{keyval}`

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

17 `\RequirePackage{url}`

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

18 `\RequirePackage{graphicx}`

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

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

19 `\RequirePackage{ifthen}`

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

20 `\RequirePackage{fancyvrb}`

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

21 `\RequirePackage{csvsimple}`

**gobble** A package that provides the `\@gobblethree`  $\TeX$  command.

22 `\RequirePackage{gobble}`

### 1.3.4 Con $\TeX$ t Prerequisites

The Con $\TeX$ t part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain  $\TeX$  prerequisites (see Section 1.3.2), and the following Con $\TeX$ t modules:

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

## 2 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  $\text{\TeX}$  nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is purely abstract. It serves as a means of structuring this 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.

### 2.1 Lua Interface

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

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

```
23 local M = {metadata = metadata}
```

#### 2.1.1 Conversion from Markdown to Plain $\text{\TeX}$

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

The following example Lua code converts the markdown string `Hello *world*!` to a  $\text{\TeX}$  output using the default options and prints the  $\text{\TeX}$  output:

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

#### 2.1.2 Options

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

```
24 local defaultOptions = {}
```

#### 2.1.3 File and Directory Names

`cacheDir`=*<path>* 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.

```
25 defaultOptions.cacheDir = "."
```

`frozenCacheFileName`=*<path>* default: frozenCache.tex

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

The frozen cache makes it possible to later typeset a plain 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.

```
26 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.

```
27 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.

```
28 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.

29 `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.

30 `defaultOptions.breakableBlockquotes = false`

`citationNbsps=true, false` default: false

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

31 `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.

```
32 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.``
```

```
33 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.

```
34 defaultOptions.contentBlocks = false
```

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

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

```
35 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.

```
36 defaultOptions.definitionLists = false
```

`fencedCode=true, false`

default: false

`true` Enable the commonmark fenced code block extension:

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

``` html


```

  <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.

```
37 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.

38 `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.

39 `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  $\langle number \rangle$ .

```
40 defaultOptions.frozenCacheCounter = 0
```

`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.

```
41 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.

```
42 defaultOptions.headerAttributes = false
```

`html=true, false` default: `false`

- `true` Enable the recognition of HTML tags, block elements, comments, HTML instructions, and entities in the input. Tags, block elements (along with contents), HTML instructions, and comments will be ignored and HTML entities will be replaced with the corresponding Unicode codepoints.
- `false` Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

43 `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.

44 `defaultOptions.hybrid = false`

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

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

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

- `false` Disable the pandoc inline footnote syntax extension.

45 `defaultOptions.inlineFootnotes = false`

`pipeTables=true, false`

default: `false`

`true` Enable the PHP Markdown table syntax extension:

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

`false` Disable the PHP Markdown table syntax extension.

46 `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.

47 `defaultOptions.preserveTabs = false`

`shiftHeadings=<shift amount>`

default: `0`

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

48 `defaultOptions.shiftHeadings = 0`

`slice=<the beginning and the end of a slice>`

default: `^ $`

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

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

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

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

`smartEllipses=true, false` default: false

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

`false` Preserve all ellipses in the input.

```
50 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

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

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

```
51 defaultOptions.startNumber = true
```

`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` Enable the Pandoc `table_captions` syntax extension.

```
52 defaultOptions.tableCaptions = false
```

`tightLists=true, false` default: true

- `true` Lists whose bullets do not consist of multiple paragraphs will be passed to the `\markdownRendererOlBeginTight`, `\markdownRendererOlEndTight`, `\markdownRendererUlBeginTight`, `\markdownRendererUlEndTight`, `\markdownRendererDlBeginTight`, and `\markdownRendererDlEndTight`  $\TeX$  macros.
- `false` Lists whose bullets do not consist of multiple paragraphs will be treated the same way as lists that do consist of multiple paragraphs.

53 `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.

54 `defaultOptions.underscores = true`

### 2.1.5 Command-Line Interface

To provide finer control over the conversion and to simplify debugging, a command-line Lua interface for converting a Markdown document to  $\TeX$  is also provided.

```
55
56 HELP_STRING = [[
57 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
58 where OPTIONS are documented in the Lua interface section of the
59 technical Markdown package documentation.
60
61 When OUTPUT_FILE is unspecified, the result of the conversion will be
62 written to the standard output. When INPUT_FILE is also unspecified, the
63 result of the conversion will be read from the standard input.
64
65 Report bugs to: witiko@mail.muni.cz
66 Markdown package home page: <https://github.com/witiko/markdown>]]
```

```

67
68 VERSION_STRING = [[
69 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
70
71 Copyright (C) ]] .. table.concat(metadata.copyright,
72                                     "\nCopyright (C) ") .. [[
73
74 License: ]] .. metadata.license
75
76 local function warn(s)
77   io.stderr:write("Warning: " .. s .. "\n") end
78
79 local function error(s)
80   io.stderr:write("Error: " .. s .. "\n")
81   os.exit(1) end
82
83 local process_options = true
84 local options = {}
85 local input_filename
86 local output_filename
87 for i = 1, #arg do
88   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.

```

89     if arg[i] == "--" then
90       process_options = false
91       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.

```

92     elseif arg[i]:match("=") then
93       key, value = arg[i]:match("(.-)=(.*)")

```

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

```

94       default_type = type(defaultOptions[key])
95       if default_type == "boolean" then
96         options[key] = (value == "true")
97       elseif default_type == "number" then
98         options[key] = tonumber(value)
99       else
100        if default_type ~= "string" then
101          if default_type == "nil" then
102            warn('Option "' .. key .. '" not recognized.')

```



```

103         else
104             warn('Option "' .. key .. '" type not recognized, please file ' ..
105                 'a report to the package maintainer.')
106         end
107         warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
108             key .. '" as a string.')
109         end
110         options[key] = value
111     end
112     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.

```

113     elseif arg[i] == "--help" or arg[i] == "-h" then
114         print(HELP_STRING)
115         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.

```

116     elseif arg[i] == "--version" or arg[i] == "-v" then
117         print(VERSION_STRING)
118         os.exit()
119     end
120 end

```

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

```

121 if input_filename == nil then
122     input_filename = arg[i]

```

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

```

123 elseif output_filename == nil then
124     output_filename = arg[i]
125 else
126     error('Unexpected argument: "' .. arg[i] .. "'.')
127 end
128 ::continue::
129 end

```

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

```

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

```

to convert the Markdown document `hello.md` to a TeX document `hello.tex`. After the Markdown package for our TeX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

This shows another advantage of using the command-line interface compared to using a higher-level TeX interface: it is unnecessary to provide shell access for the TeX engine.

## 2.2 Plain TeX Interface

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

```
130 \def\markdownLastModified{2020/09/14}%  
131 \def\markdownVersion{2.9.0}%
```

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

```
\input markdown
```

It is expected that the special plain TeX 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.

```
132 \let\markdownBegin\relax  
133 \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.

```
134 \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.

**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.

```
135 \let\markdownOptionFinalizeCache\undefined
```

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`.

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  $\TeX$  document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain  $\TeX$  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  $\TeX$  in  $\TeX$  engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

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

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

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

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

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain  $\TeX$

in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

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

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

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

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  $\TeX$  implementation. The option defaults to `..`.

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

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

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

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

The `\markdownOptionFrozenCacheFileName` macro corresponds to the Lua interface `frozenCacheFileName` option that sets the path to an output file (frozen cache) that will contain a mapping between an enumeration of the markdown documents in the plain  $\TeX$  document and their auxiliary cache files. The option defaults to `frozenCache.tex`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
142 \def\markdownOptionFrozenCacheFileName{\markdownOptionCacheDir/frozenCache.tex}
```

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

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

```
143 \let\markdownOptionBlankBeforeBlockquote\undefined
```

```

144 \let\markdownOptionBlankBeforeCodeFence\undefined
145 \let\markdownOptionBlankBeforeHeading\undefined
146 \let\markdownOptionBreakableBlockquotes\undefined
147 \let\markdownOptionCitations\undefined
148 \let\markdownOptionCitationNbsps\undefined
149 \let\markdownOptionContentBlocks\undefined
150 \let\markdownOptionContentBlocksLanguageMap\undefined
151 \let\markdownOptionDefinitionLists\undefined
152 \let\markdownOptionFootnotes\undefined
153 \let\markdownOptionFencedCode\undefined
154 \let\markdownOptionHashEnumerators\undefined
155 \let\markdownOptionHeaderAttributes\undefined
156 \let\markdownOptionHtml\undefined
157 \let\markdownOptionHybrid\undefined
158 \let\markdownOptionInlineFootnotes\undefined
159 \let\markdownOptionPipeTables\undefined
160 \let\markdownOptionPreserveTabs\undefined
161 \let\markdownOptionShiftHeadings\undefined
162 \let\markdownOptionSlice\undefined
163 \let\markdownOptionSmartEllipses\undefined
164 \let\markdownOptionStartNumber\undefined
165 \let\markdownOptionTableCaptions\undefined
166 \let\markdownOptionTightLists\undefined

```

**2.2.2.4 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 3.2.5) 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`.

```

167 \def\markdownOptionStripPercentSigns{false}%

```

### 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).

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

```

168 \def\markdownRendererInterblockSeparator{%
169   \markdownRendererInterblockSeparatorPrototype}%

```

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

```
170 \def\markdownRendererLineBreak{%  
171   \markdownRendererLineBreakPrototype}%
```

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

```
172 \def\markdownRendererEllipsis{%  
173   \markdownRendererEllipsisPrototype}%
```

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

```
174 \def\markdownRendererNbsp{%  
175   \markdownRendererNbspPrototype}%
```

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

```
176 \def\markdownRendererLeftBrace{%  
177   \markdownRendererLeftBracePrototype}%  
178 \def\markdownRendererRightBrace{%  
179   \markdownRendererRightBracePrototype}%  
180 \def\markdownRendererDollarSign{%  
181   \markdownRendererDollarSignPrototype}%  
182 \def\markdownRendererPercentSign{%  
183   \markdownRendererPercentSignPrototype}%  
184 \def\markdownRendererAmpersand{%  
185   \markdownRendererAmpersandPrototype}%  
186 \def\markdownRendererUnderscore{%  
187   \markdownRendererUnderscorePrototype}%  
188 \def\markdownRendererHash{%  
189   \markdownRendererHashPrototype}%  
190 \def\markdownRendererCircumflex{%  
191   \markdownRendererCircumflexPrototype}%  
192 \def\markdownRendererBackslash{%  
193   \markdownRendererBackslashPrototype}%  
194 \def\markdownRendererTilde{%  
195   \markdownRendererTildePrototype}%  
196 \def\markdownRendererPipe{%  
197   \markdownRendererPipePrototype}%
```

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

```
198 \def\markdownRendererCodeSpan{%  
199   \markdownRendererCodeSpanPrototype}%
```

**2.2.3.7 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.

```
200 \def\markdownRendererLink{%  
201   \markdownRendererLinkPrototype}%
```

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

```
202 \def\markdownRendererImage{%  
203   \markdownRendererImagePrototype}%
```

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

```
204 \def\markdownRendererContentBlock{%  
205   \markdownRendererContentBlockPrototype}%
```

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

```
206 \def\markdownRendererContentBlockOnlineImage{%  
207   \markdownRendererContentBlockOnlineImagePrototype}%
```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>6</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

---

<sup>6</sup>Local files take precedence. Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.



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  $\text{\TeX}$  directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```
208 \def\markdownRendererContentBlockCode{%
209   \markdownRendererContentBlockCodePrototype}%
```

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

```
210 \def\markdownRendererUllBegin{%
211   \markdownRendererUllBeginPrototype}%
```

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

```
212 \def\markdownRendererUllBeginTight{%
213   \markdownRendererUllBeginTightPrototype}%
```

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

```
214 \def\markdownRendererUllItem{%
215   \markdownRendererUllItemPrototype}%
```

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

```
216 \def\markdownRendererUllItemEnd{%
217   \markdownRendererUllItemEndPrototype}%
```

The `\markdownRendererUllEnd` 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.

```
218 \def\markdownRendererUllEnd{%
219   \markdownRendererUllEndPrototype}%
```

The `\markdownRendererUllEndTight` 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.

```
220 \def\markdownRendererU1EndTight{%  
221   \markdownRendererU1EndTightPrototype}%
```

**2.2.3.11 Ordered List Renderers** The `\markdownRendererO1Begin` 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.

```
222 \def\markdownRendererO1Begin{%  
223   \markdownRendererO1BeginPrototype}%
```

The `\markdownRendererO1BeginTight` 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.

```
224 \def\markdownRendererO1BeginTight{%  
225   \markdownRendererO1BeginTightPrototype}%
```

The `\markdownRendererO1Item` 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.

```
226 \def\markdownRendererO1Item{%  
227   \markdownRendererO1ItemPrototype}%
```

The `\markdownRendererO1ItemEnd` macro represents the end of an item in an ordered list. The macro receives no arguments.

```
228 \def\markdownRendererO1ItemEnd{%  
229   \markdownRendererO1ItemEndPrototype}%
```

The `\markdownRendererO1ItemWithNumber` macro represents an item in an ordered list. This macro will only be produced, when the `startNumber` option is enabled. The macro receives no arguments.

```
230 \def\markdownRendererO1ItemWithNumber{%  
231   \markdownRendererO1ItemWithNumberPrototype}%
```

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.

```
232 \def\markdownRendererO1End{%  
233   \markdownRendererO1EndPrototype}%
```

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.

```
234 \def\markdownRendererDlEndTight{%  
235   \markdownRendererDlEndTightPrototype}%
```

**2.2.3.12 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.

```
236 \def\markdownRendererDlBegin{%  
237   \markdownRendererDlBeginPrototype}%
```

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

```
238 \def\markdownRendererDlBeginTight{%  
239   \markdownRendererDlBeginTightPrototype}%
```

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

```
240 \def\markdownRendererDlItem{%  
241   \markdownRendererDlItemPrototype}%
```

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

```
242 \def\markdownRendererDlItemEnd{%  
243   \markdownRendererDlItemEndPrototype}%
```

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

```
244 \def\markdownRendererDlDefinitionBegin{%  
245   \markdownRendererDlDefinitionBeginPrototype}%
```

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

```
246 \def\markdownRendererDlDefinitionEnd{%  
247   \markdownRendererDlDefinitionEndPrototype}%
```

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

```
248 \def\markdownRendererDlEnd{%  
249   \markdownRendererDlEndPrototype}%
```

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

```
250 \def\markdownRendererDlEndTight{%  
251   \markdownRendererDlEndTightPrototype}%
```

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

```
252 \def\markdownRendererEmphasis{%  
253   \markdownRendererEmphasisPrototype}%
```

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

```
254 \def\markdownRendererStrongEmphasis{%  
255   \markdownRendererStrongEmphasisPrototype}%
```

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

```
256 \def\markdownRendererBlockQuoteBegin{%  
257   \markdownRendererBlockQuoteBeginPrototype}%
```

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

```
258 \def\markdownRendererBlockQuoteEnd{%  
259   \markdownRendererBlockQuoteEndPrototype}%
```

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

```
260 \def\markdownRendererInputVerbatim{%  
261   \markdownRendererInputVerbatimPrototype}%
```

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.

```
262 \def\markdownRendererInputFencedCode{%  
263   \markdownRendererInputFencedCodePrototype}%
```

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

```
264 \def\markdownRendererHeadingOne{%  
265   \markdownRendererHeadingOnePrototype}%
```

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

```
266 \def\markdownRendererHeadingTwo{%  
267   \markdownRendererHeadingTwoPrototype}%
```

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

```
268 \def\markdownRendererHeadingThree{%  
269   \markdownRendererHeadingThreePrototype}%
```

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

```
270 \def\markdownRendererHeadingFour{%  
271   \markdownRendererHeadingFourPrototype}%
```

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

```
272 \def\markdownRendererHeadingFive{%  
273   \markdownRendererHeadingFivePrototype}%
```

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

```
274 \def\markdownRendererHeadingSix{%  
275   \markdownRendererHeadingSixPrototype}%
```

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

```
276 \def\markdownRendererHorizontalRule{%  
277   \markdownRendererHorizontalRulePrototype}%
```

**2.2.3.18 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.

```
278 \def\markdownRendererFootnote{%  
279   \markdownRendererFootnotePrototype}%
```

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

```
280 \def\markdownRendererCite{%
281   \markdownRendererCitePrototype}%
```

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

```
282 \def\markdownRendererTextCite{%
283   \markdownRendererTextCitePrototype}%
```

**2.2.3.21 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 `{\langle caption \rangle}{\langle number of rows \rangle}{\langle number of columns \rangle}` followed by `{\langle alignments \rangle}` and then by `{\langle row \rangle}` repeated `\langle number of rows \rangle` times, where `\langle row \rangle` is `{\langle column \rangle}` repeated `\langle number of columns \rangle` times, `\langle alignments \rangle` is `\langle alignment \rangle` repeated `\langle number of columns \rangle` times, and `\langle alignment \rangle` 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.

```
284 \def\markdownRendererTable{%
285   \markdownRendererTablePrototype}%
```

## 2.2.4 Token Renderer Prototypes

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

```
286 \def\markdownRendererInterblockSeparatorPrototype{}%
287 \def\markdownRendererLineBreakPrototype{}%
288 \def\markdownRendererEllipsisPrototype{}%
289 \def\markdownRendererNbspPrototype{}%
```

```

290 \def\markdownRendererLeftBracePrototype{}%
291 \def\markdownRendererRightBracePrototype{}%
292 \def\markdownRendererDollarSignPrototype{}%
293 \def\markdownRendererPercentSignPrototype{}%
294 \def\markdownRendererAmpersandPrototype{}%
295 \def\markdownRendererUnderscorePrototype{}%
296 \def\markdownRendererHashPrototype{}%
297 \def\markdownRendererCircumflexPrototype{}%
298 \def\markdownRendererBackslashPrototype{}%
299 \def\markdownRendererTildePrototype{}%
300 \def\markdownRendererPipePrototype{}%
301 \def\markdownRendererCodeSpanPrototype#1{}%
302 \def\markdownRendererLinkPrototype#1#2#3#4{}%
303 \def\markdownRendererImagePrototype#1#2#3#4{}%
304 \def\markdownRendererContentBlockPrototype#1#2#3#4{}%
305 \def\markdownRendererContentBlockOnlineImagePrototype#1#2#3#4{}%
306 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{}%
307 \def\markdownRendererUlBeginPrototype{}%
308 \def\markdownRendererUlBeginTightPrototype{}%
309 \def\markdownRendererUlItemPrototype{}%
310 \def\markdownRendererUlItemEndPrototype{}%
311 \def\markdownRendererUlEndPrototype{}%
312 \def\markdownRendererUlEndTightPrototype{}%
313 \def\markdownRendererOlBeginPrototype{}%
314 \def\markdownRendererOlBeginTightPrototype{}%
315 \def\markdownRendererOlItemPrototype{}%
316 \def\markdownRendererOlItemWithNumberPrototype#1{}%
317 \def\markdownRendererOlItemEndPrototype{}%
318 \def\markdownRendererOlEndPrototype{}%
319 \def\markdownRendererOlEndTightPrototype{}%
320 \def\markdownRendererDlBeginPrototype{}%
321 \def\markdownRendererDlBeginTightPrototype{}%
322 \def\markdownRendererDlItemPrototype#1{}%
323 \def\markdownRendererDlItemEndPrototype{}%
324 \def\markdownRendererDlDefinitionBeginPrototype{}%
325 \def\markdownRendererDlDefinitionEndPrototype{}%
326 \def\markdownRendererDlEndPrototype{}%
327 \def\markdownRendererDlEndTightPrototype{}%
328 \def\markdownRendererEmphasisPrototype#1{}%
329 \def\markdownRendererStrongEmphasisPrototype#1{}%
330 \def\markdownRendererBlockQuoteBeginPrototype{}%
331 \def\markdownRendererBlockQuoteEndPrototype{}%
332 \def\markdownRendererInputVerbatimPrototype#1{}%
333 \def\markdownRendererInputFencedCodePrototype#1#2{}%
334 \def\markdownRendererHeadingOnePrototype#1{}%
335 \def\markdownRendererHeadingTwoPrototype#1{}%
336 \def\markdownRendererHeadingThreePrototype#1{}%

```

```

337 \def\markdownRendererHeadingFourPrototype#1{}%
338 \def\markdownRendererHeadingFivePrototype#1{}%
339 \def\markdownRendererHeadingSixPrototype#1{}%
340 \def\markdownRendererHorizontalRulePrototype{}%
341 \def\markdownRendererFootnotePrototype#1{}%
342 \def\markdownRendererCitePrototype#1{}%
343 \def\markdownRendererTextCitePrototype#1{}%
344 \def\markdownRendererTablePrototype#1#2#3{}%

```

### 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.

```

345 \def\markdownInfo#1{}%
346 \def\markdownWarning#1{}%

```

The `\markdownError` macro receives a second argument that provides a help text.

```

347 \def\markdownError#1#2{}%

```

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

### 2.2.6 Miscellanea

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

```

348 \let\markdownMakeOther\relax

```

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

```

349 \let\markdownReadAndConvert\relax
350 \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*.

```

351 \catcode`\|=0\catcode`\=12%
352 |gdef|markdownBegin{%
353 |markdownReadAndConvert{\markdownEnd}%

```



```

354         {|\markdownEnd}}}%
355 |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 3.2.7), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

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

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

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

```

356 \ifx\markdownMode\undefined
357   \ifx\directlua\undefined
358     \def\markdownMode{0}%
359   \else
360     \def\markdownMode{2}%
361   \fi
362 \fi

```

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

```

363 \def\markdownLuaRegisterIBCallback#1{\relax}%
364 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

## 2.3 $\LaTeX$ Interface

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

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

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

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

### 2.3.1 Typesetting Markdown

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

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

```
365 \newenvironment{markdown}\relax\relax
366 \newenvironment{markdown*}[1]\relax\relax
```

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

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

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

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

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
```

```
\markdownInput[smartEllipses]{hello.md}
% ...
\end{document}
```

## 2.3.2 Options

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

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

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

```
367 \newcommand\markdownSetup[1]{%
368   \setkeys{markdownOptions}{#1}}%
```

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

```
369 \define@key{markdownOptions}{helperScriptFileName}{%
370   \def\markdownOptionHelperScriptFileName{#1}}%
371 \define@key{markdownOptions}{inputTempFileName}{%
372   \def\markdownOptionInputTempFileName{#1}}%
373 \define@key{markdownOptions}{outputTempFileName}{%
374   \def\markdownOptionOutputTempFileName{#1}}%
375 \define@key{markdownOptions}{errorTempFileName}{%
376   \def\markdownOptionErrorTempFileName{#1}}%
377 \define@key{markdownOptions}{cacheDir}{%
378   \def\markdownOptionCacheDir{#1}}%
379 \define@key{markdownOptions}{outputDir}{%
380   \def\markdownOptionOutputDir{#1}}%
381 \define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
382   \def\markdownOptionBlankBeforeBlockquote{#1}}%
383 \define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
384   \def\markdownOptionBlankBeforeCodeFence{#1}}%
385 \define@key{markdownOptions}{blankBeforeHeading}[true]{%
386   \def\markdownOptionBlankBeforeHeading{#1}}%
387 \define@key{markdownOptions}{breakableBlockquotes}[true]{%
388   \def\markdownOptionBreakableBlockquotes{#1}}%
389 \define@key{markdownOptions}{citations}[true]{%
390   \def\markdownOptionCitations{#1}}%
391 \define@key{markdownOptions}{citationNbsps}[true]{%
```

```

392 \def\markdownOptionCitationNbsps{#1}}%
393 \define@key{markdownOptions}{contentBlocks}[true]{%
394 \def\markdownOptionContentBlocks{#1}}%
395 \define@key{markdownOptions}{codeSpans}[true]{%
396 \def\markdownOptionCodeSpans{#1}}%
397 \define@key{markdownOptions}{contentBlocksLanguageMap}{%
398 \def\markdownOptionContentBlocksLanguageMap{#1}}%
399 \define@key{markdownOptions}{definitionLists}[true]{%
400 \def\markdownOptionDefinitionLists{#1}}%
401 \define@key{markdownOptions}{footnotes}[true]{%
402 \def\markdownOptionFootnotes{#1}}%
403 \define@key{markdownOptions}{fencedCode}[true]{%
404 \def\markdownOptionFencedCode{#1}}%
405 \define@key{markdownOptions}{hashEnumerators}[true]{%
406 \def\markdownOptionHashEnumerators{#1}}%
407 \define@key{markdownOptions}{headerAttributes}[true]{%
408 \def\markdownOptionHeaderAttributes{#1}}%
409 \define@key{markdownOptions}{html}[true]{%
410 \def\markdownOptionHtml{#1}}%
411 \define@key{markdownOptions}{hybrid}[true]{%
412 \def\markdownOptionHybrid{#1}}%
413 \define@key{markdownOptions}{inlineFootnotes}[true]{%
414 \def\markdownOptionInlineFootnotes{#1}}%
415 \define@key{markdownOptions}{pipeTables}[true]{%
416 \def\markdownOptionPipeTables{#1}}%
417 \define@key{markdownOptions}{preserveTabs}[true]{%
418 \def\markdownOptionPreserveTabs{#1}}%
419 \define@key{markdownOptions}{smartEllipses}[true]{%
420 \def\markdownOptionSmartEllipses{#1}}%
421 \define@key{markdownOptions}{shiftHeadings}{%
422 \def\markdownOptionShiftHeadings{#1}}%
423 \define@key{markdownOptions}{slice}{%
424 \def\markdownOptionSlice{#1}}%
425 \define@key{markdownOptions}{startNumber}[true]{%
426 \def\markdownOptionStartNumber{#1}}%
427 \define@key{markdownOptions}{tableCaptions}[true]{%
428 \def\markdownOptionTableCaptions{#1}}%
429 \define@key{markdownOptions}{tightLists}[true]{%
430 \def\markdownOptionTightLists{#1}}%
431 \define@key{markdownOptions}{underscores}[true]{%
432 \def\markdownOptionUnderscores{#1}}%
433 \define@key{markdownOptions}{stripPercentSigns}[true]{%
434 \def\markdownOptionStripPercentSigns{#1}}%

```

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

To ensure compatibility with the `minted` package [6, 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.

```
435 \define@key{markdownOptions}{finalizeCache}[true]{%
436   \def\markdownOptionFinalizeCache{#1}}%
437 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
438 \define@key{markdownOptions}{frozenCache}[true]{%
439   \def\markdownOptionFrozenCache{#1}}%
440 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}
441 \define@key{markdownOptions}{frozenCacheFileName}{%
442   \def\markdownOptionFrozenCacheFileName{#1}}%
```

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

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

### 2.3.2.2 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).

```
443 \define@key{markdownRenderers}{interblockSeparator}{%
444   \renewcommand\markdownRendererInterblockSeparator{#1}}%
445 \define@key{markdownRenderers}{lineBreak}{%
446   \renewcommand\markdownRendererLineBreak{#1}}%
447 \define@key{markdownRenderers}{ellipsis}{%
```

```

448 \renewcommand\markdownRendererEllipsis{#1}}%
449 \define@key{markdownRenderers}{nbsp}{%
450 \renewcommand\markdownRendererNbsp{#1}}%
451 \define@key{markdownRenderers}{leftBrace}{%
452 \renewcommand\markdownRendererLeftBrace{#1}}%
453 \define@key{markdownRenderers}{rightBrace}{%
454 \renewcommand\markdownRendererRightBrace{#1}}%
455 \define@key{markdownRenderers}{dollarSign}{%
456 \renewcommand\markdownRendererDollarSign{#1}}%
457 \define@key{markdownRenderers}{percentSign}{%
458 \renewcommand\markdownRendererPercentSign{#1}}%
459 \define@key{markdownRenderers}{ampersand}{%
460 \renewcommand\markdownRendererAmpersand{#1}}%
461 \define@key{markdownRenderers}{underscore}{%
462 \renewcommand\markdownRendererUnderscore{#1}}%
463 \define@key{markdownRenderers}{hash}{%
464 \renewcommand\markdownRendererHash{#1}}%
465 \define@key{markdownRenderers}{circumflex}{%
466 \renewcommand\markdownRendererCircumflex{#1}}%
467 \define@key{markdownRenderers}{backslash}{%
468 \renewcommand\markdownRendererBackslash{#1}}%
469 \define@key{markdownRenderers}{tilde}{%
470 \renewcommand\markdownRendererTilde{#1}}%
471 \define@key{markdownRenderers}{pipe}{%
472 \renewcommand\markdownRendererPipe{#1}}%
473 \define@key{markdownRenderers}{codeSpan}{%
474 \renewcommand\markdownRendererCodeSpan[1]{#1}}%
475 \define@key{markdownRenderers}{link}{%
476 \renewcommand\markdownRendererLink[4]{#1}}%
477 \define@key{markdownRenderers}{contentBlock}{%
478 \renewcommand\markdownRendererContentBlock[4]{#1}}%
479 \define@key{markdownRenderers}{contentBlockOnlineImage}{%
480 \renewcommand\markdownRendererContentBlockOnlineImage[4]{#1}}%
481 \define@key{markdownRenderers}{contentBlockCode}{%
482 \renewcommand\markdownRendererContentBlockCode[5]{#1}}%
483 \define@key{markdownRenderers}{image}{%
484 \renewcommand\markdownRendererImage[4]{#1}}%
485 \define@key{markdownRenderers}{ulBegin}{%
486 \renewcommand\markdownRendererUlBegin{#1}}%
487 \define@key{markdownRenderers}{ulBeginTight}{%
488 \renewcommand\markdownRendererUlBeginTight{#1}}%
489 \define@key{markdownRenderers}{ulItem}{%
490 \renewcommand\markdownRendererUlItem{#1}}%
491 \define@key{markdownRenderers}{ulItemEnd}{%
492 \renewcommand\markdownRendererUlItemEnd{#1}}%
493 \define@key{markdownRenderers}{ulEnd}{%
494 \renewcommand\markdownRendererUlEnd{#1}}%

```

```

495 \define@key{markdownRenderers}{ulEndTight}{%
496   \renewcommand\markdownRendererUlEndTight{#1}}%
497 \define@key{markdownRenderers}{olBegin}{%
498   \renewcommand\markdownRendererOlBegin{#1}}%
499 \define@key{markdownRenderers}{olBeginTight}{%
500   \renewcommand\markdownRendererOlBeginTight{#1}}%
501 \define@key{markdownRenderers}{olItem}{%
502   \renewcommand\markdownRendererOlItem{#1}}%
503 \define@key{markdownRenderers}{olItemWithNumber}{%
504   \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
505 \define@key{markdownRenderers}{olItemEnd}{%
506   \renewcommand\markdownRendererOlItemEnd{#1}}%
507 \define@key{markdownRenderers}{olEnd}{%
508   \renewcommand\markdownRendererOlEnd{#1}}%
509 \define@key{markdownRenderers}{olEndTight}{%
510   \renewcommand\markdownRendererOlEndTight{#1}}%
511 \define@key{markdownRenderers}{dlBegin}{%
512   \renewcommand\markdownRendererDlBegin{#1}}%
513 \define@key{markdownRenderers}{dlBeginTight}{%
514   \renewcommand\markdownRendererDlBeginTight{#1}}%
515 \define@key{markdownRenderers}{dlItem}{%
516   \renewcommand\markdownRendererDlItem[1]{#1}}%
517 \define@key{markdownRenderers}{dlItemEnd}{%
518   \renewcommand\markdownRendererDlItemEnd{#1}}%
519 \define@key{markdownRenderers}{dlDefinitionBegin}{%
520   \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
521 \define@key{markdownRenderers}{dlDefinitionEnd}{%
522   \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
523 \define@key{markdownRenderers}{dlEnd}{%
524   \renewcommand\markdownRendererDlEnd{#1}}%
525 \define@key{markdownRenderers}{dlEndTight}{%
526   \renewcommand\markdownRendererDlEndTight{#1}}%
527 \define@key{markdownRenderers}{emphasis}{%
528   \renewcommand\markdownRendererEmphasis[1]{#1}}%
529 \define@key{markdownRenderers}{strongEmphasis}{%
530   \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
531 \define@key{markdownRenderers}{blockquoteBegin}{%
532   \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
533 \define@key{markdownRenderers}{blockquoteEnd}{%
534   \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
535 \define@key{markdownRenderers}{inputVerbatim}{%
536   \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
537 \define@key{markdownRenderers}{inputFencedCode}{%
538   \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
539 \define@key{markdownRenderers}{headingOne}{%
540   \renewcommand\markdownRendererHeadingOne[1]{#1}}%
541 \define@key{markdownRenderers}{headingTwo}{%

```

```

542 \renewcommand\markdownRendererHeadingTwo[1]{#1}%
543 \define@key{markdownRenderers}{headingThree}{%
544 \renewcommand\markdownRendererHeadingThree[1]{#1}}%
545 \define@key{markdownRenderers}{headingFour}{%
546 \renewcommand\markdownRendererHeadingFour[1]{#1}}%
547 \define@key{markdownRenderers}{headingFive}{%
548 \renewcommand\markdownRendererHeadingFive[1]{#1}}%
549 \define@key{markdownRenderers}{headingSix}{%
550 \renewcommand\markdownRendererHeadingSix[1]{#1}}%
551 \define@key{markdownRenderers}{horizontalRule}{%
552 \renewcommand\markdownRendererHorizontalRule{#1}}%
553 \define@key{markdownRenderers}{footnote}{%
554 \renewcommand\markdownRendererFootnote[1]{#1}}%
555 \define@key{markdownRenderers}{cite}{%
556 \renewcommand\markdownRendererCite[1]{#1}}%
557 \define@key{markdownRenderers}{textCite}{%
558 \renewcommand\markdownRendererTextCite[1]{#1}}%
559 \define@key{markdownRenderers}{table}{%
560 \renewcommand\markdownRendererTable[3]{#1}}%

```

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

```

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

```

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

```

561 \define@key{markdownRendererPrototypes}{interblockSeparator}{%
562 \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
563 \define@key{markdownRendererPrototypes}{lineBreak}{%
564 \renewcommand\markdownRendererLineBreakPrototype{#1}}%
565 \define@key{markdownRendererPrototypes}{ellipsis}{%
566 \renewcommand\markdownRendererEllipsisPrototype{#1}}%
567 \define@key{markdownRendererPrototypes}{nbsp}{%
568 \renewcommand\markdownRendererNbspPrototype{#1}}%
569 \define@key{markdownRendererPrototypes}{leftBrace}{%
570 \renewcommand\markdownRendererLeftBracePrototype{#1}}%

```



```

571 \define@key{markdownRendererPrototypes}{rightBrace}{%
572   \renewcommand\markdownRendererRightBracePrototype{#1}}%
573 \define@key{markdownRendererPrototypes}{dollarSign}{%
574   \renewcommand\markdownRendererDollarSignPrototype{#1}}%
575 \define@key{markdownRendererPrototypes}{percentSign}{%
576   \renewcommand\markdownRendererPercentSignPrototype{#1}}%
577 \define@key{markdownRendererPrototypes}{ampersand}{%
578   \renewcommand\markdownRendererAmpersandPrototype{#1}}%
579 \define@key{markdownRendererPrototypes}{underscore}{%
580   \renewcommand\markdownRendererUnderscorePrototype{#1}}%
581 \define@key{markdownRendererPrototypes}{hash}{%
582   \renewcommand\markdownRendererHashPrototype{#1}}%
583 \define@key{markdownRendererPrototypes}{circumflex}{%
584   \renewcommand\markdownRendererCircumflexPrototype{#1}}%
585 \define@key{markdownRendererPrototypes}{backslash}{%
586   \renewcommand\markdownRendererBackslashPrototype{#1}}%
587 \define@key{markdownRendererPrototypes}{tilde}{%
588   \renewcommand\markdownRendererTildePrototype{#1}}%
589 \define@key{markdownRendererPrototypes}{pipe}{%
590   \renewcommand\markdownRendererPipePrototype{#1}}%
591 \define@key{markdownRendererPrototypes}{codeSpan}{%
592   \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
593 \define@key{markdownRendererPrototypes}{link}{%
594   \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
595 \define@key{markdownRendererPrototypes}{contentBlock}{%
596   \renewcommand\markdownRendererContentBlockPrototype[4]{#1}}%
597 \define@key{markdownRendererPrototypes}{contentBlockOnlineImage}{%
598   \renewcommand\markdownRendererContentBlockOnlineImagePrototype[4]{#1}}%
599 \define@key{markdownRendererPrototypes}{contentBlockCode}{%
600   \renewcommand\markdownRendererContentBlockCodePrototype[5]{#1}}%
601 \define@key{markdownRendererPrototypes}{image}{%
602   \renewcommand\markdownRendererImagePrototype[4]{#1}}%
603 \define@key{markdownRendererPrototypes}{ulBegin}{%
604   \renewcommand\markdownRendererUlBeginPrototype{#1}}%
605 \define@key{markdownRendererPrototypes}{ulBeginTight}{%
606   \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
607 \define@key{markdownRendererPrototypes}{ulItem}{%
608   \renewcommand\markdownRendererUlItemPrototype{#1}}%
609 \define@key{markdownRendererPrototypes}{ulItemEnd}{%
610   \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
611 \define@key{markdownRendererPrototypes}{ulEnd}{%
612   \renewcommand\markdownRendererUlEndPrototype{#1}}%
613 \define@key{markdownRendererPrototypes}{ulEndTight}{%
614   \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
615 \define@key{markdownRendererPrototypes}{olBegin}{%
616   \renewcommand\markdownRendererOlBeginPrototype{#1}}%
617 \define@key{markdownRendererPrototypes}{olBeginTight}{%

```

```

618 \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
619 \define@key{markdownRendererPrototypes}{olItem}{%
620 \renewcommand\markdownRendererOlItemPrototype{#1}}%
621 \define@key{markdownRendererPrototypes}{olItemWithNumber}{%
622 \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
623 \define@key{markdownRendererPrototypes}{olItemEnd}{%
624 \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
625 \define@key{markdownRendererPrototypes}{olEnd}{%
626 \renewcommand\markdownRendererOlEndPrototype{#1}}%
627 \define@key{markdownRendererPrototypes}{olEndTight}{%
628 \renewcommand\markdownRendererOlEndTightPrototype{#1}}%
629 \define@key{markdownRendererPrototypes}{dlBegin}{%
630 \renewcommand\markdownRendererDlBeginPrototype{#1}}%
631 \define@key{markdownRendererPrototypes}{dlBeginTight}{%
632 \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
633 \define@key{markdownRendererPrototypes}{dlItem}{%
634 \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
635 \define@key{markdownRendererPrototypes}{dlItemEnd}{%
636 \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
637 \define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
638 \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%
639 \define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
640 \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
641 \define@key{markdownRendererPrototypes}{dlEnd}{%
642 \renewcommand\markdownRendererDlEndPrototype{#1}}%
643 \define@key{markdownRendererPrototypes}{dlEndTight}{%
644 \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
645 \define@key{markdownRendererPrototypes}{emphasis}{%
646 \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
647 \define@key{markdownRendererPrototypes}{strongEmphasis}{%
648 \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
649 \define@key{markdownRendererPrototypes}{blockquoteBegin}{%
650 \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
651 \define@key{markdownRendererPrototypes}{blockquoteEnd}{%
652 \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
653 \define@key{markdownRendererPrototypes}{inputVerbatim}{%
654 \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
655 \define@key{markdownRendererPrototypes}{inputFencedCode}{%
656 \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
657 \define@key{markdownRendererPrototypes}{headingOne}{%
658 \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
659 \define@key{markdownRendererPrototypes}{headingTwo}{%
660 \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
661 \define@key{markdownRendererPrototypes}{headingThree}{%
662 \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%
663 \define@key{markdownRendererPrototypes}{headingFour}{%
664 \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%

```

```

665 \define@key{markdownRendererPrototypes}{headingFive}{%
666   \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
667 \define@key{markdownRendererPrototypes}{headingSix}{%
668   \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
669 \define@key{markdownRendererPrototypes}{horizontalRule}{%
670   \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
671 \define@key{markdownRendererPrototypes}{footnote}{%
672   \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
673 \define@key{markdownRendererPrototypes}{cite}{%
674   \renewcommand\markdownRendererCitePrototype[1]{#1}}%
675 \define@key{markdownRendererPrototypes}{textCite}{%
676   \renewcommand\markdownRendererTextCitePrototype[1]{#1}}%
677 \define@key{markdownRendererPrototypes}{table}{%
678   \renewcommand\markdownRendererTablePrototype[3]{#1}}%

```

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).

```

679 \writestatus{loading}{ConTeXt User Module / markdown}%
680 \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.

```
681 \let\startmarkdown\relax
682 \let\stopmarkdown\relax
```

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

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

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

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

## 3 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.

### 3.1 Lua Implementation

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

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

```
683 local upper, gsub, format, length =
684   string.upper, string.gsub, string.format, string.len
685 local concat = table.concat
686 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
687   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
688   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.

```
689 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.

```
690 function util.err(msg, exit_code)
691   io.stderr:write("markdown.lua: " .. msg .. "\n")
692   os.exit(exit_code or 1)
693 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.

```
694 function util.cache(dir, string, salt, transform, suffix)
695   local digest = md5.sumhexa(string .. (salt or ""))
696   local name = util.pathname(dir, digest .. suffix)
697   local file = io.open(name, "r")
698   if file == nil then -- If no cache entry exists, then create a new one.
699     local file = assert(io.open(name, "w"))
700     local result = string
701     if transform ~= nil then
702       result = transform(result)
703     end
704     assert(file:write(result))
705     assert(file:close())
706   end
707   return name
708 end
```

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

```
709 function util.table_copy(t)
710   local u = { }
711   for k, v in pairs(t) do u[k] = v end
712   return setmetatable(u, getmetatable(t))
713 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 [7, Chapter 21].

```
714 function util.expand_tabs_in_line(s, tabstop)
```

```

715 local tab = tabstop or 4
716 local corr = 0
717 return (s:gsub("\t", function(p)
718         local sp = tab - (p - 1 + corr) % tab
719         corr = corr - 1 + sp
720         return string.rep(" ", sp)
721     end))
722 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.

```

723 function util.walk(t, f)
724 local typ = type(t)
725 if typ == "string" then
726     f(t)
727 elseif typ == "table" then
728     local i = 1
729     local n
730     n = t[i]
731     while n do
732         util.walk(n, f)
733         i = i + 1
734         n = t[i]
735     end
736 elseif typ == "function" then
737     local ok, val = pcall(t)
738     if ok then
739         util.walk(val, f)
740     end
741 else
742     f(tostring(t))
743 end
744 end

```

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

```

745 function util.flatten(ary)
746 local new = {}
747 for _,v in ipairs(ary) do
748     if type(v) == "table" then
749         for _,w in ipairs(util.flatten(v)) do
750             new[#new + 1] = w
751         end
752     else
753         new[#new + 1] = v

```

```

754     end
755   end
756   return new
757 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.

```

758 function util.rope_to_string(rope)
759   local buffer = {}
760   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
761   return table.concat(buffer)
762 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.

```

763 function util.rope_last(rope)
764   if #rope == 0 then
765     return nil
766   else
767     local l = rope[#rope]
768     if type(l) == "table" then
769       return util.rope_last(l)
770     else
771       return l
772     end
773   end
774 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  $1 \leq i \leq \#ary$ .

```

775 function util.intersperse(ary, x)
776   local new = {}
777   local l = #ary
778   for i,v in ipairs(ary) do
779     local n = #new
780     new[n + 1] = v
781     if i ~= l then
782       new[n + 2] = x
783     end
784   end
785   return new
786 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  $1 \leq i \leq \#ary$ .

```

787 function util.map(ary, f)
788   local new = {}

```

```

789   for i,v in ipairs(ary) do
790       new[i] = f(v)
791   end
792   return new
793 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.

```

794 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

795   local char_escapes_list = ""
796   for i,_ in pairs(char_escapes) do
797       char_escapes_list = char_escapes_list .. i
798   end

```

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

```

799   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.

```

800   if string_escapes then
801       for k,v in pairs(string_escapes) do
802           escapable = P(k) / v + escapable
803       end
804   end

```

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

```

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

```

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

```

806   return function(s)
807       return lpeg.match(escape_string, s)
808   end
809 end

```



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

```
810 function util.pathname(dir, file)
811   if #dir == 0 then
812     return file
813   else
814     return dir .. "/" .. file
815   end
816 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.

```
817 local entities = {}
818
819 local character_entities = {
820   ["Tab"] = 9,
821   ["NewLine"] = 10,
822   ["excl"] = 33,
823   ["quot"] = 34,
824   ["QUOT"] = 34,
825   ["num"] = 35,
826   ["dollar"] = 36,
827   ["percent"] = 37,
828   ["amp"] = 38,
829   ["AMP"] = 38,
830   ["apos"] = 39,
831   ["lpar"] = 40,
832   ["rpar"] = 41,
833   ["ast"] = 42,
834   ["midast"] = 42,
835   ["plus"] = 43,
836   ["comma"] = 44,
837   ["period"] = 46,
838   ["sol"] = 47,
839   ["colon"] = 58,
840   ["semi"] = 59,
841   ["lt"] = 60,
842   ["LT"] = 60,
843   ["equals"] = 61,
844   ["gt"] = 62,
845   ["GT"] = 62,
846   ["quest"] = 63,
847   ["commat"] = 64,
```

848 ["lsqb"] = 91,  
849 ["lbrack"] = 91,  
850 ["bsol"] = 92,  
851 ["rsqb"] = 93,  
852 ["rbrack"] = 93,  
853 ["Hat"] = 94,  
854 ["lowbar"] = 95,  
855 ["grave"] = 96,  
856 ["DiacriticalGrave"] = 96,  
857 ["lcub"] = 123,  
858 ["lbrace"] = 123,  
859 ["verbar"] = 124,  
860 ["vert"] = 124,  
861 ["VerticalLine"] = 124,  
862 ["rcub"] = 125,  
863 ["rbrace"] = 125,  
864 ["nbsp"] = 160,  
865 ["NonBreakingSpace"] = 160,  
866 ["iexcl"] = 161,  
867 ["cent"] = 162,  
868 ["pound"] = 163,  
869 ["curren"] = 164,  
870 ["yen"] = 165,  
871 ["brvbar"] = 166,  
872 ["sect"] = 167,  
873 ["Dot"] = 168,  
874 ["die"] = 168,  
875 ["DoubleDot"] = 168,  
876 ["uml"] = 168,  
877 ["copy"] = 169,  
878 ["COPY"] = 169,  
879 ["ordf"] = 170,  
880 ["laquo"] = 171,  
881 ["not"] = 172,  
882 ["shy"] = 173,  
883 ["reg"] = 174,  
884 ["circledR"] = 174,  
885 ["REG"] = 174,  
886 ["macr"] = 175,  
887 ["OverBar"] = 175,  
888 ["strns"] = 175,  
889 ["deg"] = 176,  
890 ["plusmn"] = 177,  
891 ["pm"] = 177,  
892 ["PlusMinus"] = 177,  
893 ["sup2"] = 178,  
894 ["sup3"] = 179,

895 ["acute"] = 180,  
896 ["DiacriticalAcute"] = 180,  
897 ["micro"] = 181,  
898 ["para"] = 182,  
899 ["middot"] = 183,  
900 ["centerdot"] = 183,  
901 ["CenterDot"] = 183,  
902 ["cedil"] = 184,  
903 ["Cedilla"] = 184,  
904 ["sup1"] = 185,  
905 ["ordm"] = 186,  
906 ["raquo"] = 187,  
907 ["frac14"] = 188,  
908 ["frac12"] = 189,  
909 ["half"] = 189,  
910 ["frac34"] = 190,  
911 ["iquest"] = 191,  
912 ["Agrave"] = 192,  
913 ["Aacute"] = 193,  
914 ["Acirc"] = 194,  
915 ["Atilde"] = 195,  
916 ["Auml"] = 196,  
917 ["Aring"] = 197,  
918 ["AElig"] = 198,  
919 ["Ccedil"] = 199,  
920 ["Egrave"] = 200,  
921 ["Eacute"] = 201,  
922 ["Ecirc"] = 202,  
923 ["Euml"] = 203,  
924 ["Igrave"] = 204,  
925 ["Iacute"] = 205,  
926 ["Icirc"] = 206,  
927 ["Iuml"] = 207,  
928 ["ETH"] = 208,  
929 ["Ntilde"] = 209,  
930 ["Ograve"] = 210,  
931 ["Oacute"] = 211,  
932 ["Ocirc"] = 212,  
933 ["Otilde"] = 213,  
934 ["Ouml"] = 214,  
935 ["times"] = 215,  
936 ["Oslash"] = 216,  
937 ["Ugrave"] = 217,  
938 ["Uacute"] = 218,  
939 ["Ucirc"] = 219,  
940 ["Uuml"] = 220,  
941 ["Yacute"] = 221,

942 ["THORN"] = 222,  
943 ["szlig"] = 223,  
944 ["agrave"] = 224,  
945 ["aacute"] = 225,  
946 ["acirc"] = 226,  
947 ["atilde"] = 227,  
948 ["auml"] = 228,  
949 ["aring"] = 229,  
950 ["aelig"] = 230,  
951 ["ccedil"] = 231,  
952 ["egrave"] = 232,  
953 ["eacute"] = 233,  
954 ["ecirc"] = 234,  
955 ["euml"] = 235,  
956 ["igrave"] = 236,  
957 ["iacute"] = 237,  
958 ["icirc"] = 238,  
959 ["iuml"] = 239,  
960 ["eth"] = 240,  
961 ["ntilde"] = 241,  
962 ["ograve"] = 242,  
963 ["oacute"] = 243,  
964 ["ocirc"] = 244,  
965 ["otilde"] = 245,  
966 ["ouml"] = 246,  
967 ["divide"] = 247,  
968 ["div"] = 247,  
969 ["oslash"] = 248,  
970 ["ugrave"] = 249,  
971 ["uacute"] = 250,  
972 ["ucirc"] = 251,  
973 ["uuml"] = 252,  
974 ["yacute"] = 253,  
975 ["thorn"] = 254,  
976 ["yuml"] = 255,  
977 ["Amacr"] = 256,  
978 ["amacr"] = 257,  
979 ["Abreve"] = 258,  
980 ["abreve"] = 259,  
981 ["Aogon"] = 260,  
982 ["aogon"] = 261,  
983 ["Cacute"] = 262,  
984 ["cacute"] = 263,  
985 ["Ccirc"] = 264,  
986 ["ccirc"] = 265,  
987 ["Cdot"] = 266,  
988 ["cdot"] = 267,

989 ["Ccaron"] = 268,  
990 ["ccaron"] = 269,  
991 ["Dcaron"] = 270,  
992 ["dcaron"] = 271,  
993 ["Dstrok"] = 272,  
994 ["dstrok"] = 273,  
995 ["Emacr"] = 274,  
996 ["emacr"] = 275,  
997 ["Edot"] = 278,  
998 ["edot"] = 279,  
999 ["Eogon"] = 280,  
1000 ["eogon"] = 281,  
1001 ["Ecaron"] = 282,  
1002 ["ecaron"] = 283,  
1003 ["Gcirc"] = 284,  
1004 ["gcirc"] = 285,  
1005 ["Gbreve"] = 286,  
1006 ["gbreve"] = 287,  
1007 ["Gdot"] = 288,  
1008 ["gdot"] = 289,  
1009 ["Gcedil"] = 290,  
1010 ["Hcirc"] = 292,  
1011 ["hcirc"] = 293,  
1012 ["Hstrok"] = 294,  
1013 ["hstrok"] = 295,  
1014 ["Itilde"] = 296,  
1015 ["itilde"] = 297,  
1016 ["Imacr"] = 298,  
1017 ["imacr"] = 299,  
1018 ["Iogon"] = 302,  
1019 ["iogon"] = 303,  
1020 ["Idot"] = 304,  
1021 ["imath"] = 305,  
1022 ["inodot"] = 305,  
1023 ["IJlig"] = 306,  
1024 ["ijlig"] = 307,  
1025 ["Jcirc"] = 308,  
1026 ["jcirc"] = 309,  
1027 ["Kcedil"] = 310,  
1028 ["kcedil"] = 311,  
1029 ["kgreen"] = 312,  
1030 ["Lacute"] = 313,  
1031 ["lacute"] = 314,  
1032 ["Lcedil"] = 315,  
1033 ["lcedil"] = 316,  
1034 ["Lcaron"] = 317,  
1035 ["lcaron"] = 318,

1036 ["Lmidot"] = 319,  
1037 ["lmidot"] = 320,  
1038 ["Lstrok"] = 321,  
1039 ["lstrok"] = 322,  
1040 ["Nacute"] = 323,  
1041 ["nacute"] = 324,  
1042 ["Ncedil"] = 325,  
1043 ["ncedil"] = 326,  
1044 ["Ncaron"] = 327,  
1045 ["ncaron"] = 328,  
1046 ["napos"] = 329,  
1047 ["ENG"] = 330,  
1048 ["eng"] = 331,  
1049 ["Omacr"] = 332,  
1050 ["omacr"] = 333,  
1051 ["Odblac"] = 336,  
1052 ["odblac"] = 337,  
1053 ["OElig"] = 338,  
1054 ["oelig"] = 339,  
1055 ["Racute"] = 340,  
1056 ["racute"] = 341,  
1057 ["Rcedil"] = 342,  
1058 ["rcedil"] = 343,  
1059 ["Rcaron"] = 344,  
1060 ["rcaron"] = 345,  
1061 ["Sacute"] = 346,  
1062 ["sacute"] = 347,  
1063 ["Scirc"] = 348,  
1064 ["scirc"] = 349,  
1065 ["Scedil"] = 350,  
1066 ["scedil"] = 351,  
1067 ["Scaron"] = 352,  
1068 ["scaron"] = 353,  
1069 ["Tcedil"] = 354,  
1070 ["tcedil"] = 355,  
1071 ["Tcaron"] = 356,  
1072 ["tcaron"] = 357,  
1073 ["Tstrok"] = 358,  
1074 ["tstrok"] = 359,  
1075 ["Utilde"] = 360,  
1076 ["utilde"] = 361,  
1077 ["Umacr"] = 362,  
1078 ["umacr"] = 363,  
1079 ["Ubreve"] = 364,  
1080 ["ubreve"] = 365,  
1081 ["Uring"] = 366,  
1082 ["uring"] = 367,

1083 ["Udblac"] = 368,  
1084 ["udblac"] = 369,  
1085 ["Uogon"] = 370,  
1086 ["uogon"] = 371,  
1087 ["Wcirc"] = 372,  
1088 ["wcirc"] = 373,  
1089 ["Ycirc"] = 374,  
1090 ["ycirc"] = 375,  
1091 ["Yuml"] = 376,  
1092 ["Zacute"] = 377,  
1093 ["zacute"] = 378,  
1094 ["Zdot"] = 379,  
1095 ["zdot"] = 380,  
1096 ["Zcaron"] = 381,  
1097 ["zcaron"] = 382,  
1098 ["fnof"] = 402,  
1099 ["imped"] = 437,  
1100 ["gacute"] = 501,  
1101 ["jmath"] = 567,  
1102 ["circ"] = 710,  
1103 ["caron"] = 711,  
1104 ["Hacek"] = 711,  
1105 ["breve"] = 728,  
1106 ["Breve"] = 728,  
1107 ["dot"] = 729,  
1108 ["DiacriticalDot"] = 729,  
1109 ["ring"] = 730,  
1110 ["ogon"] = 731,  
1111 ["tilde"] = 732,  
1112 ["DiacriticalTilde"] = 732,  
1113 ["dblac"] = 733,  
1114 ["DiacriticalDoubleAcute"] = 733,  
1115 ["DownBreve"] = 785,  
1116 ["UnderBar"] = 818,  
1117 ["Alpha"] = 913,  
1118 ["Beta"] = 914,  
1119 ["Gamma"] = 915,  
1120 ["Delta"] = 916,  
1121 ["Epsilon"] = 917,  
1122 ["Zeta"] = 918,  
1123 ["Eta"] = 919,  
1124 ["Theta"] = 920,  
1125 ["Iota"] = 921,  
1126 ["Kappa"] = 922,  
1127 ["Lambda"] = 923,  
1128 ["Mu"] = 924,  
1129 ["Nu"] = 925,

1130 ["Xi"] = 926,  
1131 ["Omicron"] = 927,  
1132 ["Pi"] = 928,  
1133 ["Rho"] = 929,  
1134 ["Sigma"] = 931,  
1135 ["Tau"] = 932,  
1136 ["Upsilon"] = 933,  
1137 ["Phi"] = 934,  
1138 ["Chi"] = 935,  
1139 ["Psi"] = 936,  
1140 ["Omega"] = 937,  
1141 ["alpha"] = 945,  
1142 ["beta"] = 946,  
1143 ["gamma"] = 947,  
1144 ["delta"] = 948,  
1145 ["epsiv"] = 949,  
1146 ["varepsilon"] = 949,  
1147 ["epsilon"] = 949,  
1148 ["zeta"] = 950,  
1149 ["eta"] = 951,  
1150 ["theta"] = 952,  
1151 ["iota"] = 953,  
1152 ["kappa"] = 954,  
1153 ["lambda"] = 955,  
1154 ["mu"] = 956,  
1155 ["nu"] = 957,  
1156 ["xi"] = 958,  
1157 ["omicron"] = 959,  
1158 ["pi"] = 960,  
1159 ["rho"] = 961,  
1160 ["sigmav"] = 962,  
1161 ["varsigma"] = 962,  
1162 ["sigmaf"] = 962,  
1163 ["sigma"] = 963,  
1164 ["tau"] = 964,  
1165 ["upsi"] = 965,  
1166 ["upsilon"] = 965,  
1167 ["phi"] = 966,  
1168 ["phiv"] = 966,  
1169 ["varphi"] = 966,  
1170 ["chi"] = 967,  
1171 ["psi"] = 968,  
1172 ["omega"] = 969,  
1173 ["thetav"] = 977,  
1174 ["vartheta"] = 977,  
1175 ["thetasym"] = 977,  
1176 ["Upsi"] = 978,



1177 ["upsih"] = 978,  
1178 ["straightphi"] = 981,  
1179 ["piv"] = 982,  
1180 ["varpi"] = 982,  
1181 ["Gammad"] = 988,  
1182 ["gammad"] = 989,  
1183 ["digamma"] = 989,  
1184 ["kappav"] = 1008,  
1185 ["varkappa"] = 1008,  
1186 ["rhov"] = 1009,  
1187 ["varrho"] = 1009,  
1188 ["epsi"] = 1013,  
1189 ["straightepsilon"] = 1013,  
1190 ["bepsi"] = 1014,  
1191 ["backepsilon"] = 1014,  
1192 ["IOcy"] = 1025,  
1193 ["DJcy"] = 1026,  
1194 ["GJcy"] = 1027,  
1195 ["Jukcy"] = 1028,  
1196 ["DScy"] = 1029,  
1197 ["Iukcy"] = 1030,  
1198 ["YIcy"] = 1031,  
1199 ["Jsercy"] = 1032,  
1200 ["LJcy"] = 1033,  
1201 ["NJcy"] = 1034,  
1202 ["TSHcy"] = 1035,  
1203 ["KJcy"] = 1036,  
1204 ["Ubrcy"] = 1038,  
1205 ["DZcy"] = 1039,  
1206 ["Acy"] = 1040,  
1207 ["Bcy"] = 1041,  
1208 ["Vcy"] = 1042,  
1209 ["Gcy"] = 1043,  
1210 ["Dcy"] = 1044,  
1211 ["IEcy"] = 1045,  
1212 ["ZHcy"] = 1046,  
1213 ["Zcy"] = 1047,  
1214 ["Icy"] = 1048,  
1215 ["Jcy"] = 1049,  
1216 ["Kcy"] = 1050,  
1217 ["Lcy"] = 1051,  
1218 ["Mcy"] = 1052,  
1219 ["Ncy"] = 1053,  
1220 ["Ocy"] = 1054,  
1221 ["Pcy"] = 1055,  
1222 ["Rcy"] = 1056,  
1223 ["Scy"] = 1057,

1224 ["Tcy"] = 1058,  
1225 ["Ucy"] = 1059,  
1226 ["Fcy"] = 1060,  
1227 ["KHcy"] = 1061,  
1228 ["TScy"] = 1062,  
1229 ["CHcy"] = 1063,  
1230 ["SHcy"] = 1064,  
1231 ["SHCHcy"] = 1065,  
1232 ["HARDcy"] = 1066,  
1233 ["Ycy"] = 1067,  
1234 ["SOFTcy"] = 1068,  
1235 ["Ecy"] = 1069,  
1236 ["YUcy"] = 1070,  
1237 ["YAcy"] = 1071,  
1238 ["acy"] = 1072,  
1239 ["bcy"] = 1073,  
1240 ["vcy"] = 1074,  
1241 ["gcy"] = 1075,  
1242 ["dcy"] = 1076,  
1243 ["iecy"] = 1077,  
1244 ["zhcy"] = 1078,  
1245 ["zcy"] = 1079,  
1246 ["icy"] = 1080,  
1247 ["jcy"] = 1081,  
1248 ["kcy"] = 1082,  
1249 ["lcy"] = 1083,  
1250 ["mcy"] = 1084,  
1251 ["ncy"] = 1085,  
1252 ["ocy"] = 1086,  
1253 ["pcy"] = 1087,  
1254 ["rcy"] = 1088,  
1255 ["scy"] = 1089,  
1256 ["tcy"] = 1090,  
1257 ["ucy"] = 1091,  
1258 ["fcy"] = 1092,  
1259 ["khcy"] = 1093,  
1260 ["tscy"] = 1094,  
1261 ["chcy"] = 1095,  
1262 ["shcy"] = 1096,  
1263 ["shchcy"] = 1097,  
1264 ["hardcy"] = 1098,  
1265 ["ycy"] = 1099,  
1266 ["softcy"] = 1100,  
1267 ["ecy"] = 1101,  
1268 ["yucy"] = 1102,  
1269 ["yacy"] = 1103,  
1270 ["iocy"] = 1105,

1271 ["djcy"] = 1106,  
1272 ["gjcy"] = 1107,  
1273 ["jukcy"] = 1108,  
1274 ["dscy"] = 1109,  
1275 ["iukcy"] = 1110,  
1276 ["yicy"] = 1111,  
1277 ["jsercy"] = 1112,  
1278 ["ljcy"] = 1113,  
1279 ["njcy"] = 1114,  
1280 ["tshcy"] = 1115,  
1281 ["kjcy"] = 1116,  
1282 ["ubrscy"] = 1118,  
1283 ["dscy"] = 1119,  
1284 ["ensp"] = 8194,  
1285 ["emsp"] = 8195,  
1286 ["emsp13"] = 8196,  
1287 ["emsp14"] = 8197,  
1288 ["numsp"] = 8199,  
1289 ["puncsp"] = 8200,  
1290 ["thinsp"] = 8201,  
1291 ["ThinSpace"] = 8201,  
1292 ["hairsp"] = 8202,  
1293 ["VeryThinSpace"] = 8202,  
1294 ["ZeroWidthSpace"] = 8203,  
1295 ["NegativeVeryThinSpace"] = 8203,  
1296 ["NegativeThinSpace"] = 8203,  
1297 ["NegativeMediumSpace"] = 8203,  
1298 ["NegativeThickSpace"] = 8203,  
1299 ["zwnj"] = 8204,  
1300 ["zwj"] = 8205,  
1301 ["lrm"] = 8206,  
1302 ["rlm"] = 8207,  
1303 ["hyphen"] = 8208,  
1304 ["dash"] = 8208,  
1305 ["ndash"] = 8211,  
1306 ["mdash"] = 8212,  
1307 ["horbar"] = 8213,  
1308 ["Verbar"] = 8214,  
1309 ["Vert"] = 8214,  
1310 ["lsquo"] = 8216,  
1311 ["OpenCurlyQuote"] = 8216,  
1312 ["rsquo"] = 8217,  
1313 ["rsquor"] = 8217,  
1314 ["CloseCurlyQuote"] = 8217,  
1315 ["lsquor"] = 8218,  
1316 ["sbquo"] = 8218,  
1317 ["ldquo"] = 8220,

1318 ["OpenCurlyDoubleQuote"] = 8220,  
1319 ["rdquo"] = 8221,  
1320 ["rdquor"] = 8221,  
1321 ["CloseCurlyDoubleQuote"] = 8221,  
1322 ["ldquor"] = 8222,  
1323 ["bdquo"] = 8222,  
1324 ["dagger"] = 8224,  
1325 ["Dagger"] = 8225,  
1326 ["ddagger"] = 8225,  
1327 ["bull"] = 8226,  
1328 ["bullet"] = 8226,  
1329 ["nldr"] = 8229,  
1330 ["hellip"] = 8230,  
1331 ["mldr"] = 8230,  
1332 ["permil"] = 8240,  
1333 ["pertenk"] = 8241,  
1334 ["prime"] = 8242,  
1335 ["Prime"] = 8243,  
1336 ["tprime"] = 8244,  
1337 ["bprime"] = 8245,  
1338 ["backprime"] = 8245,  
1339 ["lsaquo"] = 8249,  
1340 ["rsaquo"] = 8250,  
1341 ["oline"] = 8254,  
1342 ["caret"] = 8257,  
1343 ["hybull"] = 8259,  
1344 ["frasl"] = 8260,  
1345 ["bsemi"] = 8271,  
1346 ["qprime"] = 8279,  
1347 ["MediumSpace"] = 8287,  
1348 ["NoBreak"] = 8288,  
1349 ["ApplyFunction"] = 8289,  
1350 ["af"] = 8289,  
1351 ["InvisibleTimes"] = 8290,  
1352 ["it"] = 8290,  
1353 ["InvisibleComma"] = 8291,  
1354 ["ic"] = 8291,  
1355 ["euro"] = 8364,  
1356 ["tdot"] = 8411,  
1357 ["TripleDot"] = 8411,  
1358 ["DotDot"] = 8412,  
1359 ["Copf"] = 8450,  
1360 ["complexes"] = 8450,  
1361 ["incare"] = 8453,  
1362 ["gscr"] = 8458,  
1363 ["hamilt"] = 8459,  
1364 ["HilbertSpace"] = 8459,

1365 ["Hscr"] = 8459,  
1366 ["Hfr"] = 8460,  
1367 ["Poincareplane"] = 8460,  
1368 ["quaternions"] = 8461,  
1369 ["Hopf"] = 8461,  
1370 ["planckh"] = 8462,  
1371 ["planck"] = 8463,  
1372 ["hbar"] = 8463,  
1373 ["plankv"] = 8463,  
1374 ["hslash"] = 8463,  
1375 ["Iscr"] = 8464,  
1376 ["imagline"] = 8464,  
1377 ["image"] = 8465,  
1378 ["Im"] = 8465,  
1379 ["imagpart"] = 8465,  
1380 ["Ifr"] = 8465,  
1381 ["Lscr"] = 8466,  
1382 ["lagran"] = 8466,  
1383 ["Laplacetrif"] = 8466,  
1384 ["ell"] = 8467,  
1385 ["Nopf"] = 8469,  
1386 ["naturals"] = 8469,  
1387 ["numero"] = 8470,  
1388 ["copysr"] = 8471,  
1389 ["weierp"] = 8472,  
1390 ["wp"] = 8472,  
1391 ["Popf"] = 8473,  
1392 ["primes"] = 8473,  
1393 ["rationals"] = 8474,  
1394 ["Qopf"] = 8474,  
1395 ["Rscr"] = 8475,  
1396 ["realine"] = 8475,  
1397 ["real"] = 8476,  
1398 ["Re"] = 8476,  
1399 ["realpart"] = 8476,  
1400 ["Rfr"] = 8476,  
1401 ["reals"] = 8477,  
1402 ["Ropf"] = 8477,  
1403 ["rx"] = 8478,  
1404 ["trade"] = 8482,  
1405 ["TRADE"] = 8482,  
1406 ["integers"] = 8484,  
1407 ["Zopf"] = 8484,  
1408 ["ohm"] = 8486,  
1409 ["mho"] = 8487,  
1410 ["Zfr"] = 8488,  
1411 ["zeetrf"] = 8488,

1412 ["iota"] = 8489,  
1413 ["angst"] = 8491,  
1414 ["bernou"] = 8492,  
1415 ["Bernoullis"] = 8492,  
1416 ["Bscr"] = 8492,  
1417 ["Cfr"] = 8493,  
1418 ["Cayleys"] = 8493,  
1419 ["escr"] = 8495,  
1420 ["Escr"] = 8496,  
1421 ["expectation"] = 8496,  
1422 ["Fscr"] = 8497,  
1423 ["Fouriertrf"] = 8497,  
1424 ["phmmat"] = 8499,  
1425 ["Mellintrf"] = 8499,  
1426 ["Mscr"] = 8499,  
1427 ["order"] = 8500,  
1428 ["orderof"] = 8500,  
1429 ["oscr"] = 8500,  
1430 ["alefsym"] = 8501,  
1431 ["aleph"] = 8501,  
1432 ["beth"] = 8502,  
1433 ["gimel"] = 8503,  
1434 ["daleth"] = 8504,  
1435 ["CapitalDifferentialD"] = 8517,  
1436 ["DD"] = 8517,  
1437 ["DifferentialD"] = 8518,  
1438 ["dd"] = 8518,  
1439 ["ExponentialE"] = 8519,  
1440 ["exponentiale"] = 8519,  
1441 ["ee"] = 8519,  
1442 ["ImaginaryI"] = 8520,  
1443 ["ii"] = 8520,  
1444 ["frac13"] = 8531,  
1445 ["frac23"] = 8532,  
1446 ["frac15"] = 8533,  
1447 ["frac25"] = 8534,  
1448 ["frac35"] = 8535,  
1449 ["frac45"] = 8536,  
1450 ["frac16"] = 8537,  
1451 ["frac56"] = 8538,  
1452 ["frac18"] = 8539,  
1453 ["frac38"] = 8540,  
1454 ["frac58"] = 8541,  
1455 ["frac78"] = 8542,  
1456 ["larr"] = 8592,  
1457 ["leftarrow"] = 8592,  
1458 ["LeftArrow"] = 8592,

```

1459 ["slarr"] = 8592,
1460 ["ShortLeftArrow"] = 8592,
1461 ["uarr"] = 8593,
1462 ["uparrow"] = 8593,
1463 ["UpArrow"] = 8593,
1464 ["ShortUpArrow"] = 8593,
1465 ["rarr"] = 8594,
1466 ["rightarrow"] = 8594,
1467 ["RightArrow"] = 8594,
1468 ["srarr"] = 8594,
1469 ["ShortRightArrow"] = 8594,
1470 ["darr"] = 8595,
1471 ["downarrow"] = 8595,
1472 ["DownArrow"] = 8595,
1473 ["ShortDownArrow"] = 8595,
1474 ["harr"] = 8596,
1475 ["leftrightarrow"] = 8596,
1476 ["LeftRightArrow"] = 8596,
1477 ["varr"] = 8597,
1478 ["updownarrow"] = 8597,
1479 ["UpDownArrow"] = 8597,
1480 ["nwarr"] = 8598,
1481 ["UpperLeftArrow"] = 8598,
1482 ["nwarrow"] = 8598,
1483 ["nearr"] = 8599,
1484 ["UpperRightArrow"] = 8599,
1485 ["nearrow"] = 8599,
1486 ["searr"] = 8600,
1487 ["searrow"] = 8600,
1488 ["LowerRightArrow"] = 8600,
1489 ["swarr"] = 8601,
1490 ["swarrow"] = 8601,
1491 ["LowerLeftArrow"] = 8601,
1492 ["nlarr"] = 8602,
1493 ["nleftarrow"] = 8602,
1494 ["nrarr"] = 8603,
1495 ["nrightarrow"] = 8603,
1496 ["rarrw"] = 8605,
1497 ["rightsquigarrow"] = 8605,
1498 ["Larr"] = 8606,
1499 ["twoheadleftarrow"] = 8606,
1500 ["Uarr"] = 8607,
1501 ["Rarr"] = 8608,
1502 ["twoheadrightarrow"] = 8608,
1503 ["Darr"] = 8609,
1504 ["larrrtl"] = 8610,
1505 ["leftarrowtail"] = 8610,

```

1506 ["rarrtl"] = 8611,  
1507 ["rightarrowtail"] = 8611,  
1508 ["LeftTeeArrow"] = 8612,  
1509 ["mapstoleft"] = 8612,  
1510 ["UpTeeArrow"] = 8613,  
1511 ["mapstoup"] = 8613,  
1512 ["map"] = 8614,  
1513 ["RightTeeArrow"] = 8614,  
1514 ["mapsto"] = 8614,  
1515 ["DownTeeArrow"] = 8615,  
1516 ["mapstodown"] = 8615,  
1517 ["larrhk"] = 8617,  
1518 ["hookleftarrow"] = 8617,  
1519 ["rarrhk"] = 8618,  
1520 ["hookrightarrow"] = 8618,  
1521 ["larrlp"] = 8619,  
1522 ["looparrowleft"] = 8619,  
1523 ["rarrlp"] = 8620,  
1524 ["looparrowright"] = 8620,  
1525 ["harrw"] = 8621,  
1526 ["leftrightsquigarrow"] = 8621,  
1527 ["nharr"] = 8622,  
1528 ["nleftrightarrow"] = 8622,  
1529 ["lsh"] = 8624,  
1530 ["Lsh"] = 8624,  
1531 ["rsh"] = 8625,  
1532 ["Rsh"] = 8625,  
1533 ["ldsh"] = 8626,  
1534 ["rdsh"] = 8627,  
1535 ["crarr"] = 8629,  
1536 ["cularr"] = 8630,  
1537 ["curvearrowleft"] = 8630,  
1538 ["curarr"] = 8631,  
1539 ["curvearrowright"] = 8631,  
1540 ["olarr"] = 8634,  
1541 ["circlearrowleft"] = 8634,  
1542 ["orarr"] = 8635,  
1543 ["circlearrowright"] = 8635,  
1544 ["lharu"] = 8636,  
1545 ["LeftVector"] = 8636,  
1546 ["leftharpoonup"] = 8636,  
1547 ["lhard"] = 8637,  
1548 ["leftharpoondown"] = 8637,  
1549 ["DownLeftVector"] = 8637,  
1550 ["uharr"] = 8638,  
1551 ["upharpoonright"] = 8638,  
1552 ["RightUpVector"] = 8638,



1553 ["uharl"] = 8639,  
1554 ["upharpoonleft"] = 8639,  
1555 ["LeftUpVector"] = 8639,  
1556 ["rharu"] = 8640,  
1557 ["RightVector"] = 8640,  
1558 ["rightharpoonup"] = 8640,  
1559 ["rhard"] = 8641,  
1560 ["rightharpoondown"] = 8641,  
1561 ["DownRightVector"] = 8641,  
1562 ["dharr"] = 8642,  
1563 ["RightDownVector"] = 8642,  
1564 ["downharpoonright"] = 8642,  
1565 ["dharl"] = 8643,  
1566 ["LeftDownVector"] = 8643,  
1567 ["downharpoonleft"] = 8643,  
1568 ["rlarr"] = 8644,  
1569 ["rightleftarrows"] = 8644,  
1570 ["RightArrowLeftArrow"] = 8644,  
1571 ["udarr"] = 8645,  
1572 ["UpArrowDownArrow"] = 8645,  
1573 ["lrarr"] = 8646,  
1574 ["leftrightarrows"] = 8646,  
1575 ["LeftArrowRightArrow"] = 8646,  
1576 ["llarr"] = 8647,  
1577 ["leftleftarrows"] = 8647,  
1578 ["uuarr"] = 8648,  
1579 ["upuparrows"] = 8648,  
1580 ["rrarr"] = 8649,  
1581 ["rightrightarrows"] = 8649,  
1582 ["ddarr"] = 8650,  
1583 ["downdownarrows"] = 8650,  
1584 ["lrhar"] = 8651,  
1585 ["ReverseEquilibrium"] = 8651,  
1586 ["leftrightharpoons"] = 8651,  
1587 ["rlhar"] = 8652,  
1588 ["rightleftharpoons"] = 8652,  
1589 ["Equilibrium"] = 8652,  
1590 ["nLArr"] = 8653,  
1591 ["nLeftarrow"] = 8653,  
1592 ["nhArr"] = 8654,  
1593 ["nLeftrightarrow"] = 8654,  
1594 ["nrArr"] = 8655,  
1595 ["nRightarrow"] = 8655,  
1596 ["lArr"] = 8656,  
1597 ["Leftarrow"] = 8656,  
1598 ["DoubleLeftArrow"] = 8656,  
1599 ["uArr"] = 8657,

1600 ["Uparrow"] = 8657,  
1601 ["DoubleUpArrow"] = 8657,  
1602 ["rArr"] = 8658,  
1603 ["Rightarrow"] = 8658,  
1604 ["Implies"] = 8658,  
1605 ["DoubleRightArrow"] = 8658,  
1606 ["dArr"] = 8659,  
1607 ["Downarrow"] = 8659,  
1608 ["DoubleDownArrow"] = 8659,  
1609 ["hArr"] = 8660,  
1610 ["Leftrightarrow"] = 8660,  
1611 ["DoubleLeftRightArrow"] = 8660,  
1612 ["iff"] = 8660,  
1613 ["vArr"] = 8661,  
1614 ["Updownarrow"] = 8661,  
1615 ["DoubleUpDownArrow"] = 8661,  
1616 ["nwArr"] = 8662,  
1617 ["neArr"] = 8663,  
1618 ["seArr"] = 8664,  
1619 ["swArr"] = 8665,  
1620 ["lAarr"] = 8666,  
1621 ["Lleftarrow"] = 8666,  
1622 ["rAarr"] = 8667,  
1623 ["Rrightarrow"] = 8667,  
1624 ["zigrarr"] = 8669,  
1625 ["larrb"] = 8676,  
1626 ["LeftArrowBar"] = 8676,  
1627 ["rarrb"] = 8677,  
1628 ["RightArrowBar"] = 8677,  
1629 ["duarr"] = 8693,  
1630 ["DownArrowUpArrow"] = 8693,  
1631 ["loarr"] = 8701,  
1632 ["roarr"] = 8702,  
1633 ["hoarr"] = 8703,  
1634 ["forall"] = 8704,  
1635 ["ForAll"] = 8704,  
1636 ["comp"] = 8705,  
1637 ["complement"] = 8705,  
1638 ["part"] = 8706,  
1639 ["PartialD"] = 8706,  
1640 ["exist"] = 8707,  
1641 ["Exists"] = 8707,  
1642 ["nexist"] = 8708,  
1643 ["NotExists"] = 8708,  
1644 ["nexists"] = 8708,  
1645 ["empty"] = 8709,  
1646 ["emptyset"] = 8709,

1647 ["emptyv"] = 8709,  
 1648 ["varnothing"] = 8709,  
 1649 ["nabla"] = 8711,  
 1650 ["Del"] = 8711,  
 1651 ["isin"] = 8712,  
 1652 ["isinv"] = 8712,  
 1653 ["Element"] = 8712,  
 1654 ["in"] = 8712,  
 1655 ["notin"] = 8713,  
 1656 ["NotElement"] = 8713,  
 1657 ["notinva"] = 8713,  
 1658 ["niv"] = 8715,  
 1659 ["ReverseElement"] = 8715,  
 1660 ["ni"] = 8715,  
 1661 ["SuchThat"] = 8715,  
 1662 ["notni"] = 8716,  
 1663 ["notniva"] = 8716,  
 1664 ["NotReverseElement"] = 8716,  
 1665 ["prod"] = 8719,  
 1666 ["Product"] = 8719,  
 1667 ["coprod"] = 8720,  
 1668 ["Coproduct"] = 8720,  
 1669 ["sum"] = 8721,  
 1670 ["Sum"] = 8721,  
 1671 ["minus"] = 8722,  
 1672 ["mnplus"] = 8723,  
 1673 ["mp"] = 8723,  
 1674 ["MinusPlus"] = 8723,  
 1675 ["plusdo"] = 8724,  
 1676 ["dotplus"] = 8724,  
 1677 ["setmn"] = 8726,  
 1678 ["setminus"] = 8726,  
 1679 ["Backslash"] = 8726,  
 1680 ["ssetmn"] = 8726,  
 1681 ["smallsetminus"] = 8726,  
 1682 ["lowast"] = 8727,  
 1683 ["compfn"] = 8728,  
 1684 ["SmallCircle"] = 8728,  
 1685 ["radic"] = 8730,  
 1686 ["Sqrt"] = 8730,  
 1687 ["prop"] = 8733,  
 1688 ["propto"] = 8733,  
 1689 ["Proportional"] = 8733,  
 1690 ["vprop"] = 8733,  
 1691 ["varpropto"] = 8733,  
 1692 ["infin"] = 8734,  
 1693 ["angrt"] = 8735,

1694 ["ang"] = 8736,  
1695 ["angle"] = 8736,  
1696 ["angmsd"] = 8737,  
1697 ["measuredangle"] = 8737,  
1698 ["angsph"] = 8738,  
1699 ["mid"] = 8739,  
1700 ["VerticalBar"] = 8739,  
1701 ["smid"] = 8739,  
1702 ["shortmid"] = 8739,  
1703 ["nmid"] = 8740,  
1704 ["NotVerticalBar"] = 8740,  
1705 ["nsmid"] = 8740,  
1706 ["nshortmid"] = 8740,  
1707 ["par"] = 8741,  
1708 ["parallel"] = 8741,  
1709 ["DoubleVerticalBar"] = 8741,  
1710 ["spar"] = 8741,  
1711 ["shortparallel"] = 8741,  
1712 ["npar"] = 8742,  
1713 ["nparallel"] = 8742,  
1714 ["NotDoubleVerticalBar"] = 8742,  
1715 ["nspar"] = 8742,  
1716 ["nshortparallel"] = 8742,  
1717 ["and"] = 8743,  
1718 ["wedge"] = 8743,  
1719 ["or"] = 8744,  
1720 ["vee"] = 8744,  
1721 ["cap"] = 8745,  
1722 ["cup"] = 8746,  
1723 ["int"] = 8747,  
1724 ["Integral"] = 8747,  
1725 ["Int"] = 8748,  
1726 ["tint"] = 8749,  
1727 ["iiint"] = 8749,  
1728 ["conint"] = 8750,  
1729 ["oint"] = 8750,  
1730 ["ContourIntegral"] = 8750,  
1731 ["Conint"] = 8751,  
1732 ["DoubleContourIntegral"] = 8751,  
1733 ["Cconint"] = 8752,  
1734 ["cwint"] = 8753,  
1735 ["cwconint"] = 8754,  
1736 ["ClockwiseContourIntegral"] = 8754,  
1737 ["awconint"] = 8755,  
1738 ["CounterClockwiseContourIntegral"] = 8755,  
1739 ["there4"] = 8756,  
1740 ["therefore"] = 8756,

1741 ["Therefore"] = 8756,  
1742 ["becaus"] = 8757,  
1743 ["because"] = 8757,  
1744 ["Because"] = 8757,  
1745 ["ratio"] = 8758,  
1746 ["Colon"] = 8759,  
1747 ["Proportion"] = 8759,  
1748 ["minusd"] = 8760,  
1749 ["dotminus"] = 8760,  
1750 ["mDDot"] = 8762,  
1751 ["homtht"] = 8763,  
1752 ["sim"] = 8764,  
1753 ["Tilde"] = 8764,  
1754 ["thksim"] = 8764,  
1755 ["thicksim"] = 8764,  
1756 ["bsim"] = 8765,  
1757 ["backsim"] = 8765,  
1758 ["ac"] = 8766,  
1759 ["mstpos"] = 8766,  
1760 ["acd"] = 8767,  
1761 ["wreath"] = 8768,  
1762 ["VerticalTilde"] = 8768,  
1763 ["wr"] = 8768,  
1764 ["nsim"] = 8769,  
1765 ["NotTilde"] = 8769,  
1766 ["esim"] = 8770,  
1767 ["EqualTilde"] = 8770,  
1768 ["eqsim"] = 8770,  
1769 ["sime"] = 8771,  
1770 ["TildeEqual"] = 8771,  
1771 ["simeq"] = 8771,  
1772 ["nsime"] = 8772,  
1773 ["nsimeq"] = 8772,  
1774 ["NotTildeEqual"] = 8772,  
1775 ["cong"] = 8773,  
1776 ["TildeFullEqual"] = 8773,  
1777 ["simne"] = 8774,  
1778 ["ncong"] = 8775,  
1779 ["NotTildeFullEqual"] = 8775,  
1780 ["asymp"] = 8776,  
1781 ["ap"] = 8776,  
1782 ["TildeTilde"] = 8776,  
1783 ["approx"] = 8776,  
1784 ["thkap"] = 8776,  
1785 ["thickapprox"] = 8776,  
1786 ["nap"] = 8777,  
1787 ["NotTildeTilde"] = 8777,

1788 ["napprox"] = 8777,  
1789 ["ape"] = 8778,  
1790 ["approxpeq"] = 8778,  
1791 ["apid"] = 8779,  
1792 ["bcong"] = 8780,  
1793 ["backcong"] = 8780,  
1794 ["asympeq"] = 8781,  
1795 ["CupCap"] = 8781,  
1796 ["bump"] = 8782,  
1797 ["HumpDownHump"] = 8782,  
1798 ["Bumpeq"] = 8782,  
1799 ["bumpe"] = 8783,  
1800 ["HumpEqual"] = 8783,  
1801 ["bumpeq"] = 8783,  
1802 ["esdot"] = 8784,  
1803 ["DotEqual"] = 8784,  
1804 ["doteq"] = 8784,  
1805 ["eDot"] = 8785,  
1806 ["doteqdot"] = 8785,  
1807 ["efDot"] = 8786,  
1808 ["fallingdotseq"] = 8786,  
1809 ["erDot"] = 8787,  
1810 ["risingdotseq"] = 8787,  
1811 ["colone"] = 8788,  
1812 ["coloneq"] = 8788,  
1813 ["Assign"] = 8788,  
1814 ["ecolon"] = 8789,  
1815 ["eqcolon"] = 8789,  
1816 ["ecir"] = 8790,  
1817 ["eqcirc"] = 8790,  
1818 ["cire"] = 8791,  
1819 ["circeq"] = 8791,  
1820 ["wedgeq"] = 8793,  
1821 ["veeeq"] = 8794,  
1822 ["trie"] = 8796,  
1823 ["triangleq"] = 8796,  
1824 ["equest"] = 8799,  
1825 ["questeq"] = 8799,  
1826 ["ne"] = 8800,  
1827 ["NotEqual"] = 8800,  
1828 ["equiv"] = 8801,  
1829 ["Congruent"] = 8801,  
1830 ["nequiv"] = 8802,  
1831 ["NotCongruent"] = 8802,  
1832 ["le"] = 8804,  
1833 ["leq"] = 8804,  
1834 ["ge"] = 8805,

1835 ["GreaterEqual"] = 8805,  
1836 ["geq"] = 8805,  
1837 ["lE"] = 8806,  
1838 ["LessFullEqual"] = 8806,  
1839 ["leqq"] = 8806,  
1840 ["gE"] = 8807,  
1841 ["GreaterFullEqual"] = 8807,  
1842 ["geqq"] = 8807,  
1843 ["lnE"] = 8808,  
1844 ["lneqq"] = 8808,  
1845 ["gnE"] = 8809,  
1846 ["gneqq"] = 8809,  
1847 ["Lt"] = 8810,  
1848 ["NestedLessLess"] = 8810,  
1849 ["ll"] = 8810,  
1850 ["Gt"] = 8811,  
1851 ["NestedGreaterGreater"] = 8811,  
1852 ["gg"] = 8811,  
1853 ["twixt"] = 8812,  
1854 ["between"] = 8812,  
1855 ["NotCupCap"] = 8813,  
1856 ["nlt"] = 8814,  
1857 ["NotLess"] = 8814,  
1858 ["nless"] = 8814,  
1859 ["ngt"] = 8815,  
1860 ["NotGreater"] = 8815,  
1861 ["ngtr"] = 8815,  
1862 ["nle"] = 8816,  
1863 ["NotLessEqual"] = 8816,  
1864 ["nleq"] = 8816,  
1865 ["nge"] = 8817,  
1866 ["NotGreaterEqual"] = 8817,  
1867 ["ngeq"] = 8817,  
1868 ["lsim"] = 8818,  
1869 ["LessTilde"] = 8818,  
1870 ["lesssim"] = 8818,  
1871 ["gsim"] = 8819,  
1872 ["gtrsim"] = 8819,  
1873 ["GreaterTilde"] = 8819,  
1874 ["nlsim"] = 8820,  
1875 ["NotLessTilde"] = 8820,  
1876 ["ngsim"] = 8821,  
1877 ["NotGreaterTilde"] = 8821,  
1878 ["lg"] = 8822,  
1879 ["lessgtr"] = 8822,  
1880 ["LessGreater"] = 8822,  
1881 ["gl"] = 8823,

1882 ["gtrless"] = 8823,  
 1883 ["GreaterLess"] = 8823,  
 1884 ["ntlgt"] = 8824,  
 1885 ["NotLessGreater"] = 8824,  
 1886 ["ntgl"] = 8825,  
 1887 ["NotGreaterLess"] = 8825,  
 1888 ["pr"] = 8826,  
 1889 ["Precedes"] = 8826,  
 1890 ["prec"] = 8826,  
 1891 ["sc"] = 8827,  
 1892 ["Succeeds"] = 8827,  
 1893 ["succ"] = 8827,  
 1894 ["prcue"] = 8828,  
 1895 ["PrecedesSlantEqual"] = 8828,  
 1896 ["preccurlyeq"] = 8828,  
 1897 ["sccue"] = 8829,  
 1898 ["SucceedsSlantEqual"] = 8829,  
 1899 ["succurlyeq"] = 8829,  
 1900 ["prsim"] = 8830,  
 1901 ["precsim"] = 8830,  
 1902 ["PrecedesTilde"] = 8830,  
 1903 ["scsim"] = 8831,  
 1904 ["succsim"] = 8831,  
 1905 ["SucceedsTilde"] = 8831,  
 1906 ["npr"] = 8832,  
 1907 ["nprec"] = 8832,  
 1908 ["NotPrecedes"] = 8832,  
 1909 ["nsc"] = 8833,  
 1910 ["nsucc"] = 8833,  
 1911 ["NotSucceeds"] = 8833,  
 1912 ["sub"] = 8834,  
 1913 ["subset"] = 8834,  
 1914 ["sup"] = 8835,  
 1915 ["supset"] = 8835,  
 1916 ["Superset"] = 8835,  
 1917 ["nsub"] = 8836,  
 1918 ["nsup"] = 8837,  
 1919 ["sube"] = 8838,  
 1920 ["SubsetEqual"] = 8838,  
 1921 ["subseteq"] = 8838,  
 1922 ["supe"] = 8839,  
 1923 ["supseteq"] = 8839,  
 1924 ["SupersetEqual"] = 8839,  
 1925 ["nsube"] = 8840,  
 1926 ["nsubseteq"] = 8840,  
 1927 ["NotSubsetEqual"] = 8840,  
 1928 ["nsupe"] = 8841,



1929 ["nsupseteq"] = 8841,  
 1930 ["NotSupersetEqual"] = 8841,  
 1931 ["subne"] = 8842,  
 1932 ["subsetneq"] = 8842,  
 1933 ["supne"] = 8843,  
 1934 ["supsetneq"] = 8843,  
 1935 ["cupdot"] = 8845,  
 1936 ["uplus"] = 8846,  
 1937 ["UnionPlus"] = 8846,  
 1938 ["sqsub"] = 8847,  
 1939 ["SquareSubset"] = 8847,  
 1940 ["sqsubset"] = 8847,  
 1941 ["sqsup"] = 8848,  
 1942 ["SquareSuperset"] = 8848,  
 1943 ["sqsupset"] = 8848,  
 1944 ["sqsube"] = 8849,  
 1945 ["SquareSubsetEqual"] = 8849,  
 1946 ["sqsubseteq"] = 8849,  
 1947 ["sqsupe"] = 8850,  
 1948 ["SquareSupersetEqual"] = 8850,  
 1949 ["sqsupseteq"] = 8850,  
 1950 ["sqcap"] = 8851,  
 1951 ["SquareIntersection"] = 8851,  
 1952 ["sqcup"] = 8852,  
 1953 ["SquareUnion"] = 8852,  
 1954 ["oplus"] = 8853,  
 1955 ["CirclePlus"] = 8853,  
 1956 ["ominus"] = 8854,  
 1957 ["CircleMinus"] = 8854,  
 1958 ["otimes"] = 8855,  
 1959 ["CircleTimes"] = 8855,  
 1960 ["osol"] = 8856,  
 1961 ["odot"] = 8857,  
 1962 ["CircleDot"] = 8857,  
 1963 ["ocir"] = 8858,  
 1964 ["circledcirc"] = 8858,  
 1965 ["oast"] = 8859,  
 1966 ["circledast"] = 8859,  
 1967 ["odash"] = 8861,  
 1968 ["circleddash"] = 8861,  
 1969 ["plusb"] = 8862,  
 1970 ["boxplus"] = 8862,  
 1971 ["minusb"] = 8863,  
 1972 ["boxminus"] = 8863,  
 1973 ["timesb"] = 8864,  
 1974 ["boxtimes"] = 8864,  
 1975 ["sdotb"] = 8865,

1976 ["dotsquare"] = 8865,  
1977 ["vdash"] = 8866,  
1978 ["RightTee"] = 8866,  
1979 ["dashv"] = 8867,  
1980 ["LeftTee"] = 8867,  
1981 ["top"] = 8868,  
1982 ["DownTee"] = 8868,  
1983 ["bottom"] = 8869,  
1984 ["bot"] = 8869,  
1985 ["perp"] = 8869,  
1986 ["UpTee"] = 8869,  
1987 ["models"] = 8871,  
1988 ["vDash"] = 8872,  
1989 ["DoubleRightTee"] = 8872,  
1990 ["Vdash"] = 8873,  
1991 ["Vvdash"] = 8874,  
1992 ["VDash"] = 8875,  
1993 ["nvdash"] = 8876,  
1994 ["nvDash"] = 8877,  
1995 ["nVdash"] = 8878,  
1996 ["nVDash"] = 8879,  
1997 ["prurel"] = 8880,  
1998 ["vltri"] = 8882,  
1999 ["vartriangleleft"] = 8882,  
2000 ["LeftTriangle"] = 8882,  
2001 ["vrtri"] = 8883,  
2002 ["vartriangleright"] = 8883,  
2003 ["RightTriangle"] = 8883,  
2004 ["ltrie"] = 8884,  
2005 ["trianglelefteq"] = 8884,  
2006 ["LeftTriangleEqual"] = 8884,  
2007 ["rtrie"] = 8885,  
2008 ["trianglerighteq"] = 8885,  
2009 ["RightTriangleEqual"] = 8885,  
2010 ["origof"] = 8886,  
2011 ["imof"] = 8887,  
2012 ["mumap"] = 8888,  
2013 ["multimap"] = 8888,  
2014 ["hercon"] = 8889,  
2015 ["intcal"] = 8890,  
2016 ["intercal"] = 8890,  
2017 ["veebar"] = 8891,  
2018 ["barvee"] = 8893,  
2019 ["angrtvb"] = 8894,  
2020 ["lrtri"] = 8895,  
2021 ["xwedge"] = 8896,  
2022 ["Wedge"] = 8896,

2023 ["bigwedge"] = 8896,  
 2024 ["xvee"] = 8897,  
 2025 ["Vee"] = 8897,  
 2026 ["bigvee"] = 8897,  
 2027 ["xcap"] = 8898,  
 2028 ["Intersection"] = 8898,  
 2029 ["bigcap"] = 8898,  
 2030 ["xcup"] = 8899,  
 2031 ["Union"] = 8899,  
 2032 ["bigcup"] = 8899,  
 2033 ["diam"] = 8900,  
 2034 ["diamond"] = 8900,  
 2035 ["Diamond"] = 8900,  
 2036 ["sdot"] = 8901,  
 2037 ["sstarf"] = 8902,  
 2038 ["Star"] = 8902,  
 2039 ["divonx"] = 8903,  
 2040 ["divideontimes"] = 8903,  
 2041 ["bowtie"] = 8904,  
 2042 ["ltimes"] = 8905,  
 2043 ["rtimes"] = 8906,  
 2044 ["lthree"] = 8907,  
 2045 ["leftthreetimes"] = 8907,  
 2046 ["rthree"] = 8908,  
 2047 ["rightthreetimes"] = 8908,  
 2048 ["bsime"] = 8909,  
 2049 ["backsim"] = 8909,  
 2050 ["cuvee"] = 8910,  
 2051 ["curlyvee"] = 8910,  
 2052 ["cuwed"] = 8911,  
 2053 ["curlywedge"] = 8911,  
 2054 ["Sub"] = 8912,  
 2055 ["Subset"] = 8912,  
 2056 ["Sup"] = 8913,  
 2057 ["Supset"] = 8913,  
 2058 ["Cap"] = 8914,  
 2059 ["Cup"] = 8915,  
 2060 ["fork"] = 8916,  
 2061 ["pitchfork"] = 8916,  
 2062 ["epar"] = 8917,  
 2063 ["ltdot"] = 8918,  
 2064 ["lessdot"] = 8918,  
 2065 ["gtdot"] = 8919,  
 2066 ["gtrdot"] = 8919,  
 2067 ["Ll"] = 8920,  
 2068 ["Gg"] = 8921,  
 2069 ["ggg"] = 8921,

2070 ["leg"] = 8922,  
 2071 ["LessEqualGreater"] = 8922,  
 2072 ["lesseqgtr"] = 8922,  
 2073 ["gel"] = 8923,  
 2074 ["gtreqless"] = 8923,  
 2075 ["GreaterEqualLess"] = 8923,  
 2076 ["cuepr"] = 8926,  
 2077 ["curlyeqprec"] = 8926,  
 2078 ["cuesc"] = 8927,  
 2079 ["curlyeqsucc"] = 8927,  
 2080 ["nprcue"] = 8928,  
 2081 ["NotPrecedesSlantEqual"] = 8928,  
 2082 ["nsccue"] = 8929,  
 2083 ["NotSucceedsSlantEqual"] = 8929,  
 2084 ["nsqsube"] = 8930,  
 2085 ["NotSquareSubsetEqual"] = 8930,  
 2086 ["nsqsupe"] = 8931,  
 2087 ["NotSquareSupersetEqual"] = 8931,  
 2088 ["lnsim"] = 8934,  
 2089 ["gnsim"] = 8935,  
 2090 ["prnsim"] = 8936,  
 2091 ["precnsim"] = 8936,  
 2092 ["scnsim"] = 8937,  
 2093 ["succnsim"] = 8937,  
 2094 ["nltri"] = 8938,  
 2095 ["ntriangleleft"] = 8938,  
 2096 ["NotLeftTriangle"] = 8938,  
 2097 ["nrtri"] = 8939,  
 2098 ["ntriangleright"] = 8939,  
 2099 ["NotRightTriangle"] = 8939,  
 2100 ["nltrie"] = 8940,  
 2101 ["ntrianglelefteq"] = 8940,  
 2102 ["NotLeftTriangleEqual"] = 8940,  
 2103 ["nrtrie"] = 8941,  
 2104 ["ntrianglerighteq"] = 8941,  
 2105 ["NotRightTriangleEqual"] = 8941,  
 2106 ["vellip"] = 8942,  
 2107 ["ctdot"] = 8943,  
 2108 ["utdot"] = 8944,  
 2109 ["dtdot"] = 8945,  
 2110 ["disin"] = 8946,  
 2111 ["isinsv"] = 8947,  
 2112 ["isins"] = 8948,  
 2113 ["isindot"] = 8949,  
 2114 ["notinvc"] = 8950,  
 2115 ["notinvb"] = 8951,  
 2116 ["isinE"] = 8953,

2117 ["nisd"] = 8954,  
2118 ["xnis"] = 8955,  
2119 ["nis"] = 8956,  
2120 ["notnivc"] = 8957,  
2121 ["notnivb"] = 8958,  
2122 ["barwed"] = 8965,  
2123 ["barwedge"] = 8965,  
2124 ["Barwed"] = 8966,  
2125 ["doublebarwedge"] = 8966,  
2126 ["lceil"] = 8968,  
2127 ["LeftCeiling"] = 8968,  
2128 ["rceil"] = 8969,  
2129 ["RightCeiling"] = 8969,  
2130 ["lfloor"] = 8970,  
2131 ["LeftFloor"] = 8970,  
2132 ["rfloor"] = 8971,  
2133 ["RightFloor"] = 8971,  
2134 ["drcrop"] = 8972,  
2135 ["dlcrop"] = 8973,  
2136 ["urcrop"] = 8974,  
2137 ["ulcrop"] = 8975,  
2138 ["bnot"] = 8976,  
2139 ["proflines"] = 8978,  
2140 ["profsurf"] = 8979,  
2141 ["telrec"] = 8981,  
2142 ["target"] = 8982,  
2143 ["ulcorn"] = 8988,  
2144 ["ulcorner"] = 8988,  
2145 ["urcorn"] = 8989,  
2146 ["urcorner"] = 8989,  
2147 ["dlcorn"] = 8990,  
2148 ["llcorner"] = 8990,  
2149 ["drcorn"] = 8991,  
2150 ["lrcorn"] = 8991,  
2151 ["frown"] = 8994,  
2152 ["sfrown"] = 8994,  
2153 ["smile"] = 8995,  
2154 ["ssmile"] = 8995,  
2155 ["cylcty"] = 9005,  
2156 ["profalar"] = 9006,  
2157 ["topbot"] = 9014,  
2158 ["ovbar"] = 9021,  
2159 ["solbar"] = 9023,  
2160 ["angzarr"] = 9084,  
2161 ["lmoust"] = 9136,  
2162 ["lmoustache"] = 9136,  
2163 ["rmoust"] = 9137,

2164 ["rmoustache"] = 9137,  
2165 ["tbrk"] = 9140,  
2166 ["OverBracket"] = 9140,  
2167 ["bbrk"] = 9141,  
2168 ["UnderBracket"] = 9141,  
2169 ["bbrktbrk"] = 9142,  
2170 ["OverParenthesis"] = 9180,  
2171 ["UnderParenthesis"] = 9181,  
2172 ["OverBrace"] = 9182,  
2173 ["UnderBrace"] = 9183,  
2174 ["trpezium"] = 9186,  
2175 ["elinters"] = 9191,  
2176 ["blank"] = 9251,  
2177 ["oS"] = 9416,  
2178 ["circledS"] = 9416,  
2179 ["boxh"] = 9472,  
2180 ["HorizontalLine"] = 9472,  
2181 ["boxv"] = 9474,  
2182 ["boxdr"] = 9484,  
2183 ["boxdl"] = 9488,  
2184 ["boxur"] = 9492,  
2185 ["boxul"] = 9496,  
2186 ["boxvr"] = 9500,  
2187 ["boxvl"] = 9508,  
2188 ["boxhd"] = 9516,  
2189 ["boxhu"] = 9524,  
2190 ["boxvh"] = 9532,  
2191 ["boxH"] = 9552,  
2192 ["boxV"] = 9553,  
2193 ["boxdR"] = 9554,  
2194 ["boxDr"] = 9555,  
2195 ["boxDR"] = 9556,  
2196 ["boxdL"] = 9557,  
2197 ["boxDl"] = 9558,  
2198 ["boxDL"] = 9559,  
2199 ["boxuR"] = 9560,  
2200 ["boxUr"] = 9561,  
2201 ["boxUR"] = 9562,  
2202 ["boxuL"] = 9563,  
2203 ["boxUl"] = 9564,  
2204 ["boxUL"] = 9565,  
2205 ["boxvR"] = 9566,  
2206 ["boxVr"] = 9567,  
2207 ["boxVR"] = 9568,  
2208 ["boxvL"] = 9569,  
2209 ["boxVl"] = 9570,  
2210 ["boxVL"] = 9571,

2211 ["boxHd"] = 9572,  
 2212 ["boxhD"] = 9573,  
 2213 ["boxHD"] = 9574,  
 2214 ["boxHu"] = 9575,  
 2215 ["boxhU"] = 9576,  
 2216 ["boxHU"] = 9577,  
 2217 ["boxvH"] = 9578,  
 2218 ["boxVh"] = 9579,  
 2219 ["boxVH"] = 9580,  
 2220 ["uhblk"] = 9600,  
 2221 ["lhblk"] = 9604,  
 2222 ["block"] = 9608,  
 2223 ["blk14"] = 9617,  
 2224 ["blk12"] = 9618,  
 2225 ["blk34"] = 9619,  
 2226 ["squ"] = 9633,  
 2227 ["square"] = 9633,  
 2228 ["Square"] = 9633,  
 2229 ["squf"] = 9642,  
 2230 ["squarf"] = 9642,  
 2231 ["blacksquare"] = 9642,  
 2232 ["FilledVerySmallSquare"] = 9642,  
 2233 ["EmptyVerySmallSquare"] = 9643,  
 2234 ["rect"] = 9645,  
 2235 ["marker"] = 9646,  
 2236 ["fltns"] = 9649,  
 2237 ["xutri"] = 9651,  
 2238 ["bigtriangleup"] = 9651,  
 2239 ["utrif"] = 9652,  
 2240 ["blacktriangle"] = 9652,  
 2241 ["utri"] = 9653,  
 2242 ["triangle"] = 9653,  
 2243 ["rtrif"] = 9656,  
 2244 ["blacktriangleright"] = 9656,  
 2245 ["rtri"] = 9657,  
 2246 ["triangleright"] = 9657,  
 2247 ["xdtri"] = 9661,  
 2248 ["bigtriangledown"] = 9661,  
 2249 ["dtrif"] = 9662,  
 2250 ["blacktriangledown"] = 9662,  
 2251 ["dtri"] = 9663,  
 2252 ["triangledown"] = 9663,  
 2253 ["ltrif"] = 9666,  
 2254 ["blacktriangleleft"] = 9666,  
 2255 ["ltri"] = 9667,  
 2256 ["triangleleft"] = 9667,  
 2257 ["loz"] = 9674,

2258 ["lozenge"] = 9674,  
2259 ["cir"] = 9675,  
2260 ["tridot"] = 9708,  
2261 ["xcirc"] = 9711,  
2262 ["bigcirc"] = 9711,  
2263 ["ultri"] = 9720,  
2264 ["urtri"] = 9721,  
2265 ["lltri"] = 9722,  
2266 ["EmptySmallSquare"] = 9723,  
2267 ["FilledSmallSquare"] = 9724,  
2268 ["starf"] = 9733,  
2269 ["bigstar"] = 9733,  
2270 ["star"] = 9734,  
2271 ["phone"] = 9742,  
2272 ["female"] = 9792,  
2273 ["male"] = 9794,  
2274 ["spades"] = 9824,  
2275 ["spadesuit"] = 9824,  
2276 ["clubs"] = 9827,  
2277 ["clubsuit"] = 9827,  
2278 ["hearts"] = 9829,  
2279 ["heartsuit"] = 9829,  
2280 ["diams"] = 9830,  
2281 ["diamondsuit"] = 9830,  
2282 ["sung"] = 9834,  
2283 ["flat"] = 9837,  
2284 ["natur"] = 9838,  
2285 ["natural"] = 9838,  
2286 ["sharp"] = 9839,  
2287 ["check"] = 10003,  
2288 ["checkmark"] = 10003,  
2289 ["cross"] = 10007,  
2290 ["malt"] = 10016,  
2291 ["maltese"] = 10016,  
2292 ["sext"] = 10038,  
2293 ["VerticalSeparator"] = 10072,  
2294 ["lbrk"] = 10098,  
2295 ["rbrk"] = 10099,  
2296 ["lobrk"] = 10214,  
2297 ["LeftDoubleBracket"] = 10214,  
2298 ["robrk"] = 10215,  
2299 ["RightDoubleBracket"] = 10215,  
2300 ["lang"] = 10216,  
2301 ["LeftAngleBracket"] = 10216,  
2302 ["langle"] = 10216,  
2303 ["rang"] = 10217,  
2304 ["RightAngleBracket"] = 10217,



2305 ["rangle"] = 10217,  
2306 ["Lang"] = 10218,  
2307 ["Rang"] = 10219,  
2308 ["loang"] = 10220,  
2309 ["roang"] = 10221,  
2310 ["xlarr"] = 10229,  
2311 ["longleftarrow"] = 10229,  
2312 ["LongLeftArrow"] = 10229,  
2313 ["xrarr"] = 10230,  
2314 ["longrightarrow"] = 10230,  
2315 ["LongRightArrow"] = 10230,  
2316 ["xharr"] = 10231,  
2317 ["longlefttrightarrow"] = 10231,  
2318 ["LongLeftRightArrow"] = 10231,  
2319 ["xlArr"] = 10232,  
2320 ["Longleftarrow"] = 10232,  
2321 ["DoubleLongLeftArrow"] = 10232,  
2322 ["xrArr"] = 10233,  
2323 ["Longrightarrow"] = 10233,  
2324 ["DoubleLongRightArrow"] = 10233,  
2325 ["xhArr"] = 10234,  
2326 ["Longlefttrightarrow"] = 10234,  
2327 ["DoubleLongLeftRightArrow"] = 10234,  
2328 ["xmap"] = 10236,  
2329 ["longmapsto"] = 10236,  
2330 ["dzigrarr"] = 10239,  
2331 ["nvlArr"] = 10498,  
2332 ["nvrArr"] = 10499,  
2333 ["nvHarr"] = 10500,  
2334 ["Map"] = 10501,  
2335 ["lbarr"] = 10508,  
2336 ["rbarr"] = 10509,  
2337 ["bkarow"] = 10509,  
2338 ["lBarr"] = 10510,  
2339 ["rBarr"] = 10511,  
2340 ["dbkarow"] = 10511,  
2341 ["RBarr"] = 10512,  
2342 ["drbkarow"] = 10512,  
2343 ["DDotrahd"] = 10513,  
2344 ["UpArrowBar"] = 10514,  
2345 ["DownArrowBar"] = 10515,  
2346 ["Rarrtl"] = 10518,  
2347 ["latail"] = 10521,  
2348 ["ratail"] = 10522,  
2349 ["lAtail"] = 10523,  
2350 ["rAtail"] = 10524,  
2351 ["larrfs"] = 10525,

2352 ["rarrfs"] = 10526,  
2353 ["larrbfs"] = 10527,  
2354 ["rarrbfs"] = 10528,  
2355 ["nwarhk"] = 10531,  
2356 ["nearhk"] = 10532,  
2357 ["searhk"] = 10533,  
2358 ["hksearow"] = 10533,  
2359 ["swarhk"] = 10534,  
2360 ["hkswarow"] = 10534,  
2361 ["nwnear"] = 10535,  
2362 ["nesear"] = 10536,  
2363 ["toea"] = 10536,  
2364 ["seswar"] = 10537,  
2365 ["tosa"] = 10537,  
2366 ["swnwar"] = 10538,  
2367 ["rarrc"] = 10547,  
2368 ["cudarr"] = 10549,  
2369 ["ldca"] = 10550,  
2370 ["rdca"] = 10551,  
2371 ["cudarrl"] = 10552,  
2372 ["larrpl"] = 10553,  
2373 ["curarrm"] = 10556,  
2374 ["cularrp"] = 10557,  
2375 ["rarrpl"] = 10565,  
2376 ["harrcir"] = 10568,  
2377 ["Uarrocir"] = 10569,  
2378 ["lurdshar"] = 10570,  
2379 ["ldrushar"] = 10571,  
2380 ["LeftRightVector"] = 10574,  
2381 ["RightUpDownVector"] = 10575,  
2382 ["DownLeftRightVector"] = 10576,  
2383 ["LeftUpDownVector"] = 10577,  
2384 ["LeftVectorBar"] = 10578,  
2385 ["RightVectorBar"] = 10579,  
2386 ["RightUpVectorBar"] = 10580,  
2387 ["RightDownVectorBar"] = 10581,  
2388 ["DownLeftVectorBar"] = 10582,  
2389 ["DownRightVectorBar"] = 10583,  
2390 ["LeftUpVectorBar"] = 10584,  
2391 ["LeftDownVectorBar"] = 10585,  
2392 ["LeftTeeVector"] = 10586,  
2393 ["RightTeeVector"] = 10587,  
2394 ["RightUpTeeVector"] = 10588,  
2395 ["RightDownTeeVector"] = 10589,  
2396 ["DownLeftTeeVector"] = 10590,  
2397 ["DownRightTeeVector"] = 10591,  
2398 ["LeftUpTeeVector"] = 10592,

2399 ["LeftDownTeeVector"] = 10593,  
 2400 ["lHar"] = 10594,  
 2401 ["uHar"] = 10595,  
 2402 ["rHar"] = 10596,  
 2403 ["dHar"] = 10597,  
 2404 ["luruhar"] = 10598,  
 2405 ["ldrdhar"] = 10599,  
 2406 ["ruluhar"] = 10600,  
 2407 ["rdldhar"] = 10601,  
 2408 ["lharul"] = 10602,  
 2409 ["llhard"] = 10603,  
 2410 ["rharul"] = 10604,  
 2411 ["lrhard"] = 10605,  
 2412 ["udhar"] = 10606,  
 2413 ["UpEquilibrium"] = 10606,  
 2414 ["duhar"] = 10607,  
 2415 ["ReverseUpEquilibrium"] = 10607,  
 2416 ["RoundImplies"] = 10608,  
 2417 ["erarr"] = 10609,  
 2418 ["simrarr"] = 10610,  
 2419 ["larrsim"] = 10611,  
 2420 ["rarrsim"] = 10612,  
 2421 ["rarrap"] = 10613,  
 2422 ["ltlarr"] = 10614,  
 2423 ["gtrarr"] = 10616,  
 2424 ["subrarr"] = 10617,  
 2425 ["suplarr"] = 10619,  
 2426 ["lfisht"] = 10620,  
 2427 ["rfisht"] = 10621,  
 2428 ["ufisht"] = 10622,  
 2429 ["dfisht"] = 10623,  
 2430 ["lopar"] = 10629,  
 2431 ["ropar"] = 10630,  
 2432 ["lbrke"] = 10635,  
 2433 ["rbrke"] = 10636,  
 2434 ["lbrkslu"] = 10637,  
 2435 ["rbrksld"] = 10638,  
 2436 ["lbrksld"] = 10639,  
 2437 ["rbrkslu"] = 10640,  
 2438 ["langd"] = 10641,  
 2439 ["rangd"] = 10642,  
 2440 ["lparlt"] = 10643,  
 2441 ["rpargt"] = 10644,  
 2442 ["gtlPar"] = 10645,  
 2443 ["ltrPar"] = 10646,  
 2444 ["vzigzag"] = 10650,  
 2445 ["vangrt"] = 10652,

2446 ["angrtvbd"] = 10653,  
 2447 ["ange"] = 10660,  
 2448 ["range"] = 10661,  
 2449 ["dwangle"] = 10662,  
 2450 ["uwangle"] = 10663,  
 2451 ["angmsdaa"] = 10664,  
 2452 ["angmsdab"] = 10665,  
 2453 ["angmsdac"] = 10666,  
 2454 ["angmsdad"] = 10667,  
 2455 ["angmsdae"] = 10668,  
 2456 ["angmsdaf"] = 10669,  
 2457 ["angmsdag"] = 10670,  
 2458 ["angmsdah"] = 10671,  
 2459 ["bemptyv"] = 10672,  
 2460 ["demptyv"] = 10673,  
 2461 ["cemptyv"] = 10674,  
 2462 ["raemptyv"] = 10675,  
 2463 ["laemptyv"] = 10676,  
 2464 ["ohbar"] = 10677,  
 2465 ["omid"] = 10678,  
 2466 ["opar"] = 10679,  
 2467 ["operp"] = 10681,  
 2468 ["olcross"] = 10683,  
 2469 ["odsold"] = 10684,  
 2470 ["olcir"] = 10686,  
 2471 ["ofcir"] = 10687,  
 2472 ["olt"] = 10688,  
 2473 ["ogt"] = 10689,  
 2474 ["cirscir"] = 10690,  
 2475 ["cirE"] = 10691,  
 2476 ["solb"] = 10692,  
 2477 ["bsolb"] = 10693,  
 2478 ["boxbox"] = 10697,  
 2479 ["trisb"] = 10701,  
 2480 ["rtriltri"] = 10702,  
 2481 ["LeftTriangleBar"] = 10703,  
 2482 ["RightTriangleBar"] = 10704,  
 2483 ["race"] = 10714,  
 2484 ["iinfin"] = 10716,  
 2485 ["infintie"] = 10717,  
 2486 ["nvinfin"] = 10718,  
 2487 ["eparsl"] = 10723,  
 2488 ["smeparsl"] = 10724,  
 2489 ["eqvparsl"] = 10725,  
 2490 ["lozf"] = 10731,  
 2491 ["blacklozenge"] = 10731,  
 2492 ["RuleDelayed"] = 10740,

2493 ["dsol"] = 10742,  
2494 ["xodot"] = 10752,  
2495 ["bigodot"] = 10752,  
2496 ["xoplus"] = 10753,  
2497 ["bigoplus"] = 10753,  
2498 ["xotime"] = 10754,  
2499 ["bigotimes"] = 10754,  
2500 ["xuplus"] = 10756,  
2501 ["biguplus"] = 10756,  
2502 ["xsqcup"] = 10758,  
2503 ["bigsqcup"] = 10758,  
2504 ["qint"] = 10764,  
2505 ["iiiint"] = 10764,  
2506 ["fpartint"] = 10765,  
2507 ["cirfnint"] = 10768,  
2508 ["awint"] = 10769,  
2509 ["rppolint"] = 10770,  
2510 ["scpolint"] = 10771,  
2511 ["npolint"] = 10772,  
2512 ["pointint"] = 10773,  
2513 ["quatint"] = 10774,  
2514 ["intlarhk"] = 10775,  
2515 ["pluscir"] = 10786,  
2516 ["plusacir"] = 10787,  
2517 ["simplus"] = 10788,  
2518 ["plusdu"] = 10789,  
2519 ["plussim"] = 10790,  
2520 ["plustwo"] = 10791,  
2521 ["mcomma"] = 10793,  
2522 ["minusdu"] = 10794,  
2523 ["loplus"] = 10797,  
2524 ["roplus"] = 10798,  
2525 ["Cross"] = 10799,  
2526 ["timesd"] = 10800,  
2527 ["timesbar"] = 10801,  
2528 ["smashp"] = 10803,  
2529 ["lotimes"] = 10804,  
2530 ["rotimes"] = 10805,  
2531 ["otimesas"] = 10806,  
2532 ["Otimes"] = 10807,  
2533 ["odiv"] = 10808,  
2534 ["triplus"] = 10809,  
2535 ["triminus"] = 10810,  
2536 ["tritime"] = 10811,  
2537 ["iproduct"] = 10812,  
2538 ["intprod"] = 10812,  
2539 ["amalg"] = 10815,

2540 ["capdot"] = 10816,  
 2541 ["ncup"] = 10818,  
 2542 ["ncap"] = 10819,  
 2543 ["capand"] = 10820,  
 2544 ["cupor"] = 10821,  
 2545 ["cupcap"] = 10822,  
 2546 ["capcup"] = 10823,  
 2547 ["cupbrcap"] = 10824,  
 2548 ["capbrcup"] = 10825,  
 2549 ["cupcup"] = 10826,  
 2550 ["capcap"] = 10827,  
 2551 ["ccups"] = 10828,  
 2552 ["ccaps"] = 10829,  
 2553 ["ccupssm"] = 10832,  
 2554 ["And"] = 10835,  
 2555 ["Or"] = 10836,  
 2556 ["andand"] = 10837,  
 2557 ["oror"] = 10838,  
 2558 ["orslope"] = 10839,  
 2559 ["andslope"] = 10840,  
 2560 ["andv"] = 10842,  
 2561 ["orv"] = 10843,  
 2562 ["andd"] = 10844,  
 2563 ["ord"] = 10845,  
 2564 ["wedbar"] = 10847,  
 2565 ["sdote"] = 10854,  
 2566 ["simdot"] = 10858,  
 2567 ["congdot"] = 10861,  
 2568 ["easter"] = 10862,  
 2569 ["apacir"] = 10863,  
 2570 ["apE"] = 10864,  
 2571 ["eplus"] = 10865,  
 2572 ["pluse"] = 10866,  
 2573 ["Esim"] = 10867,  
 2574 ["Colone"] = 10868,  
 2575 ["Equal"] = 10869,  
 2576 ["eDDot"] = 10871,  
 2577 ["ddotseq"] = 10871,  
 2578 ["equivDD"] = 10872,  
 2579 ["ltcir"] = 10873,  
 2580 ["gtcir"] = 10874,  
 2581 ["ltquest"] = 10875,  
 2582 ["gtquest"] = 10876,  
 2583 ["les"] = 10877,  
 2584 ["LessSlantEqual"] = 10877,  
 2585 ["leqslant"] = 10877,  
 2586 ["ges"] = 10878,

2587 ["GreaterSlantEqual"] = 10878,  
2588 ["geqslant"] = 10878,  
2589 ["lesdot"] = 10879,  
2590 ["gesdot"] = 10880,  
2591 ["lesdoto"] = 10881,  
2592 ["gesdoto"] = 10882,  
2593 ["lesdotor"] = 10883,  
2594 ["gesdotol"] = 10884,  
2595 ["lap"] = 10885,  
2596 ["lessapprox"] = 10885,  
2597 ["gap"] = 10886,  
2598 ["gtrapprox"] = 10886,  
2599 ["lne"] = 10887,  
2600 ["lneq"] = 10887,  
2601 ["gne"] = 10888,  
2602 ["gneq"] = 10888,  
2603 ["lnap"] = 10889,  
2604 ["lnapprox"] = 10889,  
2605 ["gnap"] = 10890,  
2606 ["gnapprox"] = 10890,  
2607 ["lEg"] = 10891,  
2608 ["lesseqqgtr"] = 10891,  
2609 ["gEl"] = 10892,  
2610 ["gtreqqless"] = 10892,  
2611 ["lsime"] = 10893,  
2612 ["gsime"] = 10894,  
2613 ["lsimg"] = 10895,  
2614 ["gsiml"] = 10896,  
2615 ["lgE"] = 10897,  
2616 ["glE"] = 10898,  
2617 ["lesges"] = 10899,  
2618 ["gesles"] = 10900,  
2619 ["els"] = 10901,  
2620 ["eqslantless"] = 10901,  
2621 ["egs"] = 10902,  
2622 ["eqslantgtr"] = 10902,  
2623 ["elsdot"] = 10903,  
2624 ["egsdot"] = 10904,  
2625 ["el"] = 10905,  
2626 ["eg"] = 10906,  
2627 ["siml"] = 10909,  
2628 ["sing"] = 10910,  
2629 ["simlE"] = 10911,  
2630 ["singE"] = 10912,  
2631 ["LessLess"] = 10913,  
2632 ["GreaterGreater"] = 10914,  
2633 ["glj"] = 10916,

2634 ["gla"] = 10917,  
 2635 ["ltcc"] = 10918,  
 2636 ["gtcc"] = 10919,  
 2637 ["lescc"] = 10920,  
 2638 ["gescc"] = 10921,  
 2639 ["smt"] = 10922,  
 2640 ["lat"] = 10923,  
 2641 ["smte"] = 10924,  
 2642 ["late"] = 10925,  
 2643 ["bumpE"] = 10926,  
 2644 ["pre"] = 10927,  
 2645 ["preceq"] = 10927,  
 2646 ["PrecedesEqual"] = 10927,  
 2647 ["sce"] = 10928,  
 2648 ["succeq"] = 10928,  
 2649 ["SucceedsEqual"] = 10928,  
 2650 ["prE"] = 10931,  
 2651 ["scE"] = 10932,  
 2652 ["prnE"] = 10933,  
 2653 ["precneqq"] = 10933,  
 2654 ["scnE"] = 10934,  
 2655 ["succneqq"] = 10934,  
 2656 ["prap"] = 10935,  
 2657 ["precapprox"] = 10935,  
 2658 ["scap"] = 10936,  
 2659 ["succapprox"] = 10936,  
 2660 ["prnap"] = 10937,  
 2661 ["precnapprox"] = 10937,  
 2662 ["scnap"] = 10938,  
 2663 ["succnapprox"] = 10938,  
 2664 ["Pr"] = 10939,  
 2665 ["Sc"] = 10940,  
 2666 ["subdot"] = 10941,  
 2667 ["supdot"] = 10942,  
 2668 ["subplus"] = 10943,  
 2669 ["supplus"] = 10944,  
 2670 ["submult"] = 10945,  
 2671 ["supmult"] = 10946,  
 2672 ["subedot"] = 10947,  
 2673 ["supedot"] = 10948,  
 2674 ["subE"] = 10949,  
 2675 ["subseteqq"] = 10949,  
 2676 ["supE"] = 10950,  
 2677 ["supseteqq"] = 10950,  
 2678 ["subsim"] = 10951,  
 2679 ["supsim"] = 10952,  
 2680 ["subnE"] = 10955,



2681 ["subsetneqq"] = 10955,  
 2682 ["supnE"] = 10956,  
 2683 ["supsetneqq"] = 10956,  
 2684 ["csub"] = 10959,  
 2685 ["csup"] = 10960,  
 2686 ["csube"] = 10961,  
 2687 ["csupe"] = 10962,  
 2688 ["subsup"] = 10963,  
 2689 ["supsub"] = 10964,  
 2690 ["subsub"] = 10965,  
 2691 ["supsup"] = 10966,  
 2692 ["suphsub"] = 10967,  
 2693 ["supdsub"] = 10968,  
 2694 ["forkv"] = 10969,  
 2695 ["topfork"] = 10970,  
 2696 ["mlcp"] = 10971,  
 2697 ["Dashv"] = 10980,  
 2698 ["DoubleLeftTee"] = 10980,  
 2699 ["Vdashl"] = 10982,  
 2700 ["Barv"] = 10983,  
 2701 ["vBar"] = 10984,  
 2702 ["vBarv"] = 10985,  
 2703 ["Vbar"] = 10987,  
 2704 ["Not"] = 10988,  
 2705 ["bNot"] = 10989,  
 2706 ["rnmid"] = 10990,  
 2707 ["cirmid"] = 10991,  
 2708 ["midcir"] = 10992,  
 2709 ["topcir"] = 10993,  
 2710 ["nhpar"] = 10994,  
 2711 ["parsim"] = 10995,  
 2712 ["parsl"] = 11005,  
 2713 ["fflig"] = 64256,  
 2714 ["filig"] = 64257,  
 2715 ["fllig"] = 64258,  
 2716 ["ffilig"] = 64259,  
 2717 ["ffllig"] = 64260,  
 2718 ["Ascr"] = 119964,  
 2719 ["Cscr"] = 119966,  
 2720 ["Dscr"] = 119967,  
 2721 ["Gscr"] = 119970,  
 2722 ["Jscr"] = 119973,  
 2723 ["Kscr"] = 119974,  
 2724 ["Nscr"] = 119977,  
 2725 ["Oscr"] = 119978,  
 2726 ["Pscr"] = 119979,  
 2727 ["Qscr"] = 119980,

2728 ["Sscr"] = 119982,  
2729 ["Tscr"] = 119983,  
2730 ["Uscr"] = 119984,  
2731 ["Vscr"] = 119985,  
2732 ["Wscr"] = 119986,  
2733 ["Xscr"] = 119987,  
2734 ["Yscr"] = 119988,  
2735 ["Zscr"] = 119989,  
2736 ["ascr"] = 119990,  
2737 ["bscr"] = 119991,  
2738 ["cscr"] = 119992,  
2739 ["dscr"] = 119993,  
2740 ["fscr"] = 119995,  
2741 ["hscr"] = 119997,  
2742 ["iscr"] = 119998,  
2743 ["jscr"] = 119999,  
2744 ["kscr"] = 120000,  
2745 ["lscr"] = 120001,  
2746 ["mscr"] = 120002,  
2747 ["nscr"] = 120003,  
2748 ["pscr"] = 120005,  
2749 ["qscr"] = 120006,  
2750 ["rscr"] = 120007,  
2751 ["sscr"] = 120008,  
2752 ["tscr"] = 120009,  
2753 ["uscr"] = 120010,  
2754 ["vscr"] = 120011,  
2755 ["wscr"] = 120012,  
2756 ["xscr"] = 120013,  
2757 ["yscr"] = 120014,  
2758 ["zscr"] = 120015,  
2759 ["Afr"] = 120068,  
2760 ["Bfr"] = 120069,  
2761 ["Dfr"] = 120071,  
2762 ["Efr"] = 120072,  
2763 ["Ffr"] = 120073,  
2764 ["Gfr"] = 120074,  
2765 ["Jfr"] = 120077,  
2766 ["Kfr"] = 120078,  
2767 ["Lfr"] = 120079,  
2768 ["Mfr"] = 120080,  
2769 ["Nfr"] = 120081,  
2770 ["Ofr"] = 120082,  
2771 ["Pfr"] = 120083,  
2772 ["Qfr"] = 120084,  
2773 ["Sfr"] = 120086,  
2774 ["Tfr"] = 120087,

2775 ["Ufr"] = 120088,  
2776 ["Vfr"] = 120089,  
2777 ["Wfr"] = 120090,  
2778 ["Xfr"] = 120091,  
2779 ["Yfr"] = 120092,  
2780 ["afr"] = 120094,  
2781 ["bfr"] = 120095,  
2782 ["cfr"] = 120096,  
2783 ["dfr"] = 120097,  
2784 ["efr"] = 120098,  
2785 ["ffr"] = 120099,  
2786 ["gfr"] = 120100,  
2787 ["hfr"] = 120101,  
2788 ["ifr"] = 120102,  
2789 ["jfr"] = 120103,  
2790 ["kfr"] = 120104,  
2791 ["lfr"] = 120105,  
2792 ["mfr"] = 120106,  
2793 ["nfr"] = 120107,  
2794 ["ofr"] = 120108,  
2795 ["pfr"] = 120109,  
2796 ["qfr"] = 120110,  
2797 ["rfr"] = 120111,  
2798 ["sfr"] = 120112,  
2799 ["tfr"] = 120113,  
2800 ["ufr"] = 120114,  
2801 ["vfr"] = 120115,  
2802 ["wfr"] = 120116,  
2803 ["xfr"] = 120117,  
2804 ["yfr"] = 120118,  
2805 ["zfr"] = 120119,  
2806 ["Aopf"] = 120120,  
2807 ["Bopf"] = 120121,  
2808 ["Dopf"] = 120123,  
2809 ["Eopf"] = 120124,  
2810 ["Fopf"] = 120125,  
2811 ["Gopf"] = 120126,  
2812 ["Iopf"] = 120128,  
2813 ["Jopf"] = 120129,  
2814 ["Kopf"] = 120130,  
2815 ["Lopf"] = 120131,  
2816 ["Mopf"] = 120132,  
2817 ["Oopf"] = 120134,  
2818 ["Sopf"] = 120138,  
2819 ["Topf"] = 120139,  
2820 ["Uopf"] = 120140,  
2821 ["Vopf"] = 120141,

```

2822 ["Wopf"] = 120142,
2823 ["Xopf"] = 120143,
2824 ["Yopf"] = 120144,
2825 ["aopf"] = 120146,
2826 ["bopf"] = 120147,
2827 ["copf"] = 120148,
2828 ["dopf"] = 120149,
2829 ["eopf"] = 120150,
2830 ["fopf"] = 120151,
2831 ["gopf"] = 120152,
2832 ["hopf"] = 120153,
2833 ["iopf"] = 120154,
2834 ["jopf"] = 120155,
2835 ["kopf"] = 120156,
2836 ["lopf"] = 120157,
2837 ["mopf"] = 120158,
2838 ["nopf"] = 120159,
2839 ["oopf"] = 120160,
2840 ["popf"] = 120161,
2841 ["qopf"] = 120162,
2842 ["ropf"] = 120163,
2843 ["sopf"] = 120164,
2844 ["topf"] = 120165,
2845 ["uopf"] = 120166,
2846 ["vopf"] = 120167,
2847 ["wopf"] = 120168,
2848 ["xopf"] = 120169,
2849 ["yopf"] = 120170,
2850 ["zopf"] = 120171,
2851 }

```

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

```

2852 function entities.dec_entity(s)
2853   return unicode.utf8.char(tonumber(s))
2854 end

```

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

```

2855 function entities.hex_entity(s)
2856   return unicode.utf8.char(tonumber("0x"..s))
2857 end

```

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

```

2858 function entities.char_entity(s)
2859   local n = character_entities[s]
2860   if n == nil then

```

```

2861     return "&" .. s .. ";"
2862 end
2863 return unicode.utf8.char(n)
2864 end

```

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

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

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

```

2865 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`.

```

2866 function M.writer.new(options)
2867     local self = {}
2868     options = options or {}

```

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

```

2869     setmetatable(options, { __index = function (_, key)
2870         return defaultOptions[key] end })

```

Parse the `slice` option and define `writer->slice_begin` `writer->slice_end`, and `writer->is_writing`.

```

2871     local slice_specifiers = {}
2872     for specifier in options.slice:gmatch("[^%s]+") do
2873         table.insert(slice_specifiers, specifier)
2874     end
2875
2876     if #slice_specifiers == 2 then
2877         self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
2878         local slice_begin_type = self.slice_begin:sub(1, 1)
2879         if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
2880             self.slice_begin = "^" .. self.slice_begin
2881         end
2882         local slice_end_type = self.slice_end:sub(1, 1)

```

```

2883     if slice_end_type =~ "^" and slice_end_type =~ "$" then
2884         self.slice_end = "$" .. self.slice_end
2885     end
2886     elsif #slice_specifiers == 1 then
2887         self.slice_begin = "^" .. slice_specifiers[1]
2888         self.slice_end = "$" .. slice_specifiers[1]
2889     end
2890
2891     if self.slice_begin == "^" and self.slice_end =~ "^" then
2892         self.is_writing = true
2893     else
2894         self.is_writing = false
2895     end

```

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

```

2896     self.suffix = ".tex"

```

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

```

2897     self.space = " "

```

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

```

2898     self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

2899     function self.plain(s)
2900         return s
2901     end

```

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

```

2902     function self.paragraph(s)
2903         if not self.is_writing then return "" end
2904         return s
2905     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.

```

2906     function self.pack(name)
2907         return [[\input ]] .. name .. [[\relax{}]]
2908     end

```

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

```

2909     function self.interblocksep()
2910         if not self.is_writing then return "" end
2911         return "\\markdownRendererInterblockSeparator\n{}"
2912     end

```

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

```

2913     self.eof = [[\relax]]

```

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

```
2914 self.linebreak = "\\markdownRendererLineBreak\n{"
```

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

```
2915 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
2916 function self.hrule()
2917   if not self.is_writing then return "" end
2918   return "\\markdownRendererHorizontalRule{"
2919 end
```

Define a table `escaped_chars` containing the mapping from special plain T<sub>E</sub>X characters (including the active pipe character (|) of ConT<sub>E</sub>Xt) to their escaped variants. Define tables `escaped_minimal_chars` and `escaped_minimal_strings` containing the mapping from special plain characters and character strings that need to be escaped even in content that will not be typeset.

```
2920 local escaped_chars = {
2921   [{""] = "\\markdownRendererLeftBrace{"},
2922   ["}"] = "\\markdownRendererRightBrace{"},
2923   [{"$"}] = "\\markdownRendererDollarSign{"},
2924   [{"%"}] = "\\markdownRendererPercentSign{"},
2925   [{"&"}] = "\\markdownRendererAmpersand{"},
2926   [{"_"}] = "\\markdownRendererUnderscore{"},
2927   [{"#"}] = "\\markdownRendererHash{"},
2928   [{"^"}] = "\\markdownRendererCircumflex{"},
2929   [{"\\"}] = "\\markdownRendererBackslash{"},
2930   [{"~"}] = "\\markdownRendererTilde{"},
2931   [{"|"}] = "\\markdownRendererPipe{"},
2932 }
2933 local escaped_uri_chars = {
2934   [{""] = "\\markdownRendererLeftBrace{"},
2935   ["}"] = "\\markdownRendererRightBrace{"},
2936   [{"%"}] = "\\markdownRendererPercentSign{"},
2937   [{"\\"}] = "\\markdownRendererBackslash{"},
2938 }
2939 local escaped_citation_chars = {
2940   [{""] = "\\markdownRendererLeftBrace{"},
2941   ["}"] = "\\markdownRendererRightBrace{"},
2942   [{"%"}] = "\\markdownRendererPercentSign{"},
2943   [{"#"}] = "\\markdownRendererHash{"},
2944   [{"\\"}] = "\\markdownRendererBackslash{"},
2945 }
2946 local escaped_minimal_strings = {
2947   [{"^^"}] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
2948 }
```

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

```
2949 local escape = util.escaper(escaped_chars)
2950 local escape_citation = util.escaper(escaped_citation_chars,
2951   escaped_minimal_strings)
2952 local escape_uri = util.escaper(escaped_uri_chars, escaped_minimal_strings)
```

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

```
2953 if options.hybrid then
2954   self.string = function(s) return s end
2955   self.citation = function(c) return c end
2956   self.uri = function(u) return u end
2957 else
2958   self.string = escape
2959   self.citation = escape_citation
2960   self.uri = escape_uri
2961 end
```

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

```
2962 function self.code(s)
2963   return {"\\markdownRendererCodeSpan{", escape(s), "}"}
2964 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.

```
2965 function self.link(lab,src,tit)
2966   return {"\\markdownRendererLink{",lab,"}",
2967     "{",self.string(src),"",
2968     "{",self.uri(src),"",
2969     "{",self.string(tit or ""),"}"}
2970 end
```

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.

```
2971 function self.table(rows, caption)
2972   local buffer = {"\\markdownRendererTable{",
2973     caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
2974   local temp = rows[2] -- put alignments on the first row
2975   rows[2] = rows[1]
2976   rows[1] = temp
```



```

2977     for i, row in ipairs(rows) do
2978         table.insert(buffer, "{")
2979         for _, column in ipairs(row) do
2980             if i > 1 then -- do not use braces for alignments
2981                 table.insert(buffer, "{")
2982             end
2983             table.insert(buffer, column)
2984             if i > 1 then
2985                 table.insert(buffer, "%\n")
2986             end
2987         end
2988         table.insert(buffer, "%\n")
2989     end
2990     return buffer
2991 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.

```

2992     function self.image(lab,src,tit)
2993         return {"\markdownRendererImage{" ,lab,"}",
2994             {" ,self.string(src),"},
2995             {" ,self.uri(src),"},
2996             {" ,self.string(tit or ""),"},"}
2997     end

```

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

```

2998     local languages_json = (function()
2999         local kpse = require("kpse")
3000         kpse.set_program_name("luatex")
3001         local base, prev, curr
3002         for _, file in ipairs{kpse.lookup(options.contentBlocksLanguageMap,
3003             { all=true })} do
3004             json = io.open(file, "r"):read("*all")
3005                 :gsub('("[^\n]-"):','[%1]=')
3006             curr = (function()
3007                 local _ENV={ json=json, load=load } -- run in sandbox
3008                 return load("return "..json)()
3009             end)()
3010             if type(curr) == "table" then
3011                 if base == nil then
3012                     base = curr
3013                 else
3014                     setmetatable(prev, { __index = curr })

```

```

3015         end
3016         prev = curr
3017     end
3018 end
3019 return base or {}
3020 end()

```

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

```

3021 function self.contentblock(src,suf,type,tit)
3022     if not self.is_writing then return "" end
3023     src = src..".."..suf
3024     suf = suf:lower()
3025     if type == "onlineimage" then
3026         return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"}"} ,
3027             {" ,self.string(src),"}"} ,
3028             {" ,self.uri(src),"}"} ,
3029             {" ,self.string(tit or ""),"}"}
3030     elseif languages_json[suf] then
3031         return {"\\markdownRendererContentBlockCode{" ,suf,"}"} ,
3032             {" ,self.string(languages_json[suf]),"}"} ,
3033             {" ,self.string(src),"}"} ,
3034             {" ,self.uri(src),"}"} ,
3035             {" ,self.string(tit or ""),"}"}
3036     else
3037         return {"\\markdownRendererContentBlock{" ,suf,"}"} ,
3038             {" ,self.string(src),"}"} ,
3039             {" ,self.uri(src),"}"} ,
3040             {" ,self.string(tit or ""),"}"}
3041     end
3042 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.

```

3043 local function ulitem(s)
3044     return {"\\markdownRendererULItem " ,s,
3045         "\\markdownRendererULItemEnd "}
3046 end
3047
3048 function self.bulletlist(items,tight)
3049     if not self.is_writing then return "" end
3050     local buffer = {}
3051     for _,item in ipairs(items) do
3052         buffer[#buffer + 1] = ulitem(item)

```

```

3053     end
3054     local contents = util.intersperse(buffer,"\n")
3055     if tight and options.tightLists then
3056         return {"\\markdownRendererUlBeginTight\n",contents,
3057             "\n\\markdownRendererUlEndTight "}
3058     else
3059         return {"\\markdownRendererUlBegin\n",contents,
3060             "\n\\markdownRendererUlEnd "}
3061     end
3062 end

```

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

```

3063 local function olitem(s,num)
3064     if num ~= nil then
3065         return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
3066             "\\markdownRendererOlItemEnd "}
3067     else
3068         return {"\\markdownRendererOlItem ",s,
3069             "\\markdownRendererOlItemEnd "}
3070     end
3071 end
3072
3073 function self.orderedlist(items,tight,startnum)
3074     if not self.is_writing then return "" end
3075     local buffer = {}
3076     local num = startnum
3077     for _,item in ipairs(items) do
3078         buffer[#buffer + 1] = olitem(item,num)
3079         if num ~= nil then
3080             num = num + 1
3081         end
3082     end
3083     local contents = util.intersperse(buffer,"\n")
3084     if tight and options.tightLists then
3085         return {"\\markdownRendererOlBeginTight\n",contents,
3086             "\n\\markdownRendererOlEndTight "}
3087     else
3088         return {"\\markdownRendererOlBegin\n",contents,
3089             "\n\\markdownRendererOlEnd "}
3090     end
3091 end

```

Define `writer->inline_html` and `writer->display_html` as functions that will

transform an inline or block HTML element respectively to the output format, where `html` is the HTML input.

```
3092 function self.inline_html(html) return "" end
3093 function self.display_html(html) return "" end
```

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

```
3094 local function dlitem(term, defs)
3095     local retVal = {"\\markdownRendererDlItem{" ,term,"}"}
3096     for _, def in ipairs(defs) do
3097         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin " ,def,
3098                             "\\markdownRendererDlDefinitionEnd "}
3099     end
3100     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
3101     return retVal
3102 end
3103
3104 function self.definitionlist(items,tight)
3105     if not self.is_writing then return "" end
3106     local buffer = {}
3107     for _,item in ipairs(items) do
3108         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
3109     end
3110     if tight and options.tightLists then
3111         return {"\\markdownRendererDlBeginTight\\n", buffer,
3112               "\\n\\markdownRendererDlEndTight"}
3113     else
3114         return {"\\markdownRendererDlBegin\\n", buffer,
3115               "\\n\\markdownRendererDlEnd"}
3116     end
3117 end
```

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

```
3118 function self.emphasis(s)
3119     return {"\\markdownRendererEmphasis{" ,s,"}"}
3120 end
```

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

```
3121 function self.strong(s)
3122     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
3123 end
```

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

```

3124 function self.blockquote(s)
3125   if #util.ropetostring(s) == 0 then return "" end
3126   return {"\markdownRendererBlockQuoteBegin\n",s,
3127     "\n\markdownRendererBlockQuoteEnd "}
3128 end

```

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

```

3129 function self.verbatim(s)
3130   if not self.is_writing then return "" end
3131   s = string.gsub(s, '[\r\n%$]*$', '')
3132   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
3133   return {"\markdownRendererInputVerbatim{" ,name,"}"}
3134 end

```

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

```

3135 function self.fencedCode(i, s)
3136   if not self.is_writing then return "" end
3137   s = string.gsub(s, '[\r\n%$]*$', '')
3138   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
3139   return {"\markdownRendererInputFencedCode{" ,name,"}{" ,i,"}"}
3140 end

```

Define `writer->active_headings` as a stack of identifiers of the headings that are currently active.

```

3141 self.active_headings = {}

```

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

```

3142 function self.heading(s,level,attributes)
3143   local active_headings = self.active_headings
3144   local slice_begin_type = self.slice_begin:sub(1, 1)
3145   local slice_begin_identifier = self.slice_begin:sub(2) or ""
3146   local slice_end_type = self.slice_end:sub(1, 1)
3147   local slice_end_identifier = self.slice_end:sub(2) or ""
3148
3149   while #active_headings < level do
3150     -- push empty identifiers for implied sections
3151     table.insert(active_headings, {})
3152   end
3153
3154   while #active_headings >= level do
3155     -- pop identifiers for sections that have ended
3156     local active_identifiers = active_headings[#active_headings]
3157     if active_identifiers[slice_begin_identifier] ~= nil
3158       and slice_begin_type == "$" then
3159       self.is_writing = true

```

```

3160     end
3161     if active_identifiers[slice_end_identifier] ~= nil
3162         and slice_end_type == "$" then
3163         self.is_writing = false
3164     end
3165     table.remove(active_headings, #active_headings)
3166 end
3167
3168 -- push identifiers for the new section
3169 attributes = attributes or {}
3170 local identifiers = {}
3171 for index = 1, #attributes do
3172     attribute = attributes[index]
3173     identifiers[attribute:sub(2)] = true
3174 end
3175 if identifiers[slice_begin_identifier] ~= nil
3176     and slice_begin_type == "^" then
3177     self.is_writing = true
3178 end
3179 if identifiers[slice_end_identifier] ~= nil
3180     and slice_end_type == "^" then
3181     self.is_writing = false
3182 end
3183 table.insert(active_headings, identifiers)
3184
3185 if not self.is_writing then return "" end
3186
3187 local cmd
3188 level = level + options.shiftHeadings
3189 if level <= 1 then
3190     cmd = "\\markdownRendererHeadingOne"
3191 elseif level == 2 then
3192     cmd = "\\markdownRendererHeadingTwo"
3193 elseif level == 3 then
3194     cmd = "\\markdownRendererHeadingThree"
3195 elseif level == 4 then
3196     cmd = "\\markdownRendererHeadingFour"
3197 elseif level == 5 then
3198     cmd = "\\markdownRendererHeadingFive"
3199 elseif level >= 6 then
3200     cmd = "\\markdownRendererHeadingSix"
3201 else
3202     cmd = ""
3203 end
3204 return {cmd, "{" , s, " "}
3205 end

```

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

```
3206 function self.note(s)
3207   return {"\markdownRendererFootnote{",s,""}
3208 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.

```
3209 function self.citations(text_cites, cites)
3210   local buffer = {"\markdownRenderer", text_cites and "TextCite" or "Cite",
3211     "{", #cites, "}"}
3212   for _,cite in ipairs(cites) do
3213     buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
3214       cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
3215   end
3216   return buffer
3217 end
3218
3219 return self
3220 end
```

### 3.1.4 Parsers

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

```
3221 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
3222 parsers.percent = P("%")
3223 parsers.at      = P("@")
3224 parsers.comma  = P(",")
3225 parsers.asterisk = P("*")
```

```

3226 parsers.dash           = P("-")
3227 parsers.plus           = P("+")
3228 parsers.underscore     = P("_")
3229 parsers.period         = P(".")
3230 parsers.hash           = P("#")
3231 parsers.ampersand      = P("&")
3232 parsers.backtick       = P("`")
3233 parsers.less           = P("<")
3234 parsers.more           = P(">")
3235 parsers.space          = P(" ")
3236 parsers.squote         = P("'")
3237 parsers.dquote         = P('"')
3238 parsers.lparent        = P("(")
3239 parsers.rparent        = P(")")
3240 parsers.lbracket       = P("[")
3241 parsers.rbracket       = P("]")
3242 parsers.lbrace         = P("{")
3243 parsers.rbrace         = P("}")
3244 parsers.circumflex     = P("^")
3245 parsers.slash          = P("/")
3246 parsers.equal          = P("=")
3247 parsers.colon          = P(":")
3248 parsers.semicolon     = P(";")
3249 parsers.exclamation   = P("!")
3250 parsers.pipe           = P("|")
3251 parsers.tilde          = P("~")
3252 parsers.tab            = P("\t")
3253 parsers.newline        = P("\n")
3254 parsers.tightblocksep  = P("\001")
3255
3256 parsers.digit          = R("09")
3257 parsers.hexdigit       = R("09", "af", "AF")
3258 parsers.letter         = R("AZ", "az")
3259 parsers.alphanumeric   = R("AZ", "az", "09")
3260 parsers.keyword        = parsers.letter
3261                       * parsers.alphanumeric^0
3262 parsers.citation_chars = parsers.alphanumeric
3263                       + S("#$%&-+<>~/_")
3264 parsers.internal_punctuation = S(":,.;.?")
3265
3266 parsers.doubleasterisks = P("**")
3267 parsers.doubleunderscores = P("__")
3268 parsers.fourspace      = P("    ")
3269
3270 parsers.any            = P(1)
3271 parsers.fail           = parsers.any - 1
3272

```



```

3273 parsers.escapable      = S("\\`*_{}[]()+_!.<>#-~:~@;")
3274 parsers.anyescaped    = P("\\") / "" * parsers.escapable
3275                        + parsers.any
3276
3277 parsers.spacechar      = S("\t ")
3278 parsers.spacing        = S(" \n\r\t")
3279 parsers.nonpacechar    = parsers.any - parsers.spacing
3280 parsers.optionalspace  = parsers.spacechar^0
3281
3282 parsers.specialchar    = S("*_`&[]<!\.~")
3283
3284 parsers.normalchar     = parsers.any - (parsers.specialchar
3285                                       + parsers.spacing
3286                                       + parsers.tightblocksep)
3287 parsers.eof            = -parsers.any
3288 parsers.nonindentpace  = parsers.space^-3 * - parsers.spacechar
3289 parsers.indent         = parsers.space^-3 * parsers.tab
3290                        + parsers.fourspace / ""
3291 parsers.linechar       = P(1 - parsers.newline)
3292
3293 parsers.blankline      = parsers.optionalspace
3294                        * parsers.newline / "\n"
3295 parsers.blanklines     = parsers.blankline^0
3296 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0
3297 parsers.indentedline   = parsers.indent / ""
3298                        * C(parsers.linechar^1 * parsers.newline^-
3299                          1)
3300 parsers.optionallyindentedline = parsers.indent^-1 / ""
3301                        * C(parsers.linechar^1 * parsers.newline^-
3302                          1)
3303 parsers.sp              = parsers.spacing^0
3304 parsers.spnl           = parsers.optionalspace
3305                        * (parsers.newline * parsers.optionalspace)^-
3306                          1
3307 parsers.line           = parsers.linechar^0 * parsers.newline
3308 parsers.nonemptyline   = parsers.line - parsers.blankline
3309
3310 parsers.chunk          = parsers.line * (parsers.optionallyindentedline
3311                                       - parsers.blankline)^0
3312
3313 parsers.css_identifier_char = R("AZ", "az", "09") + S("-_")
3314 parsers.css_identifier  = (parsers.hash + parsers.period)
3315                        * (((parsers.css_identifier_char
3316                          - parsers.dash - parsers.digit)
3317                          * parsers.css_identifier_char^1)
3318                          + (parsers.dash
3319                          * (parsers.css_identifier_char

```

```

3317             - parsers.digit)
3318             * parsers.css_identifier_char^0))
3319 parsers.attribute_name_char = parsers.any - parsers.space
3320             - parsers.squote - parsers.dquote
3321             - parsers.more - parsers.slash
3322             - parsers.equal
3323 parsers.attribute_value_char = parsers.any - parsers.dquote
3324             - parsers.more
3325
3326 -- block followed by 0 or more optionally
3327 -- indented blocks with first line indented.
3328 parsers.indented_blocks = function(bl)
3329   return Cs( bl
3330     * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
3331     * (parsers.blankline^1 + parsers.eof) )
3332 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

3333 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
3334
3335 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
3336                   * (parsers.tab + parsers.space^-3)
3337                   + parsers.space * parsers.bulletchar * #parsers.spacing
3338                   * (parsers.tab + parsers.space^-2)
3339                   + parsers.space * parsers.space * parsers.bulletchar
3340                   * #parsers.spacing
3341                   * (parsers.tab + parsers.space^-1)
3342                   + parsers.space * parsers.space * parsers.space
3343                   * parsers.bulletchar * #parsers.spacing
3344                   )

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

3345 parsers.openticks = Cg(parsers.backtick^1, "ticks")
3346
3347 local function captures_equal_length(s,i,a,b)
3348   return #a == #b and i
3349 end
3350
3351 parsers.closeticks = parsers.space^-1
3352                   * Cmt(C(parsers.backtick^1)
3353                       * Cb("ticks"), captures_equal_length)
3354
3355 parsers.intickschar = (parsers.any - S("\n\r`"))
3356                   + (parsers.newline * -parsers.blankline)
3357                   + (parsers.space - parsers.closeticks)
3358                   + (parsers.backtick^1 - parsers.closeticks)

```

```

3359
3360 parsers.inticks      = parsers.openticks * parsers.space^-1
3361                      * C(parsers.intickschar^0) * parsers.closeticks

```

#### 3.1.4.4 Parsers Used for Fenced Code Blocks

```

3362 local function captures_geq_length(s,i,a,b)
3363   return #a >= #b and i
3364 end
3365
3366 parsers.infostring    = (parsers.linechar - (parsers.backtick
3367                      + parsers.space^1 * (parsers.newline + parsers.eof)))^0
3368
3369 local fenceindent
3370 parsers.fencehead     = function(char)
3371   return              C(parsers.nonindentpace) / function(s) fenceindent = #s end
3372                      * Cg(char^3, "fencelength")
3373                      * parsers.optionalspace * C(parsers.infostring)
3374                      * parsers.optionalspace * (parsers.newline + parsers.eof)
3375 end
3376
3377 parsers.fencetail     = function(char)
3378   return              parsers.nonindentpace
3379                      * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
3380                      * parsers.optionalspace * (parsers.newline + parsers.eof)
3381                      + parsers.eof
3382 end
3383
3384 parsers.fencedline    = function(char)
3385   return              C(parsers.line - parsers.fencetail(char))
3386                      / function(s)
3387                        i = 1
3388                        remaining = fenceindent
3389                        while true do
3390                          c = s:sub(i, i)
3391                          if c == " " and remaining > 0 then
3392                            remaining = remaining - 1
3393                            i = i + 1
3394                          elseif c == "\t" and remaining > 3 then
3395                            remaining = remaining - 4
3396                            i = i + 1
3397                          else
3398                            break
3399                          end
3400                        end
3401                      return s:sub(i)
3402 end

```

3403 end

### 3.1.4.5 Parsers Used for Markdown Tags and Links

```
3404 parsers.leader      = parsers.space^-3
3405
3406 -- content in balanced brackets, parentheses, or quotes:
3407 parsers.bracketed   = P{ parsers.lbracket
3408                       * ((parsers.anyescaped - (parsers.lbracket
3409                                                         + parsers.rbracket
3410                                                         + parsers.blankline^2)
3411                          ) + V(1))^0
3412                       * parsers.rbracket }
3413
3414 parsers.inparens    = P{ parsers.lparent
3415                       * ((parsers.anyescaped - (parsers.lparent
3416                                                         + parsers.rparent
3417                                                         + parsers.blankline^2)
3418                          ) + V(1))^0
3419                       * parsers.rparent }
3420
3421 parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
3422                       * ((parsers.anyescaped - (parsers.squote
3423                                                         + parsers.blankline^2)
3424                          ) + V(1))^0
3425                       * parsers.squote }
3426
3427 parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
3428                       * ((parsers.anyescaped - (parsers.dquote
3429                                                         + parsers.blankline^2)
3430                          ) + V(1))^0
3431                       * parsers.dquote }
3432
3433 -- bracketed tag for markdown links, allowing nested brackets:
3434 parsers.tag         = parsers.lbracket
3435                       * Cs((parsers.alphanumeric^1
3436                             + parsers.bracketed
3437                             + parsers.inticks
3438                             + (parsers.anyescaped
3439                               - (parsers.rbracket + parsers.blankline^2)))^0)
3440                       * parsers.rbracket
3441
3442 -- url for markdown links, allowing nested brackets:
3443 parsers.url         = parsers.less * Cs((parsers.anyescaped
3444                                         - parsers.more)^0)
3445                       * parsers.more
3446                       + Cs((parsers.inparens + (parsers.anyescaped
```

```

3447             - parsers.spacing
3448             - parsers.rparent))^1)
3449
3450 -- quoted text, possibly with nested quotes:
3451 parsers.title_s    = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
3452                    + parsers.squoted)^0)
3453                    * parsers.squote
3454
3455 parsers.title_d    = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
3456                    + parsers.dquoted)^0)
3457                    * parsers.dquote
3458
3459 parsers.title_p    = parsers.lparent
3460                    * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
3461                    * parsers.rparent
3462
3463 parsers.title      = parsers.title_d + parsers.title_s + parsers.title_p
3464
3465 parsers.optionaltitle
3466                 = parsers.spnl * parsers.title * parsers.spacechar^0
3467                 + Cc("")

```

### 3.1.4.6 Parsers Used for iA Writer Content Blocks

```

3468 parsers.contentblock_tail
3469                 = parsers.optionaltitle
3470                 * (parsers.newline + parsers.eof)
3471
3472 -- case insensitive online image suffix:
3473 parsers.onlineimagesuffix
3474                 = (function(...)
3475                    local parser = nil
3476                    for _,suffix in ipairs({...}) do
3477                        local pattern=nil
3478                        for i=1,#suffix do
3479                            local char=suffix:sub(i,i)
3480                            char = S(char:lower()..char:upper())
3481                            if pattern == nil then
3482                                pattern = char
3483                            else
3484                                pattern = pattern * char
3485                            end
3486                        end
3487                        if parser == nil then
3488                            parser = pattern
3489                        else
3490                            parser = parser + pattern

```

```

3491         end
3492     end
3493     return parser
3494 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
3495
3496 -- online image url for iA Writer content blocks with mandatory suffix,
3497 -- allowing nested brackets:
3498 parsers.onlineimageurl
3499     = (parsers.less
3500     * Cs((parsers.anyescaped
3501     - parsers.more
3502     - #(parsers.period
3503     * parsers.onlineimagesuffix
3504     * parsers.more
3505     * parsers.contentblock_tail))^0)
3506     * parsers.period
3507     * Cs(parsers.onlineimagesuffix)
3508     * parsers.more
3509     + (Cs((parsers.inparens
3510     + (parsers.anyescaped
3511     - parsers.spacing
3512     - parsers.rparent
3513     - #(parsers.period
3514     * parsers.onlineimagesuffix
3515     * parsers.contentblock_tail))))^0)
3516     * parsers.period
3517     * Cs(parsers.onlineimagesuffix))
3518     ) * Cc("onlineimage")
3519
3520 -- filename for iA Writer content blocks with mandatory suffix:
3521 parsers.localfilepath
3522     = parsers.slash
3523     * Cs((parsers.anyescaped
3524     - parsers.tab
3525     - parsers.newline
3526     - #(parsers.period
3527     * parsers.alphanumeric^1
3528     * parsers.contentblock_tail))^1)
3529     * parsers.period
3530     * Cs(parsers.alphanumeric^1)
3531     * Cc("localfile")

```

### 3.1.4.7 Parsers Used for Citations

```

3532 parsers.citation_name = Cs(parsers.dash^-1) * parsers.at
3533     * Cs(parsers.citation_chars
3534     * ((parsers.citation_chars + parsers.internal_punctuation

```

```

3535         - parsers.comma - parsers.semicolon)
3536     * -#((parsers.internal_punctuation - parsers.comma
3537         - parsers.semicolon)^0
3538         * -(parsers.citation_chars + parsers.internal_punctuat.
3539           - parsers.comma - parsers.semicolon))^0
3540     * parsers.citation_chars)^-1)
3541
3542 parsers.citation_body_prenote
3543     = Cs((parsers.alphanumeric^1
3544         + parsers.bracketed
3545         + parsers.inticks
3546         + (parsers.anyescaped
3547           - (parsers.rbracket + parsers.blankline^2))
3548         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
3549
3550 parsers.citation_body_postnote
3551     = Cs((parsers.alphanumeric^1
3552         + parsers.bracketed
3553         + parsers.inticks
3554         + (parsers.anyescaped
3555           - (parsers.rbracket + parsers.semicolon
3556             + parsers.blankline^2))
3557         - (parsers.spnl * parsers.rbracket))^0)
3558
3559 parsers.citation_body_chunk
3560     = parsers.citation_body_prenote
3561     * parsers.spnl * parsers.citation_name
3562     * (parsers.internal_punctuation - parsers.semicolon)^-
3563     1
3564     * parsers.spnl * parsers.citation_body_postnote
3565
3566 parsers.citation_body
3567     = parsers.citation_body_chunk
3568     * (parsers.semicolon * parsers.spnl
3569     * parsers.citation_body_chunk)^0
3570
3571 parsers.citation_headless_body_postnote
3572     = Cs((parsers.alphanumeric^1
3573         + parsers.bracketed
3574         + parsers.inticks
3575         + (parsers.anyescaped
3576           - (parsers.rbracket + parsers.at
3577             + parsers.semicolon + parsers.blankline^2))
3578         - (parsers.spnl * parsers.rbracket))^0)
3579
3580 parsers.citation_headless_body
3581     = parsers.citation_headless_body_postnote

```

```

3581             * (parsers.sp * parsers.semicolon * parsers.spnl
3582             * parsers.citation_body_chunk)^0

```

### 3.1.4.8 Parsers Used for Footnotes

```

3583 local function strip_first_char(s)
3584   return s:sub(2)
3585 end
3586
3587 parsers.RawNoteRef = #(parsers.lbracket * parsers.circumflex)
3588                   * parsers.tag / strip_first_char

```

### 3.1.4.9 Parsers Used for Tables

```

3589 local function make_pipe_table_rectangular(rows)
3590   local num_columns = #rows[2]
3591   local rectangular_rows = {}
3592   for i = 1, #rows do
3593     local row = rows[i]
3594     local rectangular_row = {}
3595     for j = 1, num_columns do
3596       rectangular_row[j] = row[j] or ""
3597     end
3598     table.insert(rectangular_rows, rectangular_row)
3599   end
3600   return rectangular_rows
3601 end
3602
3603 local function pipe_table_row(allow_empty_first_column
3604                             , nonempty_column
3605                             , column_separator
3606                             , column)
3607   local row_beginning
3608   if allow_empty_first_column then
3609     row_beginning = -- empty first column
3610                   #(parsers.spacechar^4
3611                   * column_separator)
3612                   * parsers.optionalspace
3613                   * column
3614                   * parsers.optionalspace
3615                   -- non-empty first column
3616                   + parsers.nonindentSPACE
3617                   * nonempty_column^-1
3618                   * parsers.optionalspace
3619   else
3620     row_beginning = parsers.nonindentSPACE
3621                   * nonempty_column^-1
3622                   * parsers.optionalspace

```



```

3623 end
3624
3625 return Ct(row_beginning
3626     * (-- single column with no leading pipes
3627         #(column_separator
3628             * parsers.optionalspace
3629             * parsers.newline)
3630         * column_separator
3631         * parsers.optionalspace
3632     -- single column with leading pipes or
3633     -- more than a single column
3634     + (column_separator
3635         * parsers.optionalspace
3636         * column
3637         * parsers.optionalspace)^1
3638     * (column_separator
3639         * parsers.optionalspace)^-1))
3640 end
3641
3642 parsers.table_hline_separator = parsers.pipe + parsers.plus
3643 parsers.table_hline_column = (parsers.dash
3644     - #(parsers.dash
3645         * (parsers.spacechar
3646             + parsers.table_hline_separator
3647             + parsers.newline)))^1
3648     * (parsers.colon * Cc("r")
3649         + parsers.dash * Cc("d"))
3650     + parsers.colon
3651     * (parsers.dash
3652         - #(parsers.dash
3653             * (parsers.spacechar
3654                 + parsers.table_hline_separator
3655                 + parsers.newline)))^1
3656     * (parsers.colon * Cc("c")
3657         + parsers.dash * Cc("l"))
3658 parsers.table_hline = pipe_table_row(false
3659     , parsers.table_hline_column
3660     , parsers.table_hline_separator
3661     , parsers.table_hline_column)
3662 parsers.table_caption_beginning = parsers.skipblanklines
3663     * parsers.nonindentospace
3664     * (P("Table")^-1 * parsers.colon)
3665     * parsers.optionalspace

```

### 3.1.4.10 Parsers Used for HTML

3666 -- case-insensitive match (we assume s is lowercase). must be single byte encoding

```

3667 parsers.keyword_exact = function(s)
3668   local parser = P(0)
3669   for i=1,#s do
3670     local c = s:sub(i,i)
3671     local m = c .. upper(c)
3672     parser = parser * S(m)
3673   end
3674   return parser
3675 end
3676
3677 parsers.block_keyword =
3678   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
3679   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
3680   parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
3681   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
3682   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
3683   parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
3684   parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
3685   parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
3686   parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
3687   parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
3688   parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
3689   parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
3690   parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
3691   parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
3692   parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
3693   parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
3694   parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
3695   parsers.keyword_exact("td") + parsers.keyword_exact("tr")
3696
3697 -- There is no reason to support bad html, so we expect quoted attributes
3698 parsers.htmlattributevalue
3699     = parsers.squote * (parsers.any - (parsers.blankline
3700                                     + parsers.squote))^0
3701         * parsers.squote
3702     + parsers.dquote * (parsers.any - (parsers.blankline
3703                                     + parsers.dquote))^0
3704         * parsers.dquote
3705
3706 parsers.htmlattribute = parsers.spacing^1
3707     * (parsers.alphanumeric + S("_-"))^1
3708     * parsers.sp * parsers.equal * parsers.sp
3709     * parsers.htmlattributevalue
3710
3711 parsers.htmlcomment = P("<!--") * (parsers.any - P("-->"))^0 * P("-->")
3712
3713 parsers.htmlinstruction = P("<?") * (parsers.any - P(">" ))^0 * P(">" )

```

```

3714
3715 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
3716         * parsers.sp * parsers.more
3717
3718 parsers.openelt_exact = function(s)
3719     return parsers.less * parsers.sp * parsers.keyword_exact(s)
3720         * parsers.htmlattribute^0 * parsers.sp * parsers.more
3721 end
3722
3723 parsers.openelt_block = parsers.sp * parsers.block_keyword
3724         * parsers.htmlattribute^0 * parsers.sp * parsers.more
3725
3726 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
3727         * parsers.keyword * parsers.sp * parsers.more
3728
3729 parsers.closeelt_exact = function(s)
3730     return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
3731         * parsers.sp * parsers.more
3732 end
3733
3734 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
3735         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
3736         * parsers.more
3737
3738 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
3739         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
3740         * parsers.more
3741
3742 parsers.displaytext = (parsers.any - parsers.less)^1
3743
3744 -- return content between two matched HTML tags
3745 parsers.in_matched = function(s)
3746     return { parsers.openelt_exact(s)
3747         * (V(1) + parsers.displaytext
3748         + (parsers.less - parsers.closeelt_exact(s)))^0
3749         * parsers.closeelt_exact(s) }
3750 end
3751
3752 local function parse_matched_tags(s,pos)
3753     local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
3754     return lpeg.match(parsers.in_matched(t),s,pos-1)
3755 end
3756
3757 parsers.in_matched_block_tags = parsers.less
3758         * Cmt(#parsers.openelt_block, parse_matched_tags)
3759
3760 parsers.displayhtml = parsers.htmlcomment

```

```

3761         + parsers.emptyelt_block
3762         + parsers.openelt_exact("hr")
3763         + parsers.in_matched_block_tags
3764         + parsers.htmlinstruction
3765
3766 parsers.inlinehtml = parsers.emptyelt_any
3767                   + parsers.htmlcomment
3768                   + parsers.htmlinstruction
3769                   + parsers.openelt_any
3770                   + parsers.closeelt_any

```

### 3.1.4.11 Parsers Used for HTML Entities

```

3771 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
3772                   * C(parsers.hexdigit^1) * parsers.semicolon
3773 parsers.decentity = parsers.ampersand * parsers.hash
3774                   * C(parsers.digit^1) * parsers.semicolon
3775 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
3776                   * parsers.semicolon

```

### 3.1.4.12 Helpers for References

```

3777 -- parse a reference definition: [foo]: /bar "title"
3778 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
3779                                 * parsers.spacechar^0 * parsers.url
3780                                 * parsers.optionaltitle * parsers.blankline^1

```

### 3.1.4.13 Inline Elements

```

3781 parsers.Inline      = V("Inline")
3782 parsers.IndentedInline = V("IndentedInline")
3783
3784 -- parse many p between starter and ender
3785 parsers.between = function(p, starter, ender)
3786   local ender2 = B(parsers.nonspacechar) * ender
3787   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
3788 end
3789
3790 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

### 3.1.4.14 Block Elements

```

3791 parsers.Block      = V("Block")
3792
3793 parsers.OfflineImageURL
3794                   = parsers.leader
3795                   * parsers.onlineimageurl
3796                   * parsers.optionaltitle
3797

```

```

3798 parsers.LocalFilePath
3799         = parsers.leader
3800         * parsers.localfilepath
3801         * parsers.optionaltitle
3802
3803 parsers.TildeFencedCode
3804         = parsers.fencehead(parsers.tilde)
3805         * Cs(parsers.fencedline(parsers.tilde)^0)
3806         * parsers.fencetail(parsers.tilde)
3807
3808 parsers.BacktickFencedCode
3809         = parsers.fencehead(parsers.backtick)
3810         * Cs(parsers.fencedline(parsers.backtick)^0)
3811         * parsers.fencetail(parsers.backtick)
3812
3813 parsers.lineof = function(c)
3814     return (parsers.leader * (P(c) * parsers.optionalspace)^3
3815           * (parsers.newline * parsers.blankline^1
3816             + parsers.newline^-1 * parsers.eof))
3817 end

```

### 3.1.4.15 Lists

```

3818 parsers.defstartchar = S("~:")
3819 parsers.defstart     = ( parsers.defstartchar * #parsers.spacing
3820                       * (parsers.tab + parsers.space^-
3821                          3)
3822                       + parsers.space * parsers.defstartchar * #parsers.spacing
3823                       * (parsers.tab + parsers.space^-2)
3824                       + parsers.space * parsers.space * parsers.defstartchar
3825                       * #parsers.spacing
3826                       * (parsers.tab + parsers.space^-1)
3827                       + parsers.space * parsers.space * parsers.space
3828                       * parsers.defstartchar * #parsers.spacing
3829                       )
3830 parsers.dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

```

### 3.1.4.16 Headings

```

3831 parsers.heading_attribute = C(parsers.css_identifier)
3832                          + C((parsers.attribute_name_char
3833                             - parsers.rbrace)^1
3834                             * parsers.equal
3835                             * (parsers.attribute_value_char
3836                                - parsers.rbrace)^1)
3837 parsers.HeadingAttributes = parsers.lbrace
3838                          * parsers.heading_attribute

```

```

3839             * (parsers.spacechar~1
3840               * parsers.heading_attribute)^0
3841             * parsers.rbrace
3842
3843 -- parse Atx heading start and return level
3844 parsers.HeadingStart = #parsers.hash * C(parsers.hash~-6)
3845                   * -parsers.hash / length
3846
3847 -- parse setext header ending and return level
3848 parsers.HeadingLevel = parsers.equal~1 * Cc(1) + parsers.dash~1 * Cc(2)
3849
3850 local function strip_atx_end(s)
3851   return s:gsub("#%s*\n$", "")
3852 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  $\text{\TeX}$  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`.

```

3853 M.reader = {}
3854 function M.reader.new(writer, options)
3855   local self = {}
3856   options = options or {}

```

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

```

3857   setmetatable(options, { __index = function (_, key)
3858     return defaultOptions[key] end })

```

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

```

3859   local function normalize_tag(tag)

```

```

3860     return unicode.utf8.lower(
3861         gsub(util.rope_to_string(tag), "[ \n\r\t]+", " "))
3862 end

```

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

```

3863 local expandtabs
3864 if options.preserveTabs then
3865     expandtabs = function(s) return s end
3866 else
3867     expandtabs = function(s)
3868         if s:find("\t") then
3869             return s:gsub("[^\n]*", util.expand_tabs_in_line)
3870         else
3871             return s
3872         end
3873     end
3874 end

```

The `larsers` (as in `local \luam{parsers}'`) hash table stores `\acro{peg}` patterns tables, which impedes their reuse between different `reader` objects.

```

3875 local larsers = {}

```

### 3.1.5.2 Top-Level Parser Functions

```

3876 local function create_parser(name, grammar)
3877     return function(str)
3878         local res = lpeg.match(grammar(), str)
3879         if res == nil then
3880             error(format("%s failed on:\n%s", name, str:sub(1,20)))
3881         else
3882             return res
3883         end
3884     end
3885 end
3886
3887 local parse_blocks
3888     = create_parser("parse_blocks",
3889         function()
3890             return larsers.blocks
3891         end)
3892
3893 local parse_blocks_toplevel
3894     = create_parser("parse_blocks_toplevel",
3895         function()
3896             return larsers.blocks_toplevel
3897         end)
3898

```

```

3899 local parse_inlines
3900   = create_parser("parse_inlines",
3901                 function()
3902                   return larsers.inlines
3903                 end)
3904
3905 local parse_inlines_no_link
3906   = create_parser("parse_inlines_no_link",
3907                 function()
3908                   return larsers.inlines_no_link
3909                 end)
3910
3911 local parse_inlines_no_inline_note
3912   = create_parser("parse_inlines_no_inline_note",
3913                 function()
3914                   return larsers.inlines_no_inline_note
3915                 end)
3916
3917 local parse_inlines_nbsp
3918   = create_parser("parse_inlines_nbsp",
3919                 function()
3920                   return larsers.inlines_nbsp
3921                 end)

```

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

```

3922 if options.hashEnumerators then
3923   larsers.dig = parsers.digit + parsers.hash
3924 else
3925   larsers.dig = parsers.digit
3926 end
3927
3928 larsers.enumerator = C(larsers.dig^3 * parsers.period) * #parsers.spacing
3929   + C(larsers.dig^2 * parsers.period) * #parsers.spacing
3930     * (parsers.tab + parsers.space^1)
3931   + C(larsers.dig * parsers.period) * #parsers.spacing
3932     * (parsers.tab + parsers.space^-2)
3933   + parsers.space * C(larsers.dig^2 * parsers.period)
3934     * #parsers.spacing
3935   + parsers.space * C(larsers.dig * parsers.period)
3936     * #parsers.spacing
3937     * (parsers.tab + parsers.space^-1)
3938   + parsers.space * parsers.space * C(larsers.dig^1
3939     * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

3940 -- strip off leading > and indents, and run through blocks

```



```

3941 larsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
1)/""
3942         * parsers.linechar^0 * parsers.newline)^1
3943         * (-(parsers.leader * parsers.more
3944           + parsers.blankline) * parsers.linechar^1
3945           * parsers.newline)^0
3946
3947 if not options.breakableBlockquotes then
3948     larsers.blockquote_body = larsers.blockquote_body
3949     * (parsers.blankline^0 / "")
3950 end

```

### 3.1.5.5 Parsers Used for Citations (local)

```

3951 larsers.citations = function(text_cites, raw_cites)
3952     local function normalize(str)
3953         if str == "" then
3954             str = nil
3955         else
3956             str = (options.citationNbsps and parse_inlines_nbsp or
3957                 parse_inlines)(str)
3958         end
3959         return str
3960     end
3961
3962     local cites = {}
3963     for i = 1,#raw_cites,4 do
3964         cites[#cites+1] = {
3965             prenote = normalize(raw_cites[i]),
3966             suppress_author = raw_cites[i+1] == "-",
3967             name = writer.citation(raw_cites[i+2]),
3968             postnote = normalize(raw_cites[i+3]),
3969         }
3970     end
3971     return writer.citations(text_cites, cites)
3972 end

```

### 3.1.5.6 Parsers Used for Footnotes (local)

```

3973 local rawnotes = {}
3974
3975 -- like indirect_link
3976 local function lookup_note(ref)
3977     return function()
3978         local found = rawnotes[normalize_tag(ref)]
3979         if found then
3980             return writer.note(parse_blocks_toplevel(found))
3981         else

```

```

3982         return {"[", parse_inlines("^" .. ref), "]" }
3983     end
3984 end
3985 end
3986
3987 local function register_note(ref,rawnote)
3988     rawnotes[normalize_tag(ref)] = rawnote
3989     return ""
3990 end
3991
3992 larsers.NoteRef      = parsers.RawNoteRef / lookup_note
3993
3994
3995 larsers.NoteBlock   = parsers.leader * parsers.RawNoteRef * parsers.colon
3996                     * parsers.spnl * parsers.indented_blocks(parsers.chunk)
3997                     / register_note
3998
3999 larsers.InlineNote  = parsers.circumflex
4000                     * (parsers.tag / parse_inlines_no_inline_note) -- no notes inside
4001                     / writer.note

```

### 3.1.5.7 Parsers Used for Tables (local)

```

4002 larsers.table_row = pipe_table_row(true
4003                               , (C((parsers.linechar - parsers.pipe)^1)
4004                               / parse_inlines)
4005                               , parsers.pipe
4006                               , (C((parsers.linechar - parsers.pipe)^0)
4007                               / parse_inlines))
4008
4009 if options.tableCaptions then
4010     larsers.table_caption = #parsers.table_caption_beginning
4011                           * parsers.table_caption_beginning
4012                           * Ct(parsers.IndentedInline^1)
4013                           * parsers.newline
4014 else
4015     larsers.table_caption = parsers.fail
4016 end
4017
4018 larsers.PipeTable = Ct(larsers.table_row * parsers.newline
4019                       * parsers.table_hline
4020                       * (parsers.newline * larsers.table_row)^0)
4021                       / make_pipe_table_rectangular
4022                       * larsers.table_caption^-1
4023                       / writer.table

```

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

```

4024 -- List of references defined in the document
4025 local references
4026
4027 -- add a reference to the list
4028 local function register_link(tag,url,title)
4029     references[normalize_tag(tag)] = { url = url, title = title }
4030     return ""
4031 end
4032
4033 -- lookup link reference and return either
4034 -- the link or nil and fallback text.
4035 local function lookup_reference(label,sps,tag)
4036     local tagpart
4037     if not tag then
4038         tag = label
4039         tagpart = ""
4040     elseif tag == "" then
4041         tag = label
4042         tagpart = "[]"
4043     else
4044         tagpart = {"[", parse_inlines(tag), "]" }
4045     end
4046     if sps then
4047         tagpart = {sps, tagpart}
4048     end
4049     local r = references[normalize_tag(tag)]
4050     if r then
4051         return r
4052     else
4053         return nil, {"[", parse_inlines(label), "]" , tagpart}
4054     end
4055 end
4056
4057 -- lookup link reference and return a link, if the reference is found,
4058 -- or a bracketed label otherwise.
4059 local function indirect_link(label,sps,tag)
4060     return function()
4061         local r,fallback = lookup_reference(label,sps,tag)
4062         if r then
4063             return writer.link(parse_inlines_no_link(label), r.url, r.title)
4064         else
4065             return fallback
4066         end
4067     end
4068 end
4069
4070 -- lookup image reference and return an image, if the reference is found,

```

```

4071 -- or a bracketed label otherwise.
4072 local function indirect_image(label,sps,tag)
4073     return function()
4074         local r,fallback = lookup_reference(label,sps,tag)
4075         if r then
4076             return writer.image(writer.string(label), r.url, r.title)
4077         else
4078             return {"!", fallback}
4079         end
4080     end
4081 end

```

### 3.1.5.9 Inline Elements (local)

```

4082 larsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
4083                 / writer.string
4084
4085 larsers.Symbol   = (parsers.specialchar - parsers.tightblocksep)
4086                 / writer.string
4087
4088 larsers.Ellipsis = P("...") / writer.ellipsis
4089
4090 larsers.Smart    = larsers.Ellipsis
4091
4092 larsers.Code     = parsers.inticks / writer.code
4093
4094 if options.blankBeforeBlockquote then
4095     larsers.bqstart = parsers.fail
4096 else
4097     larsers.bqstart = parsers.more
4098 end
4099
4100 if options.blankBeforeHeading then
4101     larsers.headerstart = parsers.fail
4102 else
4103     larsers.headerstart = parsers.hash
4104                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
4105                         * parsers.optionalspace * parsers.newline)
4106 end
4107
4108 if not options.fencedCode or options.blankBeforeCodeFence then
4109     larsers.fencestart = parsers.fail
4110 else
4111     larsers.fencestart = parsers.fencehead(parsers.backtick)
4112                       + parsers.fencehead(parsers.tilde)
4113 end
4114

```

```

4115 larsers.Endline = parsers.newline * -( -- newline, but not before...
4116     parsers.blankline -- paragraph break
4117     + parsers.tightblocksep -- nested list
4118     + parsers.eof      -- end of document
4119     + larsers.bqstart
4120     + larsers.headerstart
4121     + larsers.fencestart
4122     ) * parsers.spacechar^0 / writer.space
4123
4124 larsers.OptionalIndent
4125     = parsers.spacechar^1 / writer.space
4126
4127 larsers.Space = parsers.spacechar^2 * larsers.Endline / writer.linebreak
4128     + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
4129     + parsers.spacechar^1 * larsers.Endline^-1
4130     * parsers.optionalspace / writer.space
4131
4132 larsers.NonbreakingEndline
4133     = parsers.newline * -( -- newline, but not before...
4134     parsers.blankline -- paragraph break
4135     + parsers.tightblocksep -- nested list
4136     + parsers.eof      -- end of document
4137     + larsers.bqstart
4138     + larsers.headerstart
4139     + larsers.fencestart
4140     ) * parsers.spacechar^0 / writer.nbsp
4141
4142 larsers.NonbreakingSpace
4143     = parsers.spacechar^2 * larsers.Endline / writer.linebreak
4144     + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
4145     + parsers.spacechar^1 * larsers.Endline^-1
4146     * parsers.optionalspace / writer.nbsp
4147
4148 if options.underscores then
4149     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
4150     parsers.doubleasterisks)
4151     + parsers.between(parsers.Inline, parsers.doubleunderscores,
4152     parsers.doubleunderscores)
4153     ) / writer.strong
4154
4155     larsers.Emph = ( parsers.between(parsers.Inline, parsers.asterisk,
4156     parsers.asterisk)
4157     + parsers.between(parsers.Inline, parsers.underscore,
4158     parsers.underscore)
4159     ) / writer.emphasis
4160 else
4161     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,

```

```

4162                 parsers.doubleasterisks)
4163             ) / writer.strong
4164
4165     larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
4166                                     parsers.asterisk)
4167                   ) / writer.emphasis
4168 end
4169
4170 larsers.AutoLinkUrl   = parsers.less
4171                       * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
4172                       * parsers.more
4173                       / function(url)
4174                           return writer.link(writer.string(url), url)
4175                       end
4176
4177 larsers.AutoLinkEmail = parsers.less
4178                       * C((parsers.alphanumeric + S("-._+"))^1
4179                           * P("@") * parsers.urlchar^1)
4180                       * parsers.more
4181                       / function(email)
4182                           return writer.link(writer.string(email),
4183                                               "mailto:".email)
4184                       end
4185
4186 larsers.DirectLink   = (parsers.tag / parse_inlines_no_link) -- no links inside link
4187                       * parsers.spnl
4188                       * parsers.lparent
4189                       * (parsers.url + Cc("")) -- link can be empty [foo]()
4190                       * parsers.optionaltitle
4191                       * parsers.rparent
4192                       / writer.link
4193
4194 larsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
4195
4196
4197 -- parse a link or image (direct or indirect)
4198 larsers.Link         = larsers.DirectLink + larsers.IndirectLink
4199
4200 larsers.DirectImage  = parsers.exclamation
4201                       * (parsers.tag / parse_inlines)
4202                       * parsers.spnl
4203                       * parsers.lparent
4204                       * (parsers.url + Cc("")) -- link can be empty [foo]()
4205                       * parsers.optionaltitle
4206                       * parsers.rparent
4207                       / writer.image

```

```

4208
4209 larsers.IndirectImage = parsers.exclamation * parsers.tag
4210                       * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
4211
4212 larsers.Image         = larsers.DirectImage + larsers.IndirectImage
4213
4214 larsers.TextCitations = Ct(Cc(""))
4215                       * parsers.citation_name
4216                       * ((parsers.spnl
4217                           * parsers.lbracket
4218                           * parsers.citation_headless_body
4219                           * parsers.rbracket) + Cc(""))))
4220                       / function(raw_cites)
4221                           return larsers.citations(true, raw_cites)
4222                       end
4223
4224 larsers.ParenthesizedCitations
4225     = Ct(parsers.lbracket
4226         * parsers.citation_body
4227         * parsers.rbracket)
4228     / function(raw_cites)
4229         return larsers.citations(false, raw_cites)
4230     end
4231
4232 larsers.Citations     = larsers.TextCitations + larsers.ParenthesizedCitations
4233
4234 -- avoid parsing long strings of * or _ as emph/strong
4235 larsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
4236                       / writer.string
4237
4238 larsers.EscapedChar   = S("\\") * C(parsers.escapable) / writer.string
4239
4240 larsers.InlineHtml    = C(parsers.inlinehtml) / writer.inline_html
4241
4242 larsers.HtmlEntity    = parsers.hexentity / entities.hex_entity / writer.string
4243                       + parsers.decentity / entities.dec_entity / writer.string
4244                       + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.10 Block Elements (local)

```

4245 larsers.ContentBlock = parsers.leader
4246                       * (parsers.localfilepath + parsers.onlineimageurl)
4247                       * parsers.contentblock_tail
4248                       / writer.contentblock
4249
4250 larsers.DisplayHtml  = C(parsers.displayhtml)
4251                       / expandtabs / writer.display_html

```

```

4252
4253 larsers.Verbatim = Cs( (parsers.blanklines
4254 * ((parsers.indentedline - parsers.blankline))^1)^1
4255 ) / expandtabs / writer.verbatim
4256
4257 larsers.FencedCode = (parsers.TildeFencedCode
4258 + parsers.BacktickFencedCode)
4259 / function(infostring, code)
4260 return writer.fencedCode(writer.string(infostring),
4261 expandtabs(code))
4262 end
4263
4264 larsers.Blockquote = Cs(larsers.blockquote_body^1)
4265 / parse_blocks_toplevel / writer.blockquote
4266
4267 larsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
4268 + parsers.lineof(parsers.dash)
4269 + parsers.lineof(parsers.underscore)
4270 ) / writer.hrule
4271
4272 larsers.Reference = parsers.define_reference_parser / register_link
4273
4274 larsers.Paragraph = parsers.nonindentpace * Ct(parsers.Inline^1)
4275 * parsers.newline
4276 * ( parsers.blankline^1
4277 + #parsers.hash
4278 + #(parsers.leader * parsers.more * parsers.space^-
4279 1)
4280 )
4281 / writer.paragraph
4282
4283 larsers.ToplevelParagraph
4284 = parsers.nonindentpace * Ct(parsers.Inline^1)
4285 * ( parsers.newline
4286 * ( parsers.blankline^1
4287 + #parsers.hash
4288 + #(parsers.leader * parsers.more * parsers.space^-
4289 1)
4290 + parsers.eof
4291 )
4292 + parsers.eof )
4293 / writer.paragraph
4294
4295 larsers.Plain = parsers.nonindentpace * Ct(parsers.Inline^1)
4296 / writer.plain

```



### 3.1.5.11 Lists (local)

```
4295 larsers.starter = parsers.bullet + larsers.enumerator
4296
4297 -- we use \001 as a separator between a tight list item and a
4298 -- nested list under it.
4299 larsers.NestedList      = Cs((parsers.optionallyindentedline
4300                          - larsers.starter)^1)
4301                          / function(a) return "\001"..a end
4302
4303 larsers.ListBlockLine  = parsers.optionallyindentedline
4304                          - parsers.blankline - (parsers.indent^-1
4305                                                  * larsers.starter)
4306
4307 larsers.ListBlock      = parsers.line * larsers.ListBlockLine^0
4308
4309 larsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
4310                                                  * larsers.ListBlock
4311
4312 larsers.TightListItem = function(starter)
4313   return -larsers.HorizontalRule
4314          * (Cs(starter / "" * larsers.ListBlock * larsers.NestedList^-
4315 1)
4316          / parse_blocks)
4317          * -(parsers.blanklines * parsers.indent)
4318 end
4319
4319 larsers.LooseListItem = function(starter)
4320   return -larsers.HorizontalRule
4321          * Cs( starter / "" * larsers.ListBlock * Cc("\n")
4322          * (larsers.NestedList + larsers.ListContinuationBlock^0)
4323          * (parsers.blanklines / "\n\n")
4324          ) / parse_blocks
4325 end
4326
4327 larsers.BulletList = ( Ct(larsers.TightListItem(parsers.bullet)^1) * Cc(true)
4328                      * parsers.skipblanklines * -parsers.bullet
4329                      + Ct(larsers.LooseListItem(parsers.bullet)^1) * Cc(false)
4330                      * parsers.skipblanklines )
4331                      / writer.bulletlist
4332
4333 local function ordered_list(items,tight,startNumber)
4334   if options.startNumber then
4335     startNumber = tonumber(startNumber) or 1 -- fallback for '#'
4336     if startNumber ~= nil then
4337       startNumber = math.floor(startNumber)
4338     end
4339   else
```

```

4340     startNumber = nil
4341     end
4342     return writer.orderedlist(items,tight,startNumber)
4343 end
4344
4345 larsers.OrderedList = Cg(larsers.enumerator, "listtype") *
4346     ( Ct(larsers.TightListItem(Cb("listtype")))
4347       * larsers.TightListItem(larsers.enumerator)^0)
4348     * Cc(true) * parsers.skipblanklines * -larsers.enumerator
4349     + Ct(larsers.LooseListItem(Cb("listtype")))
4350       * larsers.LooseListItem(larsers.enumerator)^0)
4351     * Cc(false) * parsers.skipblanklines
4352     ) * Cb("listtype") / ordered_list
4353
4354 local function definition_list_item(term, defs, tight)
4355     return { term = parse_inlines(term), definitions = defs }
4356 end
4357
4358 larsers.DefinitionListItemLoose = C(parsers.line) * parsers.skipblanklines
4359     * Ct((parsers.defstart
4360         * parsers.indented_blocks(parsers.dlchunk)
4361         / parse_blocks_toplevel)^1)
4362     * Cc(false) / definition_list_item
4363
4364 larsers.DefinitionListItemTight = C(parsers.line)
4365     * Ct((parsers.defstart * parsers.dlchunk
4366         / parse_blocks)^1)
4367     * Cc(true) / definition_list_item
4368
4369 larsers.DefinitionList = ( Ct(larsers.DefinitionListItemLoose^1) * Cc(false)
4370     + Ct(larsers.DefinitionListItemTight^1)
4371     * (parsers.skipblanklines
4372       * -larsers.DefinitionListItemLoose * Cc(true))
4373     ) / writer.definitionlist

```

### 3.1.5.12 Blank (local)

```

4374 larsers.Blank           = parsers.blankline / ""
4375                        + larsers.NoteBlock
4376                        + larsers.Reference
4377                        + (parsers.tightblocksep / "\n")

```

### 3.1.5.13 Headings (local)

```

4378 -- parse atx header
4379 if options.headerAttributes then
4380     larsers.AtHeading = Cg(parsers.HeadingStart,"level")
4381                        * parsers.optionalspace

```

```

4382         * (C(((parsers.linechar
4383             - ((parsers.hash^1
4384                 * parsers.optionalspace
4385                 * parsers.HeadingAttributes^-1
4386                 + parsers.HeadingAttributes)
4387                 * parsers.optionalspace
4388                 * parsers.newline))
4389             * (parsers.linechar
4390               - parsers.hash
4391               - parsers.lbrace)^0)^1)
4392         / parse_inlines)
4393     * Cg(Ct(parsers.newline
4394         + (parsers.hash^1
4395           * parsers.optionalspace
4396           * parsers.HeadingAttributes^-1
4397           + parsers.HeadingAttributes)
4398           * parsers.optionalspace
4399           * parsers.newline), "attributes")
4400     * Cb("level")
4401     * Cb("attributes")
4402     / writer.heading
4403
4404     larsers.SettextHeading = #(parsers.line * S("--"))
4405         * (C(((parsers.linechar
4406             - (parsers.HeadingAttributes
4407               * parsers.optionalspace
4408               * parsers.newline))
4409             * (parsers.linechar
4410               - parsers.lbrace)^0)^1)
4411         / parse_inlines)
4412     * Cg(Ct(parsers.newline
4413         + (parsers.HeadingAttributes
4414           * parsers.optionalspace
4415           * parsers.newline)), "attributes")
4416     * parsers.HeadingLevel
4417     * Cb("attributes")
4418     * parsers.optionalspace
4419     * parsers.newline
4420     / writer.heading
4421 else
4422     larsers.AtXHeading = Cg(parsers.HeadingStart, "level")
4423     * parsers.optionalspace
4424     * (C(parsers.line) / strip_atx_end / parse_inlines)
4425     * Cb("level")
4426     / writer.heading
4427
4428     larsers.SettextHeading = #(parsers.line * S("--"))

```

```

4429         * Ct(parsers.linechar1 / parse_inlines)
4430         * parsers.newline
4431         * parsers.HeadingLevel
4432         * parsers.optionalspace
4433         * parsers.newline
4434         / writer.heading
4435     end
4436
4437     larsers.Heading = larsers.AtxHeading + larsers.SetextHeading

```

### 3.1.5.14 Syntax Specification

```

4438     local syntax =
4439         { "Blocks",
4440
4441           Blocks          = larsers.Blank0 * parsers.Block-1
4442                           * (larsers.Blank0 / writer.interblocksep
4443                               * parsers.Block)0
4444                           * larsers.Blank0 * parsers.eof,
4445
4446           Blank          = larsers.Blank,
4447
4448           Block          = V("ContentBlock")
4449                           + V("Blockquote")
4450                           + V("PipeTable")
4451                           + V("Verbatim")
4452                           + V("FencedCode")
4453                           + V("HorizontalRule")
4454                           + V("BulletList")
4455                           + V("OrderedList")
4456                           + V("Heading")
4457                           + V("DefinitionList")
4458                           + V("DisplayHtml")
4459                           + V("Paragraph")
4460                           + V("Plain"),
4461
4462           ContentBlock   = larsers.ContentBlock,
4463           Blockquote     = larsers.Blockquote,
4464           Verbatim       = larsers.Verbatim,
4465           FencedCode     = larsers.FencedCode,
4466           HorizontalRule = larsers.HorizontalRule,
4467           BulletList     = larsers.BulletList,
4468           OrderedList    = larsers.OrderedList,
4469           Heading        = larsers.Heading,
4470           DefinitionList = larsers.DefinitionList,
4471           DisplayHtml    = larsers.DisplayHtml,
4472           Paragraph      = larsers.Paragraph,

```

```

4473     PipeTable           = larsers.PipeTable,
4474     Plain                = larsers.Plain,
4475
4476     Inline               = V("Str")
4477                         + V("Space")
4478                         + V("Endline")
4479                         + V("U1OrStarLine")
4480                         + V("Strong")
4481                         + V("Emph")
4482                         + V("InlineNote")
4483                         + V("NoteRef")
4484                         + V("Citations")
4485                         + V("Link")
4486                         + V("Image")
4487                         + V("Code")
4488                         + V("AutoLinkUrl")
4489                         + V("AutoLinkEmail")
4490                         + V("InlineHtml")
4491                         + V("HtmlEntity")
4492                         + V("EscapedChar")
4493                         + V("Smart")
4494                         + V("Symbol"),
4495
4496     IndentedInline       = V("Str")
4497                         + V("OptionalIndent")
4498                         + V("Endline")
4499                         + V("U1OrStarLine")
4500                         + V("Strong")
4501                         + V("Emph")
4502                         + V("InlineNote")
4503                         + V("NoteRef")
4504                         + V("Citations")
4505                         + V("Link")
4506                         + V("Image")
4507                         + V("Code")
4508                         + V("AutoLinkUrl")
4509                         + V("AutoLinkEmail")
4510                         + V("InlineHtml")
4511                         + V("HtmlEntity")
4512                         + V("EscapedChar")
4513                         + V("Smart")
4514                         + V("Symbol"),
4515
4516     Str                   = larsers.Str,
4517     Space                 = larsers.Space,
4518     OptionalIndent       = larsers.OptionalIndent,
4519     Endline               = larsers.Endline,

```

```

4520     U1OrStarLine      = larsers.U1OrStarLine,
4521     Strong             = larsers.Strong,
4522     Emph               = larsers.Emph,
4523     InlineNote        = larsers.InlineNote,
4524     NoteRef           = larsers.NoteRef,
4525     Citations         = larsers.Citations,
4526     Link               = larsers.Link,
4527     Image             = larsers.Image,
4528     Code               = larsers.Code,
4529     AutoLinkUrl       = larsers.AutoLinkUrl,
4530     AutoLinkEmail     = larsers.AutoLinkEmail,
4531     InlineHtml        = larsers.InlineHtml,
4532     HtmlEntity        = larsers.HtmlEntity,
4533     EscapedChar       = larsers.EscapedChar,
4534     Smart              = larsers.Smart,
4535     Symbol            = larsers.Symbol,
4536   }
4537
4538   if not options.citations then
4539     syntax.Citations = parsers.fail
4540   end
4541
4542   if not options.contentBlocks then
4543     syntax.ContentBlock = parsers.fail
4544   end
4545
4546   if not options.codeSpans then
4547     syntax.Code = parsers.fail
4548   end
4549
4550   if not options.definitionLists then
4551     syntax.DefinitionList = parsers.fail
4552   end
4553
4554   if not options.fencedCode then
4555     syntax.FencedCode = parsers.fail
4556   end
4557
4558   if not options.footnotes then
4559     syntax.NoteRef = parsers.fail
4560   end
4561
4562   if not options.html then
4563     syntax.DisplayHtml = parsers.fail
4564     syntax.InlineHtml = parsers.fail
4565     syntax.HtmlEntity = parsers.fail
4566   end

```

```

4567
4568   if not options.inlineFootnotes then
4569       syntax.InlineNote = parsers.fail
4570   end
4571
4572   if not options.smartEllipses then
4573       syntax.Smart = parsers.fail
4574   end
4575
4576   if not options.pipeTables then
4577       syntax.PipeTable = parsers.fail
4578   end
4579
4580   local blocks_toplevel_t = util.table_copy(syntax)
4581   blocks_toplevel_t.Paragraph = larsers.ToplevelParagraph
4582   larsers.blocks_toplevel = Ct(blocks_toplevel_t)
4583
4584   larsers.blocks = Ct(syntax)
4585
4586   local inlines_t = util.table_copy(syntax)
4587   inlines_t[1] = "Inlines"
4588   inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
4589   larsers.inlines = Ct(inlines_t)
4590
4591   local inlines_no_link_t = util.table_copy(inlines_t)
4592   inlines_no_link_t.Link = parsers.fail
4593   larsers.inlines_no_link = Ct(inlines_no_link_t)
4594
4595   local inlines_no_inline_note_t = util.table_copy(inlines_t)
4596   inlines_no_inline_note_t.InlineNote = parsers.fail
4597   larsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
4598
4599   local inlines_nbsp_t = util.table_copy(inlines_t)
4600   inlines_nbsp_t.Endline = larsers.NonbreakingEndline
4601   inlines_nbsp_t.Space = larsers.NonbreakingSpace
4602   larsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

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

```

4603   function self.convert(input)
4604       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.

```

4605     local opt_string = {}
4606     for k,_ in pairs(defaultOptions) do
4607         local v = options[k]
4608         if k ~= "cacheDir" then
4609             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4610         end
4611     end
4612     table.sort(opt_string)
4613     local salt = table.concat(opt_string, ",") .. "," .. metadata.version

```

Produce the cache file and transform its filename to plain T<sub>E</sub>X output via the `writer->pack` method.

```

4614     local name = util.cache(options.cacheDir, input, salt, function(input)
4615         return util.ropetostring(parse_blocks_toplevel(input)) .. writer.eof
4616     end, ".md" .. writer.suffix)
4617     local output = writer.pack(name)

```

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

```

4618     if options.finalizeCache then
4619         local file, mode
4620         if options.frozenCacheCounter > 0 then
4621             mode = "a"
4622         else
4623             mode = "w"
4624         end
4625         file = assert(io.open(options.frozenCacheFileName, mode))
4626         assert(file:write([[\\expandafter\\def\\csname markdownFrozenCache]] ..
4627             options.frozenCacheCounter .. [[\\endcsname{}]] .. output .. [[]] ..
4628             "\\n"))
4629         assert(file:close())
4630     end
4631     return output
4632 end
4633 return self
4634 end

```

### 3.1.6 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`.

```

4635 function M.new(options)
4636     local writer = M.writer.new(options)
4637     local reader = M.reader.new(writer, options)
4638     return reader.convert

```



```

4639 end
4640
4641 return M

```

### 3.1.7 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.5.

```

4642
4643 local input
4644 if input_filename then
4645   local input_file = assert(io.open(input_filename, "r"))
4646   input = assert(input_file:read("*a"))
4647   assert(input_file:close())
4648 else
4649   input = assert(io.read("*a"))
4650 end
4651

```

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

```

4652 local lfs = require("lfs")
4653 if options.cacheDir and not lfs.isdir(options.cacheDir) then
4654   assert(lfs.mkdir(options["cacheDir"]))
4655 end
4656
4657 local kpse = require("kpse")
4658 kpse.set_program_name("luatex")
4659 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.

```

4660 if metadata.version ~= md.metadata.version then
4661   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
4662     "markdown.lua " .. md.metadata.version .. ".")
4663 end
4664 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.

```

4665 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
4666
4667 if output_filename then
4668   local output_file = assert(io.open(output_filename, "w"))
4669   assert(output_file:write(output))
4670   assert(output_file:close())
4671 else
4672   assert(io.write(output))
4673 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

```
4674 \def\markdownInfo#1{%
4675   \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
4676 \def\markdownWarning#1{%
4677   \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
4678 \def\markdownError#1#2{%
4679   \errhelp{#2.}}%
4680   \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
```

### 3.2.2 Finalizing and Freezing the Cache

When the `\markdownOptionFinalizeCache` option is enabled, then the `\markdownFrozenCacheCounter` counter is used to enumerate the markdown documents using the Lua interface `frozenCacheCounter` option.

When the `\markdownOptionFrozenCache` option is enabled, then the `\markdownFrozenCacheCounter` counter is used to render markdown documents from the frozen cache without invoking Lua.

```
4681 \newcount\markdownFrozenCacheCounter
```

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
4682 \def\markdownRendererInterblockSeparatorPrototype{\par}%
4683 \def\markdownRendererLineBreakPrototype{\hfil\break}%
4684 \let\markdownRendererEllipsisPrototype\dots
4685 \def\markdownRendererNbspPrototype{~}%
4686 \def\markdownRendererLeftBracePrototype{\char`\{}%
4687 \def\markdownRendererRightBracePrototype{\char`\}}%
4688 \def\markdownRendererDollarSignPrototype{\char`$}%
4689 \def\markdownRendererPercentSignPrototype{\char`\}%
4690 \def\markdownRendererAmpersandPrototype{\&}%
4691 \def\markdownRendererUnderscorePrototype{\char`_%}
4692 \def\markdownRendererHashPrototype{\char`\#}%
4693 \def\markdownRendererCircumflexPrototype{\char`^}%
4694 \def\markdownRendererBackslashPrototype{\char`\}%
4695 \def\markdownRendererTildePrototype{\char`~}%
4696 \def\markdownRendererPipePrototype{|}%
4697 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
```

```

4698 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
4699 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
4700   \markdownInput{#3}}%
4701 \def\markdownRendererContentBlockOnlineImagePrototype{%
4702   \markdownRendererImage}%
4703 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
4704   \markdownRendererInputFencedCode{#3}{#2}}%
4705 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
4706 \def\markdownRendererUlBeginPrototype{}%
4707 \def\markdownRendererUlBeginTightPrototype{}%
4708 \def\markdownRendererUlItemPrototype{}%
4709 \def\markdownRendererUlItemEndPrototype{}%
4710 \def\markdownRendererUlEndPrototype{}%
4711 \def\markdownRendererUlEndTightPrototype{}%
4712 \def\markdownRendererOlBeginPrototype{}%
4713 \def\markdownRendererOlBeginTightPrototype{}%
4714 \def\markdownRendererOlItemPrototype{}%
4715 \def\markdownRendererOlItemWithNumberPrototype#1{}%
4716 \def\markdownRendererOlItemEndPrototype{}%
4717 \def\markdownRendererOlEndPrototype{}%
4718 \def\markdownRendererOlEndTightPrototype{}%
4719 \def\markdownRendererDlBeginPrototype{}%
4720 \def\markdownRendererDlBeginTightPrototype{}%
4721 \def\markdownRendererDlItemPrototype#1{#1}%
4722 \def\markdownRendererDlItemEndPrototype{}%
4723 \def\markdownRendererDlDefinitionBeginPrototype{}%
4724 \def\markdownRendererDlDefinitionEndPrototype{\par}%
4725 \def\markdownRendererDlEndPrototype{}%
4726 \def\markdownRendererDlEndTightPrototype{}%
4727 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
4728 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
4729 \def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
4730 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
4731 \def\markdownRendererInputVerbatimPrototype#1{%
4732   \par{\tt\input#1\relax{}}\par}%
4733 \def\markdownRendererInputFencedCodePrototype#1#2{%
4734   \markdownRendererInputVerbatimPrototype{#1}}%
4735 \def\markdownRendererHeadingOnePrototype#1{#1}%
4736 \def\markdownRendererHeadingTwoPrototype#1{#1}%
4737 \def\markdownRendererHeadingThreePrototype#1{#1}%
4738 \def\markdownRendererHeadingFourPrototype#1{#1}%
4739 \def\markdownRendererHeadingFivePrototype#1{#1}%
4740 \def\markdownRendererHeadingSixPrototype#1{#1}%
4741 \def\markdownRendererHorizontalRulePrototype{}%
4742 \def\markdownRendererFootnotePrototype#1{#1}%
4743 \def\markdownRendererCitePrototype#1{}%
4744 \def\markdownRendererTextCitePrototype#1{}%

```

### 3.2.4 Lua Snippets

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

```
4745 \def\markdownLuaOptions{%
4746 \ifx\markdownOptionBlankBeforeBlockquote\undefined\else
4747   blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
4748 \fi
4749 \ifx\markdownOptionBlankBeforeCodeFence\undefined\else
4750   blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
4751 \fi
4752 \ifx\markdownOptionBlankBeforeHeading\undefined\else
4753   blankBeforeHeading = \markdownOptionBlankBeforeHeading,
4754 \fi
4755 \ifx\markdownOptionBreakableBlockquotes\undefined\else
4756   breakableBlockquotes = \markdownOptionBreakableBlockquotes,
4757 \fi
4758   cacheDir = "\markdownOptionCacheDir",
4759 \ifx\markdownOptionCitations\undefined\else
4760   citations = \markdownOptionCitations,
4761 \fi
4762 \ifx\markdownOptionCitationNbsps\undefined\else
4763   citationNbsps = \markdownOptionCitationNbsps,
4764 \fi
4765 \ifx\markdownOptionCodeSpans\undefined\else
4766   codeSpans = \markdownOptionCodeSpans,
4767 \fi
4768 \ifx\markdownOptionContentBlocks\undefined\else
4769   contentBlocks = \markdownOptionContentBlocks,
4770 \fi
4771 \ifx\markdownOptionContentBlocksLanguageMap\undefined\else
4772   contentBlocksLanguageMap =
4773     "\markdownOptionContentBlocksLanguageMap",
4774 \fi
4775 \ifx\markdownOptionDefinitionLists\undefined\else
4776   definitionLists = \markdownOptionDefinitionLists,
4777 \fi
4778 \ifx\markdownOptionFinalizeCache\undefined\else
4779   finalizeCache = \markdownOptionFinalizeCache,
4780 \fi
4781   frozenCacheFileName = "\markdownOptionFrozenCacheFileName",
4782   frozenCacheCounter = \the\markdownFrozenCacheCounter,
4783 \ifx\markdownOptionFootnotes\undefined\else
4784   footnotes = \markdownOptionFootnotes,
4785 \fi
4786 \ifx\markdownOptionFencedCode\undefined\else
4787   fencedCode = \markdownOptionFencedCode,
```

```

4788 \fi
4789 \ifx\markdownOptionHashEnumerators\undefined\else
4790   hashEnumerators = \markdownOptionHashEnumerators,
4791 \fi
4792 \ifx\markdownOptionHeaderAttributes\undefined\else
4793   headerAttributes = \markdownOptionHeaderAttributes,
4794 \fi
4795 \ifx\markdownOptionHtml\undefined\else
4796   html = \markdownOptionHtml,
4797 \fi
4798 \ifx\markdownOptionHybrid\undefined\else
4799   hybrid = \markdownOptionHybrid,
4800 \fi
4801 \ifx\markdownOptionInlineFootnotes\undefined\else
4802   inlineFootnotes = \markdownOptionInlineFootnotes,
4803 \fi
4804 \ifx\markdownOptionPipeTables\undefined\else
4805   pipeTables = \markdownOptionPipeTables,
4806 \fi
4807 \ifx\markdownOptionPreserveTabs\undefined\else
4808   preserveTabs = \markdownOptionPreserveTabs,
4809 \fi
4810 \ifx\markdownOptionShiftHeadings\undefined\else
4811   shiftHeadings = "\markdownOptionShiftHeadings",
4812 \fi
4813 \ifx\markdownOptionSlice\undefined\else
4814   slice = "\markdownOptionSlice",
4815 \fi
4816 \ifx\markdownOptionSmartEllipses\undefined\else
4817   smartEllipses = \markdownOptionSmartEllipses,
4818 \fi
4819 \ifx\markdownOptionStartNumber\undefined\else
4820   startNumber = \markdownOptionStartNumber,
4821 \fi
4822 \ifx\markdownOptionTableCaptions\undefined\else
4823   tableCaptions = \markdownOptionTableCaptions,
4824 \fi
4825 \ifx\markdownOptionTightLists\undefined\else
4826   tightLists = \markdownOptionTightLists,
4827 \fi
4828 \ifx\markdownOptionUnderscores\undefined\else
4829   underscores = \markdownOptionUnderscores,
4830 \fi}
4831 }%

```

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.

```
4832 \def\markdownPrepare{%
```

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

```
4833 local lfs = require("lfs")
4834 local cacheDir = "\markdownOptionCacheDir"
4835 if not lfs.isdir(cacheDir) then
4836     assert(lfs.mkdir(cacheDir))
4837 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.

```
4838 local md = require("markdown")
4839 local convert = md.new(\markdownLuaOptions)
4840 }%
```

### 3.2.5 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.

```
4841 \def\markdownIfOption#1#2#3{%
4842     \begingroup
4843     \def\next{true}%
4844     \expandafter\ifx\csname markdownOption#1\endcsname\next
4845     \endgroup#2\else\endgroup#3\fi}%
```

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.

```
4846 \csname newread\endcsname\markdownInputFileStream
4847 \csname newwrite\endcsname\markdownOutputFileStream
```

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

```
4848 \begingroup
4849 \catcode\^^I=12%
4850 \gdef\markdownReadAndConvertTab{^^I}%
4851 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```
4852 \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.

```
4853 \catcode\^^M=13%
4854 \catcode\^^I=13%
4855 \catcode|=0%
4856 \catcode\\=12%
4857 |catcode`=14%
4858 |catcode`|=12@
4859 |gdef|markdownReadAndConvert#1#2{@
4860 |begingroup@
```

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

```
4861 |markdownIfOption{FrozenCache}{@
4862 |immediate|openout|markdownOutputFileStream@
4863 |markdownOptionInputTempFileName|relax@
4864 |markdownInfo{Buffering markdown input into the temporary @
4865 |input file "|markdownOptionInputTempFileName" and scanning @
4866 |for the closing token sequence "#1"}@
4867 }@
```

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

```
4868 |def|do##1{|catcode`##1=12}|dospecials@
4869 |catcode`=12@
4870 |markdownMakeOther@
```

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

```
4871 |def|markdownReadAndConvertStripPercentSign##1{@
4872 |markdownIfOption{StripPercentSigns}{@
4873 |if##1%@
4874 |expandafter|expandafter|expandafter@
4875 |markdownReadAndConvertProcessLine@
4876 |else@
4877 |expandafter|expandafter|expandafter@
4878 |markdownReadAndConvertProcessLine@
4879 |expandafter|expandafter|expandafter##1@
4880 |fi@
4881 }{@
4882 |expandafter@
4883 |markdownReadAndConvertProcessLine@
4884 |expandafter##1@
```

```
4885     }@
4886     }@
```

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

```
4887     |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@
```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```
4888         |ifx|relax##3|relax@
4889         |markdownIfOption{FrozenCache}-{-}@
4890         |immediate|write|markdownOutputFileStream{##1}@
4891         }@
4892     |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.

```
4893         |def^^M{@
4894         |markdownInfo{The ending token sequence was found}@
4895         |markdownIfOption{FrozenCache}-{-}@
4896         |immediate|closeout|markdownOutputFileStream@
4897         }@
4898         |endgroup@
4899         |markdownInput{@
4900         |markdownOptionOutputDir@
4901         /|markdownOptionInputTempFileName@
4902         }@
4903         #2}@
4904     |fi@
```

Repeat with the next line.

```
4905     ^^M}@
```

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

```
4906     |catcode`|^I=13@
4907     |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.



```

4908 |catcode`|^M=13@
4909 |def^^M##1^^M{@
4910 |def^^M###1^^M{@
4911 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
4912 ^^M}@
4913 ^^M}@

```

Reset the character categories back to the former state.

```
4914 |endgroup
```

### 3.2.6 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 3.2.7 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 [2, Section 3.1.1].

```

4915 \ifnum\markdownMode<2\relax
4916 \ifnum\markdownMode=0\relax
4917 \markdownInfo{Using mode 0: Shell escape via write18}%
4918 \else
4919 \markdownInfo{Using mode 1: Shell escape via os.execute}%
4920 \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.

```

4921 \ifx\pdfshellescape\undefined
4922 \ifx\shellescape\undefined
4923 \ifnum\markdownMode=0\relax
4924 \def\markdownExecuteShellEscape{1}%
4925 \else
4926 \def\markdownExecuteShellEscape{%
4927 \directlua{tex.sprint(status.shell_escape or "1")}}%
4928 \fi
4929 \else
4930 \let\markdownExecuteShellEscape\shellescape
4931 \fi
4932 \else
4933 \let\markdownExecuteShellEscape\pdfshellescape

```

4934 \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.

```
4935 \ifnum\markdownMode=0\relax
4936   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
4937 \else
4938   \def\markdownExecuteDirect#1{%
4939     \directlua{os.execute("\luaescapestring{#1}")}}%
4940 \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.

```
4941 \def\markdownExecute#1{%
4942   \ifnum\markdownExecuteShellEscape=1\relax
4943     \markdownExecuteDirect{#1}%
4944   \else
4945     \markdownError{I can not access the shell}{Either run the TeX
4946       compiler with the --shell-escape or the --enable-write18 flag,
4947       or set shell_escape=t in the texmf.cnf file}%
4948   \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.

4949 \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.

```
4950 \catcode`\|=0%
4951 \catcode`\|=12%
4952 |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.

```
4953   |immediate|openout|markdownOutputFileStream=%
4954   |markdownOptionHelperScriptFileName
4955 |markdownInfo{Writing a helper Lua script to the file
4956   "|markdownOptionHelperScriptFileName"}%
4957 |immediate|write|markdownOutputFileStream{%
4958   local ran_ok, error = pcall(function()
4959     local kpse = require("kpse")
4960     kpse.set_program_name("luatex")
4961     #1
4962   end)
```

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

```

4963     if not ran_ok then
4964         local file = io.open("%
4965             |markdownOptionOutputDir
4966             /|markdownOptionErrorTempFileName", "w")
4967         if file then
4968             file:write(error .. "\n")
4969             file:close()
4970         end
4971         print('\markdownError{An error was encountered while executing
4972             Lua code}{For further clues, examine the file
4973             "|markdownOptionOutputDir
4974             /|markdownOptionErrorTempFileName"}')
4975     end}%
4976 |immediate|closeout|markdownOutputFileStream

```

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

```

4977 |markdownInfo{Executing a helper Lua script from the file
4978     "|markdownOptionHelperScriptFileName" and storing the result in the
4979     file "|markdownOptionOutputTempFileName"}%
4980 |markdownExecute{texlua "|markdownOptionOutputDir
4981     /|markdownOptionHelperScriptFileName" > %
4982     "|markdownOptionOutputDir
4983     /|markdownOptionOutputTempFileName"}%

```

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

```

4984 |input|markdownOptionOutputTempFileName|relax}%
4985 |endgroup

```

### 3.2.7 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 3.2.6 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

4986 \else
4987 \markdownInfo{Using mode 2: Direct Lua access}%

```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `\tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.6,

```

4988 \def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%
4989 \fi

```

### 3.2.8 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```
4990 \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.

```
4991 \catcode`\|=0%
4992 \catcode`\|=12%
4993 \gdef\markdownInput#1{%
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache`(*number*) macro, and increment `\markdownFrozenCacheCounter`.

```
4994     \markdownIfOption{FrozenCache}{%
4995         \ifnum\markdownFrozenCacheCounter=0\relax
4996             \markdownInfo{Reading frozen cache from
4997                 "\markdownOptionFrozenCacheFileName"}%
4998             \input\markdownOptionFrozenCacheFileName\relax
4999         \fi
5000         \markdownInfo{Including markdown document number
5001             "\the\markdownFrozenCacheCounter" from frozen cache}%
5002         \csname markdownFrozenCache\the\markdownFrozenCacheCounter\endcsname
5003         \advance\markdownFrozenCacheCounter by 1\relax
5004     }{%
5005         \markdownInfo{Including markdown document "#1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```
5006         \openin\markdownInputFileStream#1
5007         \closein\markdownInputFileStream
5008         \markdownLuaExecute{%
5009             \markdownPrepare
5010             local file = assert(io.open("#1", "r"))
5011             local input = assert(file:read("*a"))
5012             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.

```
5013             print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%
```

If we are finalizing the frozen cache, increment `\markdownFrozenCacheCounter`.

```
5014         \markdownIfOption{FinalizeCache}{%
5015             \advance\markdownFrozenCacheCounter by 1\relax
5016         }%
5017     }%
```

```

5018 }%
5019 |endgroup

```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implementation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [8, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```

5020 \input markdown
5021 \def\markdownVersionSpace{ }%
5022 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
5023 \markdownVersion\markdownVersionSpace markdown renderer]%

```

#### 3.3.1 Logging Facilities

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

```

5024 \renewcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
5025 \renewcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
5026 \renewcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%

```

#### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.2).

```

5027 \let\markdownInputPlainTeX\markdownInput
5028 \renewcommand\markdownInput[2] []{%
5029   \begingroup
5030     \markdownSetup{#1}%
5031     \markdownInputPlainTeX{#2}%
5032   \endgroup}%

```

The `markdown`, and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

5033 \renewenvironment{markdown}{%
5034   \markdownReadAndConvert@markdown{}}{%
5035   \markdownEnd}%
5036 \renewenvironment{markdown*}[1]{%
5037   \markdownSetup{#1}%
5038   \markdownReadAndConvert@markdown*}{%
5039   \markdownEnd}%
5040 \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*.

```
5041 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
5042 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
5043 |gdef|markdownReadAndConvert@markdown#1<%
5044     |markdownReadAndConvert<\end{markdown#1}>%
5045         <|end<markdown#1>>>%
5046 |endgroup
```

### 3.3.3 Options

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

```
5047 \DeclareOption*{%
5048     \expandafter\markdownSetup\expandafter{\CurrentOption}}%
5049 \ProcessOptions\relax
```

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

```
5050 \define@key{markdownOptions}{renderers}{%
5051     \setkeys{markdownRenderers}{#1}%
5052     \def\KV@prefix{KV@markdownOptions@}}%
5053 \define@key{markdownOptions}{rendererPrototypes}{%
5054     \setkeys{markdownRendererPrototypes}{#1}%
5055     \def\KV@prefix{KV@markdownOptions@}}%
```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

If the `\markdownOptionTightLists` macro expands to `false`, do not load the `paralist` package. This is necessary for L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> document classes that do not play nice with `paralist`, such as `beamer`. If the `\markdownOptionTightLists` is undefined and the `beamer` document class is in use, then do not load the `paralist` package either.

```
5056 \ifx\markdownOptionTightLists\undefined
5057     \@ifclassloaded{beamer}{}{
5058         \RequirePackage{paralist}}
5059 \else
5060     \ifthenelse{\equal{\markdownOptionTightLists}{false}}{){
5061         \RequirePackage{paralist}}
5062 \fi
```

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

```

5063 \@ifpackageloaded{paralist}{
5064   \markdownSetup{rendererPrototypes={
5065     ulBeginTight = {\begin{compactitem}},
5066     ulEndTight = {\end{compactitem}},
5067     olBeginTight = {\begin{compactenum}},
5068     olEndTight = {\end{compactenum}},
5069     dlBeginTight = {\begin{compactdesc}},
5070     dlEndTight = {\end{compactdesc}}}}
5071 }{
5072   \markdownSetup{rendererPrototypes={
5073     ulBeginTight = {\markdownRendererUlBegin},
5074     ulEndTight = {\markdownRendererUlEnd},
5075     olBeginTight = {\markdownRendererOlBegin},
5076     olEndTight = {\markdownRendererOlEnd},
5077     dlBeginTight = {\markdownRendererDlBegin},
5078     dlEndTight = {\markdownRendererDlEnd}}}}
5079 \markdownSetup{rendererPrototypes={
5080   lineBreak = {\},
5081   leftBrace = {\textbraceleft},
5082   rightBrace = {\textbraceright},
5083   dollarSign = {\textdollar},
5084   underscore = {\textunderscore},
5085   circumflex = {\textasciicircum},
5086   backslash = {\textbackslash},
5087   tilde = {\textasciitilde},
5088   pipe = {\textbar},
5089   codeSpan = {\texttt{#1}},
5090   contentBlock = {%
5091     \ifthenelse{\equal{#1}{csv}}{%
5092       \begin{table}%
5093         \begin{center}%
5094           \csvautotabular{#3}%
5095         \end{center}
5096       \ifx\empty#4\empty\else
5097         \caption{#4}%
5098       \fi
5099     \end{table}}{%
5100     \markdownInput{#3}}},
5101   image = {%
5102     \begin{figure}%
5103       \begin{center}%
5104         \includegraphics{#3}%
5105       \end{center}%
5106     \ifx\empty#4\empty\else
5107       \caption{#4}%
5108     \fi
5109     \label{fig:#1}%

```

```

5110     \end{figure}},
5111     ulBegin = {\begin{itemize}},
5112     ulItem = {\item},
5113     ulEnd = {\end{itemize}},
5114     olBegin = {\begin{enumerate}},
5115     olItem = {\item},
5116     olItemWithNumber = {\item[#1.]},
5117     olEnd = {\end{enumerate}},
5118     dlBegin = {\begin{description}},
5119     dlItem = {\item[#1]},
5120     dlEnd = {\end{description}},
5121     emphasis = {\emph{#1}},
5122     blockQuoteBegin = {\begin{quotation}},
5123     blockQuoteEnd = {\end{quotation}},
5124     inputVerbatim = {\VerbatimInput{#1}},
5125     inputFencedCode = {%
5126         \ifx\relax#2\relax
5127         \VerbatimInput{#1}%
5128     \else
5129         \ifx\minted@code\undefined
5130         \ifx\lst@version\undefined
5131         \markdownRendererInputFencedCode{#1}{}%

```

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

```

5132         \else
5133         \lstinputlisting[language=#2]{#1}%
5134     \fi

```

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

```

5135         \else
5136         \inputminted{#2}{#1}%
5137     \fi
5138 \fi},
5139 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
5140 footnote = {\footnote{#1}}}}

```

Support the nesting of strong emphasis.

```

5141 \newif\ifmarkdownLATEXStrongEmphasisNested
5142 \markdownLATEXStrongEmphasisNestedfalse
5143 \markdownSetup{rendererPrototypes={
5144     strongEmphasis = {%
5145         \ifmarkdownLATEXStrongEmphasisNested
5146         \markdownLATEXStrongEmphasisNestedfalse
5147         \textmd{#1}%
5148         \markdownLATEXStrongEmphasisNestedtrue
5149     \else
5150         \markdownLATEXStrongEmphasisNestedtrue

```



```

5151     \textbf{#1}%
5152     \markdownLATEXStrongEmphasisNestedfalse
5153 \fi}}

```

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

```

5154 \ifx\chapter\undefined
5155   \markdownSetup{rendererPrototypes = {
5156     headingOne = {\section{#1}},
5157     headingTwo = {\subsection{#1}},
5158     headingThree = {\subsubsection{#1}},
5159     headingFour = {\paragraph{#1}\leavevmode},
5160     headingFive = {\subparagraph{#1}\leavevmode}}}
5161 \else
5162   \markdownSetup{rendererPrototypes = {
5163     headingOne = {\chapter{#1}},
5164     headingTwo = {\section{#1}},
5165     headingThree = {\subsection{#1}},
5166     headingFour = {\subsubsection{#1}},
5167     headingFive = {\paragraph{#1}\leavevmode},
5168     headingSix = {\subparagraph{#1}\leavevmode}}}
5169 \fi

```

There is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

5170 \newcount\markdownLaTeXCitationsCounter
5171
5172 % Basic implementation
5173 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
5174   \advance\markdownLaTeXCitationsCounter by 1\relax
5175   \ifx\relax#4\relax
5176     \ifx\relax#5\relax
5177       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5178         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
5179         \expandafter\expandafter\expandafter
5180         \expandafter\expandafter\expandafter\expandafter
5181         \@gobblethree
5182       \fi
5183     \else% Before a postnote (#5), dump the accumulator
5184       \ifx\relax#1\relax\else
5185         \cite{#1}%
5186       \fi
5187       \cite[#5]{#6}%
5188     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5189     \else
5190       \expandafter\expandafter\expandafter
5191       \expandafter\expandafter\expandafter\expandafter

```

```

5192     \expandafter\expandafter\expandafter
5193     \expandafter\expandafter\expandafter\expandafter
5194     \markdownLaTeXBasicCitations
5195     \fi
5196     \expandafter\expandafter\expandafter
5197     \expandafter\expandafter\expandafter\expandafter{%
5198     \expandafter\expandafter\expandafter
5199     \expandafter\expandafter\expandafter\expandafter}%
5200     \expandafter\expandafter\expandafter
5201     \expandafter\expandafter\expandafter\expandafter{%
5202     \expandafter\expandafter\expandafter
5203     \expandafter\expandafter\expandafter\expandafter}%
5204     \expandafter\expandafter\expandafter
5205     \@gobblethree
5206     \fi
5207 \else% Before a prenote (#4), dump the accumulator
5208     \ifx\relax#1\relax\else
5209         \cite{#1}%
5210     \fi
5211     \ifnum\markdownLaTeXCitationsCounter>1\relax
5212         \space % Insert a space before the prenote in later citations
5213     \fi
5214     #4-\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
5215     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5216     \else
5217         \expandafter\expandafter\expandafter
5218         \expandafter\expandafter\expandafter\expandafter
5219         \markdownLaTeXBasicCitations
5220     \fi
5221     \expandafter\expandafter\expandafter{%
5222     \expandafter\expandafter\expandafter}%
5223     \expandafter\expandafter\expandafter{%
5224     \expandafter\expandafter\expandafter}%
5225     \expandafter
5226     \@gobblethree
5227     \fi\markdownLaTeXBasicCitations{#1#2#6},}
5228 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
5229
5230 % Natbib implementation
5231 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
5232     \advance\markdownLaTeXCitationsCounter by 1\relax
5233     \ifx\relax#3\relax
5234         \ifx\relax#4\relax
5235             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5236                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
5237             \expandafter\expandafter\expandafter
5238             \expandafter\expandafter\expandafter\expandafter

```

```

5239     \@gobbletwo
5240     \fi
5241 \else% Before a postnote (#4), dump the accumulator
5242     \ifx\relax#1\relax\else
5243         \citep{#1}%
5244     \fi
5245     \citep[] [#4]{#5}%
5246     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5247     \else
5248         \expandafter\expandafter\expandafter
5249         \expandafter\expandafter\expandafter\expandafter
5250         \expandafter\expandafter\expandafter
5251         \expandafter\expandafter\expandafter\expandafter
5252         \markdownLaTeXNatbibCitations
5253     \fi
5254     \expandafter\expandafter\expandafter
5255     \expandafter\expandafter\expandafter\expandafter{%
5256     \expandafter\expandafter\expandafter
5257     \expandafter\expandafter\expandafter\expandafter}%
5258     \expandafter\expandafter\expandafter
5259     \@gobbletwo
5260 \fi
5261 \else% Before a prenote (#3), dump the accumulator
5262     \ifx\relax#1\relax\relax\else
5263         \citep{#1}%
5264     \fi
5265     \citep[#3] [#4]{#5}%
5266     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5267     \else
5268         \expandafter\expandafter\expandafter
5269         \expandafter\expandafter\expandafter\expandafter
5270         \markdownLaTeXNatbibCitations
5271     \fi
5272     \expandafter\expandafter\expandafter{%
5273     \expandafter\expandafter\expandafter}%
5274     \expandafter
5275     \@gobbletwo
5276 \fi\markdownLaTeXNatbibCitations{#1,#5}}
5277 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
5278 \advance\markdownLaTeXCitationsCounter by 1\relax
5279 \ifx\relax#3\relax
5280     \ifx\relax#4\relax
5281         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5282             \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
5283         \expandafter\expandafter\expandafter
5284         \expandafter\expandafter\expandafter\expandafter
5285         \@gobbletwo

```

```

5286     \fi
5287 \else% After a prenote or a postnote, dump the accumulator
5288     \ifx\relax#1\relax\else
5289         \citet{#1}%
5290     \fi
5291     , \citet[#3][#4]{#5}%
5292     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
5293     ,
5294     \else
5295         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
5296         ,
5297         \fi
5298     \fi
5299     \expandafter\expandafter\expandafter
5300     \expandafter\expandafter\expandafter\expandafter
5301     \markdownLaTeXNatbibTextCitations
5302     \expandafter\expandafter\expandafter
5303     \expandafter\expandafter\expandafter\expandafter{%
5304     \expandafter\expandafter\expandafter
5305     \expandafter\expandafter\expandafter\expandafter}%
5306     \expandafter\expandafter\expandafter
5307     \@gobbletwo
5308     \fi
5309 \else% After a prenote or a postnote, dump the accumulator
5310     \ifx\relax#1\relax\relax\else
5311         \citet{#1}%
5312     \fi
5313     , \citet[#3][#4]{#5}%
5314     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
5315     ,
5316     \else
5317         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
5318         ,
5319         \fi
5320     \fi
5321     \expandafter\expandafter\expandafter
5322     \markdownLaTeXNatbibTextCitations
5323     \expandafter\expandafter\expandafter{%
5324     \expandafter\expandafter\expandafter}%
5325     \expandafter
5326     \@gobbletwo
5327     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
5328
5329 % BibLaTeX implementation
5330 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
5331     \advance\markdownLaTeXCitationsCounter by 1\relax
5332     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```

```

5333     \autocites#1[#3][#4]{#5}%
5334     \expandafter\@gobbletwo
5335     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
5336 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
5337     \advance\markdownLaTeXCitationsCounter by 1\relax
5338     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
5339     \textcites#1[#3][#4]{#5}%
5340     \expandafter\@gobbletwo
5341     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
5342
5343 \markdownSetup{rendererPrototypes = {
5344     cite = {%
5345         \markdownLaTeXCitationsCounter=1%
5346         \def\markdownLaTeXCitationsTotal{#1}%
5347         \ifx\autocites\undefined
5348             \ifx\citep\undefined
5349                 \expandafter\expandafter\expandafter
5350                 \markdownLaTeXBasicCitations
5351                 \expandafter\expandafter\expandafter{%
5352                 \expandafter\expandafter\expandafter}%
5353                 \expandafter\expandafter\expandafter{%
5354                 \expandafter\expandafter\expandafter}%
5355             \else
5356                 \expandafter\expandafter\expandafter
5357                 \markdownLaTeXNatbibCitations
5358                 \expandafter\expandafter\expandafter{%
5359                 \expandafter\expandafter\expandafter}%
5360             \fi
5361         \else
5362             \expandafter\expandafter\expandafter
5363             \markdownLaTeXBibLaTeXCitations
5364             \expandafter{\expandafter}%
5365         \fi},
5366     textCite = {%
5367         \markdownLaTeXCitationsCounter=1%
5368         \def\markdownLaTeXCitationsTotal{#1}%
5369         \ifx\autocites\undefined
5370             \ifx\citep\undefined
5371                 \expandafter\expandafter\expandafter
5372                 \markdownLaTeXBasicTextCitations
5373                 \expandafter\expandafter\expandafter{%
5374                 \expandafter\expandafter\expandafter}%
5375                 \expandafter\expandafter\expandafter{%
5376                 \expandafter\expandafter\expandafter}%
5377             \else
5378                 \expandafter\expandafter\expandafter
5379                 \markdownLaTeXNatbibTextCitations

```

```

5380     \expandafter\expandafter\expandafter{%
5381     \expandafter\expandafter\expandafter}%
5382     \fi
5383   \else
5384     \expandafter\expandafter\expandafter
5385     \markdownLaTeXBibLaTeXTextCitations
5386     \expandafter{\expandafter}%
5387   \fi}}

```

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. After the hyperlink has been typeset, we restore the original catcode.

```

5388 \def\markdownRendererLinkPrototype{%
5389   \begingroup
5390   \catcode`\#=12
5391   \def\next##1##2##3##4{%
5392     #1\footnote{%
5393       \ifx\empty##4\empty\else##4: \fi\texttt<\url{##3}\texttt>}%
5394   \endgroup}%
5395 \next}

```

There is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

5396 \newcount\markdownLaTeXRowCounter
5397 \newcount\markdownLaTeXRowTotal
5398 \newcount\markdownLaTeXColumnCounter
5399 \newcount\markdownLaTeXColumnTotal
5400 \newtoks\markdownLaTeXTable
5401 \newtoks\markdownLaTeXTableAlignment
5402 \newtoks\markdownLaTeXTableEnd
5403 \@ifpackageloaded{booktabs}{
5404   \let\markdownLaTeXTopRule\toprule
5405   \let\markdownLaTeXMidRule\midrule
5406   \let\markdownLaTeXBottomRule\bottomrule
5407 }{
5408   \let\markdownLaTeXTopRule\hline
5409   \let\markdownLaTeXMidRule\hline
5410   \let\markdownLaTeXBottomRule\hline
5411 }
5412 \markdownSetup{rendererPrototypes={
5413   table = {%
5414     \markdownLaTeXTable={}%
5415     \markdownLaTeXTableAlignment={}%
5416     \markdownLaTeXTableEnd={%
5417       \markdownLaTeXBottomRule
5418     \end{tabular}}}%
5419   \ifx\empty#1\empty\else
5420     \addto@hook\markdownLaTeXTable{%

```

```

5421     \begin{table}
5422     \centering}%
5423     \addto@hook\markdownLaTeXTableEnd{%
5424     \caption{#1}
5425     \end{table}}}%
5426     \fi
5427     \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
5428     \markdownLaTeXRowCounter=0%
5429     \markdownLaTeXRowTotal=#2%
5430     \markdownLaTeXColumnTotal=#3%
5431     \markdownLaTeXRenderTableRow
5432   }
5433 }}
5434 \def\markdownLaTeXRenderTableRow#1{%
5435   \markdownLaTeXColumnCounter=0%
5436   \ifnum\markdownLaTeXRowCounter=0\relax
5437     \markdownLaTeXReadAlignments#1%
5438     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
5439       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
5440         \the\markdownLaTeXTableAlignment}}}%
5441     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
5442   \else
5443     \markdownLaTeXRenderTableCell#1%
5444   \fi
5445   \ifnum\markdownLaTeXRowCounter=1\relax
5446     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
5447   \fi
5448   \advance\markdownLaTeXRowCounter by 1\relax
5449   \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
5450     \markdownInfo{\the\markdownLaTeXTable}
5451     \markdownInfo{\the\markdownLaTeXTableEnd}
5452     \the\markdownLaTeXTable
5453     \the\markdownLaTeXTableEnd
5454     \expandafter\@gobble
5455   \fi\markdownLaTeXRenderTableRow}
5456 \def\markdownLaTeXReadAlignments#1{%
5457   \advance\markdownLaTeXColumnCounter by 1\relax
5458   \if#1d%
5459     \addto@hook\markdownLaTeXTableAlignment{1}%
5460   \else
5461     \addto@hook\markdownLaTeXTableAlignment{#1}%
5462   \fi
5463   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
5464     \expandafter\@gobble
5465   \fi\markdownLaTeXReadAlignments}
5466 \def\markdownLaTeXRenderTableCell#1{%
5467   \advance\markdownLaTeXColumnCounter by 1\relax

```

```

5468 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
5469   \addto@hook\markdownLaTeXTable{#1&}%
5470 \else
5471   \addto@hook\markdownLaTeXTable{#1\\}%
5472   \expandafter\@gobble
5473 \fi\markdownLaTeXRenderTableCell}

```

### 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.

```

5474 \newcommand\markdownMakeOther{%
5475   \count0=128\relax
5476   \loop
5477     \catcode\count0=11\relax
5478     \advance\count0 by 1\relax
5479   \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.

```

5480 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
5481   \do\#\do\^\do\_ \do\% \do\~}%
5482 \input markdown

```

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L<sup>A</sup>T<sub>E</sub>X package.

```

5483 \def\markdownMakeOther{%
5484   \count0=128\relax
5485   \loop
5486     \catcode\count0=11\relax
5487     \advance\count0 by 1\relax
5488   \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.

```

5489   \catcode`|=12}%

```



### 3.4.1 Logging Facilities

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

```
5490 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
5491 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
```

### 3.4.2 Typesetting Markdown

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

```
5492 \begingroup
```

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

```
5493 \catcode`\|=0%
5494 \catcode`\|=12%
5495 |gdef|startmarkdown{%
5496     |markdownReadAndConvert{\stopmarkdown}%
5497     {|stopmarkdown}}%
5498 |gdef|stopmarkdown{|markdownEnd}%
5499 |endgroup
```

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
5500 \def\markdownRendererLineBreakPrototype{\blank}%
5501 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
5502 \def\markdownRendererRightBracePrototype{\textbraceright}%
5503 \def\markdownRendererDollarSignPrototype{\textdollar}%
5504 \def\markdownRendererPercentSignPrototype{\percent}%
5505 \def\markdownRendererUnderscorePrototype{\textunderscore}%
5506 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
5507 \def\markdownRendererBackslashPrototype{\textbackslash}%
5508 \def\markdownRendererTildePrototype{\textasciitilde}%
5509 \def\markdownRendererPipePrototype{\char`|}%
5510 \def\markdownRendererLinkPrototype#1#2#3#4{%
5511     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
5512     \fi\texttt<\hyphenatedurl{#3}>}}%
5513 \usemodule[database]
5514 \defineseparatedlist
5515     [MarkdownConTeXtCSV
5516     [separator={,},
5517     before=\bTABLE,after=\eTABLE,
5518     first=\bTR,last=\eTR,
```

```

5519     left=\bTD,right=\eTD]
5520 \def\markdownConTeXtCSV{csv}
5521 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
5522     \def\markdownConTeXtCSV@arg{#1}%
5523     \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
5524         \placetable[] [tab:#1]{#4}{%
5525             \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
5526     \else
5527     \markdownInput{#3}%
5528     \fi}%
5529 \def\markdownRendererImagePrototype#1#2#3#4{%
5530     \placefigure[] [fig:#1]{#4}{\externalfigure [#3]}%
5531 \def\markdownRendererUlBeginPrototype{\startitemize}%
5532 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
5533 \def\markdownRendererUlItemPrototype{\item}%
5534 \def\markdownRendererUlEndPrototype{\stopitemize}%
5535 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
5536 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
5537 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
5538 \def\markdownRendererOlItemPrototype{\item}%
5539 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
5540 \def\markdownRendererOlEndPrototype{\stopitemize}%
5541 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
5542 \definedescription
5543     [MarkdownConTeXtDlItemPrototype]
5544     [location=hanging,
5545     margin=standard,
5546     headstyle=bold]%
5547 \definestartstop
5548     [MarkdownConTeXtDlPrototype]
5549     [before=\blank,
5550     after=\blank]%
5551 \definestartstop
5552     [MarkdownConTeXtDlTightPrototype]
5553     [before=\blank\startpacked,
5554     after=\stoppacked\blank]%
5555 \def\markdownRendererDlBeginPrototype{%
5556     \startMarkdownConTeXtDlPrototype}%
5557 \def\markdownRendererDlBeginTightPrototype{%
5558     \startMarkdownConTeXtDlTightPrototype}%
5559 \def\markdownRendererDlItemPrototype#1{%
5560     \startMarkdownConTeXtDlItemPrototype{#1}}%
5561 \def\markdownRendererDlItemEndPrototype{%
5562     \stopMarkdownConTeXtDlItemPrototype}%
5563 \def\markdownRendererDlEndPrototype{%
5564     \stopMarkdownConTeXtDlPrototype}%
5565 \def\markdownRendererDlEndTightPrototype{%

```

```

5566 \stopMarkdownConTeXtDlTightPrototype}%
5567 \def\markdownRendererEmphasisPrototype#1{\em#1}%
5568 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
5569 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
5570 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
5571 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
5572 \def\markdownRendererInputFencedCodePrototype#1#2{%
5573   \ifx\relax#2\relax
5574     \typefile{#1}%
5575   \else

```

The code fence infostring is used as a name from the ConT<sub>E</sub>Xt `\definetyping` macro. This allows the user to set up code highlighting mapping as follows:

```

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

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

```

```

5576   \typefile[#2] []{#1}%
5577   \fi}%
5578 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
5579 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
5580 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
5581 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
5582 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
5583 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
5584 \def\markdownRendererHorizontalRulePrototype{%
5585   \blackrule[height=1pt, width=\hsize]}%
5586 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
5587 \stopmodule\protect

```

There is a basic implementation of tables.

```

5588 \newcount\markdownConTeXtRowCounter
5589 \newcount\markdownConTeXtRowTotal

```

```

5590 \newcount\markdownConTeXtColumnCounter
5591 \newcount\markdownConTeXtColumnTotal
5592 \newtoks\markdownConTeXtTable
5593 \newtoks\markdownConTeXtTableFloat
5594 \def\markdownRendererTablePrototype#1#2#3{%
5595   \markdownConTeXtTable={}%
5596   \ifx\empty#1\empty
5597     \markdownConTeXtTableFloat={%
5598       \the\markdownConTeXtTable}%
5599   \else
5600     \markdownConTeXtTableFloat={%
5601       \placetable{#1}{\the\markdownConTeXtTable}}%
5602   \fi
5603   \begingroup
5604   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
5605   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
5606   \setupTABLE[r][1][topframe=on, bottomframe=on]
5607   \setupTABLE[r][#1][bottomframe=on]
5608   \markdownConTeXtRowCounter=0%
5609   \markdownConTeXtRowTotal=#2%
5610   \markdownConTeXtColumnTotal=#3%
5611   \markdownConTeXtRenderTableRow}
5612 \def\markdownConTeXtRenderTableRow#1{%
5613   \markdownConTeXtColumnCounter=0%
5614   \ifnum\markdownConTeXtRowCounter=0\relax
5615     \markdownConTeXtReadAlignments#1%
5616     \markdownConTeXtTable={\bTABLE}%
5617   \else
5618     \markdownConTeXtTable=\expandafter{%
5619       \the\markdownConTeXtTable\bTR}%
5620     \markdownConTeXtRenderTableCell#1%
5621     \markdownConTeXtTable=\expandafter{%
5622       \the\markdownConTeXtTable\eTR}%
5623   \fi
5624   \advance\markdownConTeXtRowCounter by 1\relax
5625   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
5626     \markdownConTeXtTable=\expandafter{%
5627       \the\markdownConTeXtTable\eTABLE}%
5628     \the\markdownConTeXtTableFloat
5629   \endgroup
5630   \expandafter\gobbleoneargument
5631   \fi\markdownConTeXtRenderTableRow}
5632 \def\markdownConTeXtReadAlignments#1{%
5633   \advance\markdownConTeXtColumnCounter by 1\relax
5634   \if#1d%
5635     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
5636   \fi\if#1l%

```

```

5637 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
5638 \fi\if#1c%
5639 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
5640 \fi\if#1r%
5641 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
5642 \fi
5643 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
5644 \expandafter\gobbleoneargument
5645 \fi\markdownConTeXtReadAlignments}
5646 \def\markdownConTeXtRenderTableCell#1{%
5647 \advance\markdownConTeXtColumnCounter by 1\relax
5648 \markdownConTeXtTable=\expandafter{%
5649 \the\markdownConTeXtTable\bTD#1\eTD}%
5650 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
5651 \expandafter\gobbleoneargument
5652 \fi\markdownConTeXtRenderTableCell}

```

## References

- [1] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [2] LuaTeX development team. *LuaTeX reference manual*. Feb. 2017. URL: <http://www.luatex.org/svn/trunk/manual/luatex.pdf> (visited on 01/08/2018).
- [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 TeXbook*. 3rd ed. Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [5] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <http://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [6] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <http://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [7] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [8] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <http://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).