A Markdown Interpreter for TeX

Vít Starý Novotný, Andrej Genčur
witiko@mail.muni.cz

Version 3.10.0-0-g626df6ad
2025-01-27

Contents

1 Introduction **1**
1.1 Requirements 2
1.2 Feedback 6
1.3 Acknowledgements 7

2 Interfaces **7**
2.1 Lua Interface 7
2.2 Plain TeX Interface 53
2.3 LaTeX Interface 152
2.4 ConTeXt Interface 161

3 Implementation **166**
3.1 Lua Implementation . . . 167
3.2 Plain TeX Implementation 380
3.3 LaTeX Implementation . . 421
3.4 ConTeXt Implementation 462

References **473**

Index **475**

List of Figures

1 A block diagram of the Markdown package 8
2 A sequence diagram of typesetting a document using the TeX interface . . 49
3 A sequence diagram of typesetting a document using the Lua CLI 50
4 An example directed graph 75
5 An example mindmap 76
6 An example UML sequence diagram 77
7 The banner of the Markdown package 78
8 A pushdown automaton that recognizes TeX comments 258

1 Introduction

The Markdown package[1] converts CommonMark[2] markup to TeX commands. The functionality is provided both as a Lua module and as plain TeX, LaTeX, and ConTeXt macro packages that can be used to directly typeset TeX 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.

[1] See https://ctan.org/pkg/markdown.
[2] See https://commonmark.org/.

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.[3]

```
local metadata = {
    version   = "(((VERSION)))",
    comment   = "A module for the conversion from markdown "
             .. "to plain TeX",
    author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný, "
             .. "Andrej Genčur",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2024 Vít Starý Novotný, Andrej Genčur"},
    license   = "LPPL 1.3c"
}

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

1.1 Requirements

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

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the LuaTEX engine (though not necessarily in the LuaMetaTEX engine).

LPeg $\geqslant$ 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 $\geqslant$ 0.10 is included in LuaTEX $\geqslant$ 0.72.0 (TEX Live $\geqslant$ 2013).

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

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

```
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 (TEX Live $\geqslant$ 2008).

[3] See http://mirrors.ctan.org/macros/generic/markdown/markdown.html.

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

Kpathsea A package that implements the loading of third-party Lua libraries and looking up files in the TEX directory structure.

```
;(function()
```

If Kpathsea has not been loaded before or if LuaTEX has not yet been initialized, configure Kpathsea on top of loading it. Since ConTEXt MkIV provides a `kpse` global that acts as a stub for Kpathsea and the lua-uni-case library expects that `kpse` is a reference to the full Kpathsea library, we load Kpathsea to the `kpse` global.

```
  local should_initialize = package.loaded.kpse == nil
                         or tex.initialize ~= nil
  kpse = require("kpse")
  if should_initialize then
    kpse.set_program_name("luatex")
  end
end)()
```

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

lua-uni-algos A package that implements Unicode case-folding in TEX Live ⩾ 2020.

```
hard lua-uni-algos
```

```
local uni_algos = require("lua-uni-algos")
```

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

```
hard lua-tinyyaml
```

1.1.2 Plain TEX Requirements

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

expl3 A package that enables the expl3 language [2] from the LATEX3 kernel in TEX Live ⩽ 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
hard l3kernel
```

```
\unprotect
\expandafter\ifx\csname ExplSyntaxOn\endcsname\relax
  \input expl3-generic
\fi
```

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

```
hard lt3luabridge
```

The plain TEX part of the package also requires the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TEX code to create the cache directory specified by the `cacheDir` option before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTEX (TEXLive ⩾ 2008).

The plain TEX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTEX engine developers [1, Section 4.2.4].

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

Unless you convert markdown documents to TEX manually using the Lua command-line interface (see Section 2.1.7), 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), then unless your TEX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 LaTeX Requirements

The LaTeX part of the package requires that the LaTeX 2_ε format is loaded, a TEX engine that extends ε-TEX, and all the plain TEX prerequisites (see Section 1.1.2).

```
\NeedsTeXFormat{LaTeX2e}
\RequirePackage{expl3}
```

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or LaTeX themes (see Section 2.3.4) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
soft url
```

graphicx A package that provides the `\includegraphics` macro for the typesetting of images. Furthermore, it also provides a key–value interface that is used in the default renderer prototypes for image attribute contexts.

```
soft graphics
```

enumitem and paralist Packages that provide macros for the default renderer prototypes for tight and fancy lists.

The package paralist will be used unless the option `experimental` has been enabled, in which case, the package enumitem will be used. Furthermore, enabling any test phase [3] will also cause enumitem to be used. In a future major version, enumitem will replace paralist altogether.

```
soft enumitem
soft paralist
```

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

```
soft fancyvrb
```

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

```
soft csvsimple
soft pgf  # required by `csvsimple`, which loads `pgfkeys.sty`
soft tools  # required by `csvsimple`, which loads `shellesc.sty`
soft etoolbox  # required by `csvsimple`, which loads `etoolbox.sty`
```

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

```
soft amsmath
soft amsfonts
```

graphicx A package that provides extended support for graphics. It is used in the `witiko/diagrams`, and `witiko/graphicx/http` plain TEX themes, see Section 2.2.3.

```
soft graphics
soft epstopdf  # required by `graphics` and `graphicx`, which load `epsopdf-
  base.sty`
soft epstopdf-pkg  # required by `graphics` and `graphicx`, which load `epsopdf-
  base.sty`
```

soul and xcolor Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdfTeX.

```
soft soul
soft xcolor
```

lua-ul and luacolor Packages that are used in the default renderer prototypes for strike-throughs and marked text in LuaTeX.

```
soft lua-ul
soft luacolor
```

ltxcmds A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.

```
soft ltxcmds
```

luaxml A package that is used to convert HTML to LaTeX in the default renderer prototypes for content blocks, raw blocks, and inline raw spans.

```
soft luaxml
```

verse A package that is used in the default renderer prototypes for line blocks.

```
soft verse
```

1.1.4 ConTeXt Prerequisites

The ConTeXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain TeX prerequisites (see Section 1.1.2), and the following ConTeXt modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the CSV filename extension (see Section 2.2.6).

1.2 Feedback

Please use the Markdown project page on GitHub[4] to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the TeX-LaTeX Stack Exchange.[5] community question answering web site under the `markdown` tag.

[4] See https://github.com/witiko/markdown/issues.
[5] See https://tex.stackexchange.com.

1.3 Acknowledgements

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

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

Funding by the Faculty of Informatics at the Masaryk University in Brno [4] is gratefully acknowledged.

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

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

2 Interfaces

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

Since neither TEX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is exposed by the Lua layer. The plain TEX layer exposes the conversion capabilities of Lua as TEX macros. The LATEX and ConTEXt layers provide syntactic sugar on top of plain TEX macros. The user can interface with any and all layers.

2.1 Lua Interface

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

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

```
57 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain TEX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain TEX according to the table `options`

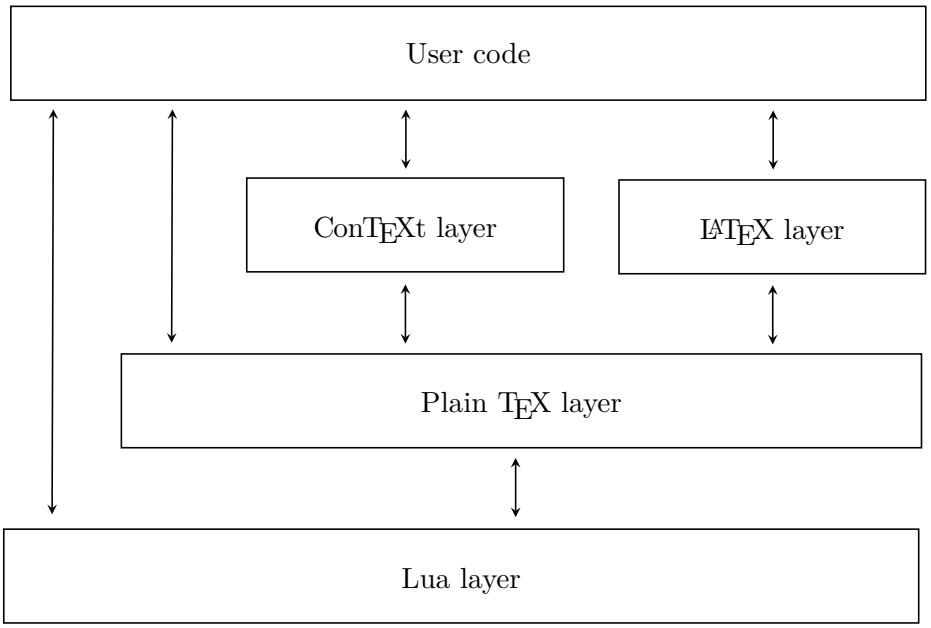

Figure 1: A block diagram of the Markdown package

that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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

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

2.1.2 User-Defined Syntax Extensions

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

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

```
58 local walkable_syntax = {
```

```
  Block = {
    "Blockquote",
    "Verbatim",
    "ThematicBreak",
    "BulletList",
    "OrderedList",
    "DisplayHtml",
    "Heading",
  },
  BlockOrParagraph = {
    "Block",
    "Paragraph",
    "Plain",
  },
  Inline = {
    "Str",
    "Space",
    "Endline",
    "EndlineBreak",
    "LinkAndEmph",
    "Code",
    "AutoLinkUrl",
    "AutoLinkEmail",
    "AutoLinkRelativeReference",
    "InlineHtml",
    "HtmlEntity",
    "EscapedChar",
    "Smart",
    "Symbol",
  },
}
```

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

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

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

2.1.3 Options

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

```
local defaultOptions = {}
```

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

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

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

```
\prop_new:N \g_@@_lua_option_types_prop
\prop_new:N \g_@@_default_lua_options_prop
\seq_new:N \g_@@_option_layers_seq
\tl_const:Nn \c_@@_option_layer_lua_tl { lua }
\seq_gput_right:NV
  \g_@@_option_layers_seq
  \c_@@_option_layer_lua_tl
\cs_new:Nn
  \@@_add_lua_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_lua_tl
      { #1 }
      { #2 }
      { #3 }
  }
\cs_new:Nn
  \@@_add_option:nnnn
  {
    \seq_gput_right:cn
      { g_@@_ #1 _options_seq }
      { #2 }
    \prop_gput:cnn
      { g_@@_ #1 _option_types_prop }
      { #2 }
      { #3 }
    \prop_gput:cnn
      { g_@@_default_ #1 _options_prop }
      { #2 }
      { #4 }
    \@@_typecheck_option:n
      { #2 }
  }
```

```
\cs_generate_variant:Nn
  \@@_add_option:nnnn
  { Vnnn }
\tl_const:Nn \c_@@_option_value_true_tl  { true  }
\tl_const:Nn \c_@@_option_value_false_tl { false }
\cs_new:Nn \@@_typecheck_option:n
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_case_e:Vn
      \l_tmpa_tl
      {
        { \c_@@_option_type_boolean_tl }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \bool_if:nF
              {
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_true_tl ||
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_false_tl
              }
              {
                \msg_error:nnnV
                  { markdown }
                  { failed-typecheck-for-boolean-option }
                  { #1 }
                  \l_tmpa_tl
              }
          }
      }
  }
\msg_new:nnn
  { markdown }
  { failed-typecheck-for-boolean-option }
  {
    Option~#1~has~value~#2,~
    but~a~boolean~(true~or~false)~was~expected.
  }
\cs_generate_variant:Nn
  \str_case_e:nn
  { Vn }
```

```
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnnV }
\seq_new:N
  \g_@@_option_types_seq
\tl_const:Nn
  \c_@@_option_type_clist_tl
  { clist }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_clist_tl
\tl_const:Nn
  \c_@@_option_type_counter_tl
  { counter }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_counter_tl
\tl_const:Nn
  \c_@@_option_type_boolean_tl
  { boolean }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_boolean_tl
\tl_const:Nn
  \c_@@_option_type_number_tl
  { number }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_number_tl
\tl_const:Nn
  \c_@@_option_type_path_tl
  { path }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_path_tl
\tl_const:Nn
  \c_@@_option_type_slice_tl
  { slice }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_slice_tl
\tl_const:Nn
  \c_@@_option_type_string_tl
  { string }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_string_tl
```

```
\cs_new:Nn
  \@@_get_option_type:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_ ##1 _option_types_prop }
          { #1 }
          \l_tmpa_tl
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { markdown }
          { undefined-option }
          { #1 }
      }
    \seq_if_in:NVF
      \g_@@_option_types_seq
      \l_tmpa_tl
      {
        \msg_error:nnnV
          { markdown }
          { unknown-option-type }
          { #1 }
          \l_tmpa_tl
      }
    \tl_set_eq:NN
      #2
      \l_tmpa_tl
  }
\msg_new:nnn
  { markdown }
  { unknown-option-type }
  {
    Option~#1~has~unknown~type~#2.
  }
\msg_new:nnn
```

```
  { markdown }
  { undefined-option }
  {
    Option~#1~is~undefined.
  }
\cs_new:Nn
  \@@_get_default_option_value:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_default_ ##1 _options_prop }
          { #1 }
          #2
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { markdown }
          { undefined-option }
          { #1 }
      }
  }
\cs_new:Nn
  \@@_get_option_value:nN
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cTF
      { \l_tmpa_tl }
      {
        \@@_get_default_option_value:nN
          { #1 }
          #2
      }
      {
        \@@_get_option_type:nN
```

```
          { #1 }
          \l_tmpa_tl
        \str_if_eq:NNTF
          \c_@@_option_type_counter_tl
          \l_tmpa_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nx
              #2
              { \the \cs:w \l_tmpa_tl \cs_end: }  % noqa: W200
          }
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nv
              #2
              { \l_tmpa_tl }
          }
      }
  }
\cs_new:Nn \@@_option_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownOption
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
```

To make it easier to support different coding styles in the interface, engines, we define the `\@@_with_various_cases:nn` function that allows us to generate different variants of a string using different cases.

```
\cs_new:Nn \@@_with_various_cases:nn
  {
    \seq_clear:N
      \l_tmpa_seq
    \seq_map_inline:Nn
      \g_@@_cases_seq
      {
```

```
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \use:c { ##1 }
          \l_tmpa_tl
        \seq_put_right:NV
          \l_tmpa_seq
          \l_tmpa_tl
      }
    \seq_map_inline:Nn
      \l_tmpa_seq
      { #2 }
  }
```

To interrupt the `\@@_with_various_cases:nn` function prematurely, use the `\@@_with_various_cases_break:` function.

```
\cs_new:Nn \@@_with_various_cases_break:
  {
    \seq_map_break:
  }
```

By default, camelCase and snake_case are supported. Additional cases can be added by adding functions to the `\g_@@_cases_seq` sequence.

```
\seq_new:N \g_@@_cases_seq
\cs_new:Nn \@@_camel_case:N
  {
    \regex_replace_all:nnN
      { _ ([a-z]) }
      { \c { str_uppercase:n } \cB\{ \1 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
\cs_new:Nn \@@_snake_case:N
  {
    \regex_replace_all:nnN
      { ([a-z])([A-Z]) }
      { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }
```

2.1.4 General Behavior

`eagerCache`=`true`, `false` default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. Furthermore, it can also significantly improve the processing speed for documents that require multiple compilation runs, since each markdown document is only converted once. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing. However, it makes it impossible to post-process the converted documents and recover historical versions of the documents from the cache. Furthermore, it can significantly reduce the processing speed for documents that require multiple compilation runs, since each markdown document is converted multiple times needlessly.

This behavior will only be used when the `finalizeCache` option is disabled.

```
\@@_add_lua_option:nnn
  { eagerCache }
  { boolean }
  { true }
defaultOptions.eagerCache = true
```

`experimental`=`true`, `false` default: `false`

`true` Experimental features that are planned to be the new default in the next major release of the Markdown package will be enabled.

At the moment, this just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

`false` Experimental features will be disabled.

```
\@@_add_lua_option:nnn
  { experimental }
  { boolean }
  { false }
defaultOptions.experimental = false
```

`singletonCache`=`true`, `false` default: `true`

`true` Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time- and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions.

This has been the default behavior since version 3.0.0 of the Markdown package.

`false` Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)[6].

This was the default behavior until version 3.0.0 of the Markdown package.

```
\@@_add_lua_option:nnn
  { singletonCache }
  { boolean }
  { true }
defaultOptions.singletonCache = true
local singletonCache = {
  convert = nil,
  options = nil,
}
```

`unicodeNormalization`=`true`, `false` default: `true`

`true` Markdown documents will be normalized using one of the four Unicode normalization forms[7] before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

`false` Markdown documents will not be Unicode-normalized before conversion.

[6] See https://github.com/witiko/markdown/pull/226#issuecomment-1599641634.
[7] See https://unicode.org/faq/normalization.html.

```
\@@_add_lua_option:nnn
  { unicodeNormalization }
  { boolean }
  { true }
defaultOptions.unicodeNormalization = true
```

`unicodeNormalizationForm`=`nfc`, `nfd`, `nfkc`, `nfkd`
default: `nfc`

`nfc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form C (NFC) before conversion.

`nfd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form D (NFD) before conversion.

`nfkc` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KC (NFKC) before conversion.

`nfkd` When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form KD (NFKD) before conversion.

```
\@@_add_lua_option:nnn
  { unicodeNormalizationForm }
  { string }
  { nfc }
defaultOptions.unicodeNormalizationForm = "nfc"
```

2.1.5 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 TEX 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.

```
\@@_add_lua_option:nnn
  { cacheDir }
  { path }
  { \markdownOptionOutputDir / _markdown_\jobname }
defaultOptions.cacheDir = "."
```

`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 when the `contentBlocks` option is enabled. See Section 2.2.5.9 for more information.

```
\@@_add_lua_option:nnn
  { contentBlocksLanguageMap }
  { path }
  { markdown-languages.json }
defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`debugExtensionsFileName`=⟨*filename*⟩ default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied.

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

`frozenCacheFileName`=⟨*path*⟩ default: `frozenCache.tex`

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

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

```
\@@_add_lua_option:nnn
  { frozenCacheFileName }
  { path }
  { \markdownOptionCacheDir / frozenCache.tex }
defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.6 Parser Options

`autoIdentifiers`=`true`, `false` default: `false`

`true` Enable the Pandoc auto identifiers syntax extension[8]:

```
The following heading received the identifier `sesame-street`:

# 123 Sesame Street
```

`false` Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
\@@_add_lua_option:nnn
  { autoIdentifiers }
  { boolean }
  { false }
defaultOptions.autoIdentifiers = false
```

`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.

```
\@@_add_lua_option:nnn
  { blankBeforeBlockquote }
  { boolean }
  { false }
defaultOptions.blankBeforeBlockquote = false
```

[8] See https://pandoc.org/MANUAL.html#extension-auto_identifiers.

`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.

```
\@@_add_lua_option:nnn
  { blankBeforeCodeFence }
  { boolean }
  { false }
defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeDivFence`=`true`, `false` default: `false`

`true` Require a blank line before the closing fence of a fenced div.

`false` Do not require a blank line before the closing fence of a fenced div.

```
\@@_add_lua_option:nnn
  { blankBeforeDivFence }
  { boolean }
  { false }
defaultOptions.blankBeforeDivFence = 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.

```
\@@_add_lua_option:nnn
  { blankBeforeHeading }
  { boolean }
  { false }
defaultOptions.blankBeforeHeading = false
```

`blankBeforeList`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following list.

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

```
\@@_add_lua_option:nnn
  { blankBeforeList }
  { boolean }
  { false }
defaultOptions.blankBeforeList = false
```

`bracketedSpans`=`true`, `false` default: `false`

`true` Enable the Pandoc bracketed span syntax extension[9]:

```
[This is *some text*]{.class key=val}
```

`false` Disable the Pandoc bracketed span syntax extension.

```
\@@_add_lua_option:nnn
  { bracketedSpans }
  { boolean }
  { false }
defaultOptions.bracketedSpans = false
```

`breakableBlockquotes`=`true`, `false` default: `true`

`true` A blank line separates block quotes.

`false` Blank lines in the middle of a block quote are ignored.

```
\@@_add_lua_option:nnn
  { breakableBlockquotes }
  { boolean }
  { true }
defaultOptions.breakableBlockquotes = true
```

`citationNbsps`=`true`, `false` default: `false`

`true` Replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

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

```
\@@_add_lua_option:nnn
  { citationNbsps }
  { boolean }
  { true }
defaultOptions.citationNbsps = true
```

[9] See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`citations`=true, false default: `false`

`true` Enable the Pandoc citation syntax extension[10]:

```
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.

```
\@@_add_lua_option:nnn
  { citations }
  { boolean }
  { false }
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.''
```

```
\@@_add_lua_option:nnn
  { codeSpans }
  { boolean }
  { true }
defaultOptions.codeSpans = true
```

[10] See https://pandoc.org/MANUAL.html#extension-citations.

`contentBlocks`=true, false default: `false`

true

: Enable the iA Writer content blocks syntax extension [5]:

```
``` md
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.

```
\@@_add_lua_option:nnn
  { contentBlocks }
  { boolean }
  { false }
defaultOptions.contentBlocks = false
```

`contentLevel`=block, inline default: `block`

`block` Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

`inline` Treat all content as inline content.

```
- this is a text
- not a list
```

```
\@@_add_lua_option:nnn
  { contentLevel }
  { string }
  { block }
defaultOptions.contentLevel = "block"
```

`debugExtensions`=`true`, `false` default: `false`

`true` Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

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

```
\@@_add_lua_option:nnn
  { debugExtensions }
  { boolean }
  { false }
defaultOptions.debugExtensions = false
```

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

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

```
\@@_add_lua_option:nnn
  { definitionLists }
  { boolean }
  { false }
defaultOptions.definitionLists = false
```

`ensureJekyllData`=`true`, `false` default: `false`

`false` When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. Otherwise, the markdown document is processed as markdown text.

`true` When the `jekyllData` and `expectJekyllData` options are enabled, then a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata. Otherwise, an error is produced.

```
\@@_add_lua_option:nnn
  { ensureJekyllData }
  { boolean }
  { false }
defaultOptions.ensureJekyllData = false
```

`expectJekyllData`=`true`, `false` default: `false`

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

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

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

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

```
\@@_add_lua_option:nnn
  { expectJekyllData }
  { boolean }
  { false }
defaultOptions.expectJekyllData = false
```

`extensions`=⟨*filenames*⟩

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

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

```
local strike_through = {
  api_version = 2,
  grammar_version = 4,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
```

```
    local function between(p, starter, ender)
      ender = lpeg.B(nonspacechar) * ender
      return (starter * #nonspacechar
             * lpeg.Ct(p * (p - ender)^0) * ender)
    end

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

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

return strike_through
```

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

```
metadata.user_extension_api_version = 2
metadata.grammar_version = 4
```

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

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `reader` object, such as the `reader->insert_pattern` and `reader->add_special_character` methods, see Section 2.1.2.

```
\cs_generate_variant:Nn
  \@@_add_lua_option:nnn
  { nnV }
\@@_add_lua_option:nnV
  { extensions }
  { clist }
  \c_empty_clist
```

```
defaultOptions.extensions = {}
```

`fancyLists`=`true`, `false` default: `false`

`true` Enable the Pandoc fancy list syntax extension[11]:

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

`false` Disable the Pandoc fancy list syntax extension.

```
\@@_add_lua_option:nnn
  { fancyLists }
  { boolean }
  { false }
defaultOptions.fancyLists = false
```

`fencedCode`=`true`, `false` default: `true`

`true` Enable the commonmark fenced code block extension:

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

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

`false` Disable the commonmark fenced code block extension.

```
\@@_add_lua_option:nnn
  { fencedCode }
  { boolean }
  { true }
defaultOptions.fencedCode = true
```

[11] See https://pandoc.org/MANUAL.html#org-fancy-lists.

`fencedCodeAttributes`=true, false default: `false`

`true` Enable the Pandoc fenced code attribute syntax extension[12]:

```
~~~~ {#mycode .haskell .numberLines startFrom=100}
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

`false` Disable the Pandoc fenced code attribute syntax extension.

```
\@@_add_lua_option:nnn
  { fencedCodeAttributes }
  { boolean }
  { false }
defaultOptions.fencedCodeAttributes = false
```

`fencedDivs`=true, false default: `false`

`true` Enable the Pandoc fenced div syntax extension[13]:

```
::::: {#special .sidebar}
Here is a paragraph.

And another.
:::::
```

`false` Disable the Pandoc fenced div syntax extension.

```
\@@_add_lua_option:nnn
  { fencedDivs }
  { boolean }
  { false }
defaultOptions.fencedDivs = false
```

[12] See https://pandoc.org/MANUAL.html#extension-fenced_code_attributes.
[13] See https://pandoc.org/MANUAL.html#extension-fenced_divs.

`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 TEX document that contains markdown documents without invoking Lua using the `frozenCache` plain TEX option. As a result, the plain TEX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
\@@_add_lua_option:nnn
  { finalizeCache }
  { boolean }
  { false }
defaultOptions.finalizeCache = false
```

`frozenCacheCounter`=⟨*number*⟩ default: `0`

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

Each frozen cache entry will define a TEX macro `\markdownFrozenCache`⟨*number*⟩ that will typeset markdown document number ⟨*number*⟩.

```
\@@_add_lua_option:nnn
  { frozenCacheCounter }
  { counter }
  { 0 }
defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers`=`true`, `false` default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension[14]:

```
The following heading received the identifier `123-sesame-street`:

# 123 Sesame Street
```

`false` Disable the Pandoc GitHub-flavored auto identifiers syntax extension.

[14] See https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers.

See also the option autoIdentifiers.

```
\@@_add_lua_option:nnn
  { gfmAutoIdentifiers }
  { boolean }
  { false }
defaultOptions.gfmAutoIdentifiers = false
```

hashEnumerators=true, false default: false

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

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

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

```
\@@_add_lua_option:nnn
  { hashEnumerators }
  { boolean }
  { false }
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}
==================
```

false Disable the assignment of HTML attributes to headings.

```
\@@_add_lua_option:nnn
  { headerAttributes }
  { boolean }
  { false }
defaultOptions.headerAttributes = false
```

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

`true` Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.

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

```
\@@_add_lua_option:nnn
  { html }
  { boolean }
  { true }
defaultOptions.html = true
```

`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.

The `hybrid` option makes it difficult to untangle TEX input from markdown text, which makes documents written with the `hybrid` option less interoperable and more difficult to read for authors. Therefore, the option has been soft-deprecated in version 3.7.1 of the Markdown package: It will never be removed but using it prints a warning and is discouraged.

Consider one of the following better alternatives for mixing TEX and markdown:

- With the `contentBlocks` option, authors can move large blocks of TeX code to separate files and include them in their markdown documents as external resources:

```
Here is a mathematical formula:

 /math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as TEX code:

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

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

- With options `texMathDollars`, `texMathSingleBackslash`, and `texMathDoubleBackslash`, authors can freely type TEX commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.

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

```
\@@_add_lua_option:nnn
  { hybrid }
  { boolean }
  { false }
defaultOptions.hybrid = false
```

`inlineCodeAttributes`=`true`, `false` default: `false`

`true` Enable the Pandoc inline code span attribute extension[15]:

```
`<$>`{.haskell}
```

[15] See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.

`false` Enable the Pandoc inline code span attribute extension.

```
\@@_add_lua_option:nnn
  { inlineCodeAttributes }
  { boolean }
  { false }
defaultOptions.inlineCodeAttributes = false
```

`inlineNotes`=true, false default: `false`

`true` Enable the Pandoc inline note syntax extension[16]:

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

`false` Disable the Pandoc inline note syntax extension.

```
\@@_add_lua_option:nnn
  { inlineNotes }
  { boolean }
  { false }
defaultOptions.inlineNotes = false
```

`jekyllData`=true, false default: `false`

`true` Enable the Pandoc YAML metadata block syntax extension[17] for entering metadata in YAML:

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

  It consists of two paragraphs.
---
```

[16] See https://pandoc.org/MANUAL.html#extension-inline_notes.
[17] See https://pandoc.org/MANUAL.html#extension-yaml_metadata_block.

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

```
\@@_add_lua_option:nnn
  { jekyllData }
  { boolean }
  { false }
defaultOptions.jekyllData = false
```

`linkAttributes`=true, false default: `false`

`true` Enable the Pandoc link and image attribute syntax extension[18]:

```
An inline ![image](foo.jpg){#id .class width=30 height=20px}
and a reference ![image][ref] with attributes.

[ref]: foo.jpg "optional title" {#id .class key=val key2=val2}
```

`false` Enable the Pandoc link and image attribute syntax extension.

```
\@@_add_lua_option:nnn
  { linkAttributes }
  { boolean }
  { false }
defaultOptions.linkAttributes = false
```

`lineBlocks`=true, false default: `false`

`true` Enable the Pandoc line block syntax extension[19]:

```
| this is a line block that
| spans multiple
| even
  discontinuous
| lines
```

`false` Disable the Pandoc line block syntax extension.

```
\@@_add_lua_option:nnn
  { lineBlocks }
  { boolean }
  { false }
defaultOptions.lineBlocks = false
```

[18] See https://pandoc.org/MANUAL.html#extension-link_attributes.
[19] See https://pandoc.org/MANUAL.html#extension-line_blocks.

`mark`=true, `false` default: `false`

`true` Enable the Pandoc mark syntax extension[20]:

```
This ==is highlighted text.==
```

`false` Disable the Pandoc mark syntax extension.

```
\@@_add_lua_option:nnn
  { mark }
  { boolean }
  { false }
defaultOptions.mark = false
```

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

`true` Enable the Pandoc note syntax extension[21]:

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

[^1]: Here is the note.

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

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

        { some.code }

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

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

`false` Disable the Pandoc note syntax extension.

```
\@@_add_lua_option:nnn
  { notes }
  { boolean }
  { false }
defaultOptions.notes = false
```

[20] See https://pandoc.org/MANUAL.html#extension-mark.
[21] See https://pandoc.org/MANUAL.html#extension-footnotes.

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

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

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

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

```
\@@_add_lua_option:nnn
  { pipeTables }
  { boolean }
  { false }
defaultOptions.pipeTables = false
```

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

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

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

```
\@@_add_lua_option:nnn
  { preserveTabs }
  { boolean }
  { true }
defaultOptions.preserveTabs = true
```

`rawAttribute`=true, false default: `false`

`true` Enable the Pandoc raw attribute syntax extension[22]:

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

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

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

[22] See https://pandoc.org/MANUAL.html#extension-raw_attribute.
```

```
        c & d
    \end{dcases}
\]
```

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

`false` Disable the Pandoc raw attribute syntax extension.

```
\@@_add_lua_option:nnn
  { rawAttribute }
  { boolean }
  { false }
defaultOptions.rawAttribute = false
```

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

`true` Enable relative references[23] in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
==========
In this paper, we have discovered that most
grandmas would rather eat dinner with their
grandchildren than get eaten. Begone, wolf!
```

`false` Disable relative references in autolinks.

```
\@@_add_lua_option:nnn
  { relativeReferences }
  { boolean }
  { false }
defaultOptions.relativeReferences = false
```

[23] See https://datatracker.ietf.org/doc/html/rfc3986#section-4.2.

`shiftHeadings`=⟨*shift amount*⟩ default: `0`

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

```
\@@_add_lua_option:nnn
  { shiftHeadings }
  { number }
  { 0 }
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 (`$`) selects the end of a document.
- `^`⟨*identifier*⟩ selects the beginning of a section (see the `headerAttributes` option) or a fenced div (see the `fencedDivs` option) with the HTML attribute `#`⟨*identifier*⟩.
- `$`⟨*identifier*⟩ selects the end of a section with the HTML attribute `#`⟨*identifier*⟩.
- ⟨*identifier*⟩ corresponds to `^`⟨*identifier*⟩ for the first selector and to `$`⟨*identifier*⟩ for the second selector.

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

```
\@@_add_lua_option:nnn
  { slice }
  { slice }
  { ^~$ }
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.

```
\@@_add_lua_option:nnn
  { smartEllipses }
  { boolean }
  { false }
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 `\markdownRendererOlItemWithNumber` TEX macro.

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

```
\@@_add_lua_option:nnn
  { startNumber }
  { boolean }
  { true }
defaultOptions.startNumber = true
```

`strikeThrough`=`true`, `false` default: `false`

`true` Enable the Pandoc strike-through syntax extension[24]:

```
This ~~is deleted text.~~
```

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

```
\@@_add_lua_option:nnn
  { strikeThrough }
  { boolean }
  { false }
defaultOptions.strikeThrough = false
```

[24] See https://pandoc.org/MANUAL.html#extension-strikeout.

`stripIndent`=true, false default: `false`

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

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

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

```
\@@_add_lua_option:nnn
  { stripIndent }
  { boolean }
  { false }
defaultOptions.stripIndent = false
```

`subscripts`=true, false default: `false`

`true` Enable the Pandoc subscript syntax extension[25]:

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

`false` Disable the Pandoc subscript syntax extension.

```
\@@_add_lua_option:nnn
  { subscripts }
  { boolean }
  { false }
defaultOptions.subscripts = false
```

[25] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.

`superscripts`=true, false default: `false`

`true` Enable the Pandoc superscript syntax extension[26]:

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

`false` Disable the Pandoc superscript syntax extension.

```
\@@_add_lua_option:nnn
  { superscripts }
  { boolean }
  { false }
defaultOptions.superscripts = false
```

`tableAttributes`=true, false default: `false`

true

: Enable the assignment of HTML attributes to table captions (see the `tableCaptions` option).

```
``` md
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

  : Demonstration of pipe table syntax. {#example-table}
```
```

`false` Disable the assignment of HTML attributes to table captions.

```
\@@_add_lua_option:nnn
  { tableAttributes }
  { boolean }
  { false }
defaultOptions.tableAttributes = false
```

[26] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.

`tableCaptions`=true, `false` default: `false`

true

: Enable the Pandoc table caption syntax extension[27] for pipe tables (see the `pipeTables` option).

```
``` md
| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

  : Demonstration of pipe table syntax.
``````
```

`false` Disable the Pandoc table caption syntax extension.

```
\@@_add_lua_option:nnn
  { tableCaptions }
  { boolean }
  { false }
defaultOptions.tableCaptions = false
```

`taskLists`=true, `false` default: `false`

`true` Enable the Pandoc task list syntax extension[28]:

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

`false` Disable the Pandoc task list syntax extension.

```
\@@_add_lua_option:nnn
  { taskLists }
  { boolean }
  { false }
defaultOptions.taskLists = false
```

[27] See https://pandoc.org/MANUAL.html#extension-table_captions.
[28] See https://pandoc.org/MANUAL.html#extension-task_lists.

`texComments`=`true`, `false` default: `false`

`true` Strip TEX-style comments.

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

Always enabled when `hybrid` is enabled.

`false` Do not strip TEX-style comments.

```
\@@_add_lua_option:nnn
  { texComments }
  { boolean }
  { false }
defaultOptions.texComments = false
```

`texMathDollars`=`true`, `false` default: `false`

`true` Enable the Pandoc dollar math syntax extension[29]:

```
inline math: $E=mc^2$

display math: $$E=mc^2$$
```

`false` Disable the Pandoc dollar math syntax extension.

```
\@@_add_lua_option:nnn
  { texMathDollars }
  { boolean }
  { false }
defaultOptions.texMathDollars = false
```

[29] See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.

`texMathDoubleBackslash`=`true`, `false` default: `false`

`true` Enable the Pandoc double backslash math syntax extension[30]:

```
inline math: \\(E=mc^2\\)

display math: \\[E=mc^2\\]
```

`false` Disable the Pandoc double backslash math syntax extension.

```
\@@_add_lua_option:nnn
  { texMathDoubleBackslash }
  { boolean }
  { false }
defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash`=`true`, `false` default: `false`

`true` Enable the Pandoc single backslash math syntax extension[31]:

```
inline math: \(E=mc^2\)

display math: \[E=mc^2\]
```

`false` Disable the Pandoc single backslash math syntax extension.

```
\@@_add_lua_option:nnn
  { texMathSingleBackslash }
  { boolean }
  { false }
defaultOptions.texMathSingleBackslash = false
```

`tightLists`=`true`, `false` default: `true`

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

[30] See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.
[31] See https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash.

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

- This is

  not a tight

- unordered list.
```

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

```
\@@_add_lua_option:nnn
  { tightLists }
  { boolean }
  { true }
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.

```
\@@_add_lua_option:nnn
  { underscores }
  { boolean }
  { true }
\ExplSyntaxOff
defaultOptions.underscores = true
```

2.1.7 Command-Line Interface

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

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

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

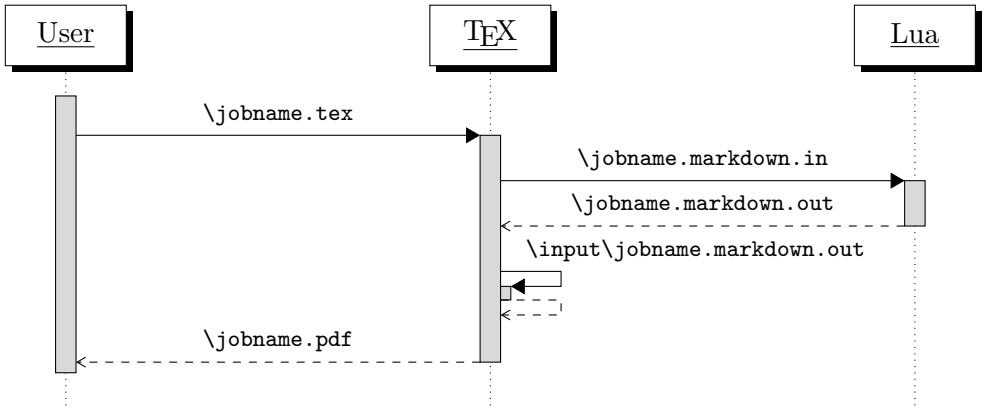

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
737
738 local HELP_STRING = [[
739 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
740 where OPTIONS are documented in the Lua interface section of the
741 technical Markdown package documentation.
742
743 When OUTPUT_FILE is unspecified, the result of the conversion will be
744 written to the standard output. When INPUT_FILE is also unspecified, the
745 result of the conversion will be read from the standard input.
746
747 Report bugs to: witiko@mail.muni.cz
748 Markdown package home page: <https://github.com/witiko/markdown>]]
749
```

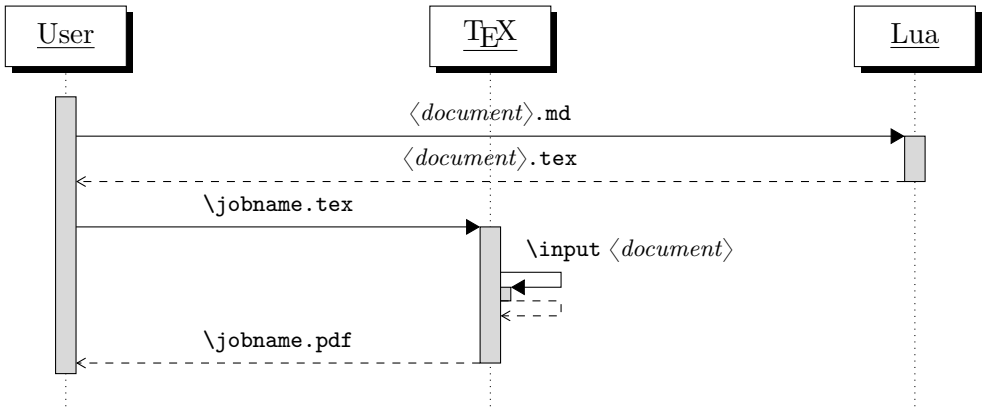

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
local VERSION_STRING = [[
markdown-cli (Markdown) ]] .. metadata.version .. [[

Copyright (C) ]] .. table.concat(metadata.copyright,
                                 "\nCopyright (C) ") .. [[

License: ]] .. metadata.license

local function warn(s)
  io.stderr:write("Warning: " .. s .. "\n")
end

local function error(s)
  io.stderr:write("Error: " .. s .. "\n")
  os.exit(1)
end
```

To make it easier to copy-and-paste options from Pandoc [6] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [7] also show that snake_case is faster to read than camelCase.

```
local function camel_case(option_name)
  local cased_option_name = option_name:gsub("_(%l)", function(match)
    return match:sub(2, 2):upper()
  end)
  return cased_option_name
end

local function snake_case(option_name)
  local cased_option_name = option_name:gsub("%l%u", function(match)
```

```
775     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
776   end)
777   return cased_option_name
778 end
779
780 local cases = {camel_case, snake_case}
781 local various_case_options = {}
782 for option_name, _ in pairs(defaultOptions) do
783   for _, case in ipairs(cases) do
784     various_case_options[case(option_name)] = option_name
785   end
786 end
787
788 local process_options = true
789 local options = {}
790 local input_filename
791 local output_filename
792 for i = 1, #arg do
793   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.

```
794     if arg[i] == "--" then
795       process_options = false
796       goto continue
```

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

```
797     elseif arg[i]:match("=") then
798       local key, value = arg[i]:match("(.-)=(.*)")
799       if defaultOptions[key] == nil and
800          various_case_options[key] ~= nil then
801         key = various_case_options[key]
802       end
```

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

```
803       local default_type = type(defaultOptions[key])
804       if default_type == "boolean" then
805         options[key] = (value == "true")
806       elseif default_type == "number" then
807         options[key] = tonumber(value)
808       elseif default_type == "table" then
809         options[key] = {}
810         for item in value:gmatch("[^ ,]+") do
```

```
811           table.insert(options[key], item)
812         end
813       else
814         if default_type ~= "string" then
815           if default_type == "nil" then
816             warn('Option "' .. key .. '" not recognized.')
817           else
818             warn('Option "' .. key .. '" type not recognized, ' ..
819                  'please file a report to the package maintainer.')
820           end
821           warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
822                key .. '" as a string.')
823         end
824         options[key] = value
825       end
826       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.

```
827     elseif arg[i] == "--help" or arg[i] == "-h" then
828       print(HELP_STRING)
829       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.

```
830     elseif arg[i] == "--version" or arg[i] == "-v" then
831       print(VERSION_STRING)
832       os.exit()
833     end
834   end
```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a TEX document.

```
835   if input_filename == nil then
836     input_filename = arg[i]
```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the TEX document that will result from the conversion.

```
837   elseif output_filename == nil then
838     output_filename = arg[i]
839   else
840     error('Unexpected argument: "' .. arg[i] .. '".')
841   end
842   ::continue::
```

```
end
```

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

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

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

```
\input hello
```

2.2 Plain 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 2.1.3) used during the conversion from markdown to plain TEX and for changing the way markdown the tokens are rendered.

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

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 `\input`ting the file.

2.2.1 Typesetting Markdown and YAML

The interface exposes the `\markdownBegin`, `\markdownEnd`, `\yamlBegin`, `\yamlEnd`, `\markinline`, `\markdownInput`, `\yamlInput`, and `\markdownEscape` macros.

2.2.1.1 Typesetting Markdown and YAML directly

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

```
\let\markdownBegin\relax
\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 corrolary, 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 TEX [8, p. 46]. As a corrolary, 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 TEX code, the characters `c`, `e`, and `f` will not appear in the output.

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

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

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

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

The `\yamlBegin` macro marks the beginning of an YAML document fragment and the `\yamlEnd` macro marks its end.

```
\let\yamlBegin\relax
\def\yamlEnd{\markdownEnd\endgroup}
```

The `\yamlBegin` and `\yamlEnd` macros are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```
\input markdown
\yamlBegin
title: _Hello_ **world** ...
```

```
author: John Doe
\yamlEnd
\bye
```

The above code has the same effect as the below code:

```
\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownBegin
title: _Hello_ **world** ...
author: John Doe
\markdownEnd
\bye
```

You can use the `\markinline` macro to input inline markdown content.

```
\let\markinline\relax
```

The following example plain TEX code showcases the usage of the `\markinline` macro:

```
\input markdown
\markinline{_Hello_ **world**}
\bye
```

The above code has the same effect as the below code:

```
\input markdown
\markdownSetup{contentLevel=inline}
\markdownBegin
_Hello_ **world** ...
\markdownEnd
\bye
```

The `\markinline` macro is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

2.2.1.2 Typesetting Markdown and YAML from external documents

You can use the `\markdownInput` macro to include markdown documents, similarly to how you might use the `\input` TEX primitive to include TEX documents. The `\markdownInput` macro accepts a single parameter with the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TEX.

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

The macro \markdownInput is not subject to the limitations of the \markdownBegin and \markdownEnd macros.

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

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

You can use the \yamlInput macro to include YAML documents. similarly to how you might use the \input TEX primitive to include TEX documents. The \yamlInput macro accepts a single parameter with the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain TEX.

```
\def\yamlInput#1{%
  \begingroup
  \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
  \markdownInput{#1}%
  \endgroup
}%
```

The macro \yamlInput is also not subject to the limitations of the \markdownBegin and \markdownEnd macros.

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

```
\input markdown
\yamlInput{hello.yml}
\bye
```

The above code has the same effect as the below code:

```
\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownInput{hello.yml}
\bye
```

2.2.1.3 Typesetting TeX from inside Markdown and YAML documents

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

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

2.2.2 Options

The plain TEX options are represented by TEX commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain TEX interface.

To determine whether plain TEX is the top layer or if there are other layers above plain TEX, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that plain TEX is the top layer.

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_plain_tex_tl
  }
```

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

```
\seq_new:N \g_@@_plain_tex_options_seq
```

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

```
\prop_new:N \g_@@_plain_tex_option_types_prop
\prop_new:N \g_@@_default_plain_tex_options_prop
\seq_gput_right:NV
  \g_@@_option_layers_seq
  \c_@@_option_layer_plain_tex_tl
\cs_new:Nn
  \@@_add_plain_tex_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_plain_tex_tl
      { #1 }
      { #2 }
      { #3 }
  }
```

The plain TEX options may be also be specified via the `\markdownSetup` macro. Here, the plain TEX options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted

as ⟨*key*⟩`=true` if the `=`⟨*value*⟩ part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/options }
      { #1 }
  }
\cs_gset_eq:NN
  \markdownSetup
  \@@_setup:n
```

The command `\yamlSetup` is also available as an alias for the command `\markdownSetup`.

```
\cs_gset_eq:NN
  \yamlSetup
  \markdownSetup
```

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.

```
\prg_new_conditional:Nnn
  \@@_if_option:n
  { TF, T, F }
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \msg_error:nnxx
          { markdown }
          { expected-boolean-option }
          { #1 }
          { \l_tmpa_tl }
      }
    \@@_get_option_value:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNTF
      \l_tmpa_tl
      \c_@@_option_value_true_tl
      { \prg_return_true: }
      { \prg_return_false: }
```

```
  }
\msg_new:nnn
  { markdown }
  { expected-boolean-option }
  {
    Option~#1~has~type~#2,~
    but~a~boolean~was~expected.
  }
\let
  \markdownIfOption
  \@@_if_option:nTF
```

2.2.2.1 Finalizing and Freezing the Cache

The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `frozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain TEX document and their auxiliary files cached in the `cacheDir` directory.

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `finalizeCache` 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`.

```
\@@_add_plain_tex_option:nnn
  { frozenCache }
  { boolean }
  { 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 `finalizeCache` option.
4. Typeset the plain TEX document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain TEX document and the `cacheDir` directory.

2.2.2.2 File and Directory Names

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a TEX source. It defaults to `\jobname.markdown.in`.

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.

```
\@@_add_plain_tex_option:nnn
  { inputTempFileName }
  { path }
```

```
  { \jobname.markdown.in }
```

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 `.` or, since TEX Live 2024, to the value of the `-output-directory` option of your TEX engine.

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 `inputTempFileName` macro.

The `\markdownOptionOutputDir` macro has been deprecated and will be removed in the next major version of the Markdown package.

```
\@@_add_plain_tex_option:nnn
  { outputDir }
  { path }
  { . }
```

2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level TEX formats such as LATEX and ConTEXt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

The `\markdownOptionPlain` macro specifies whether higher-level TEX formats should only use the plain TEX default definitions or whether they should also use the format-specific default definitions. Whereas plain TEX default definitions only provide definitions for simple elements such as emphasis, strong emphasis, and paragraph separators, format-specific default definitions add support for more complex elements such as lists, tables, and citations. On the flip side, plain TEX default definitions load no extra resources and are rather stable, whereas format-specific default definitions load extra resources and are subject to a more rapid change.

Here is how you would enable the macro in a LATEX document:

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

Here is how you would enable the macro in a ConTEXt document:

```
\def\markdownOptionPlain{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
\@@_add_plain_tex_option:nnn
  { plain }
  { boolean }
  { false }
```

The `\markdownOptionNoDefaults` macro specifies whether we should prevent the loading of default definitions or not. This is useful in contexts, where we want to have total control over how all elements are rendered.

Here is how you would enable the macro in a LaTeX document:

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

Here is how you would enable the macro in a ConTeXt document:

```
\def\markdownOptionNoDefaults{true}
\usemodule[t][markdown]
```

The macro must be set before or during the loading of the package. Setting the macro after loading the package has no effect.

```
\@@_add_plain_tex_option:nnn
  { noDefaults }
  { boolean }
  { false }
```

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 sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing TeX package documentation using the Doc LaTeX package [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
\seq_gput_right:Nn
  \g_@@_plain_tex_options_seq
  { stripPercentSigns }
\prop_gput:Nnn
  \g_@@_plain_tex_option_types_prop
  { stripPercentSigns }
  { boolean }
\prop_gput:Nnx
  \g_@@_default_plain_tex_options_prop
  { stripPercentSigns }
  { false }
```

2.2.2.5 Generating Plain TEX Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain TEX macros and the key–value interface of the `\markdownSetup` macro for the above plain TEX options.

The command also defines macros and key–values that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain TEX implementation, only passed along to Lua.

Furthermore, the command also defines options and key–values for subsequently loaded layers that correspond to higher-level TEX formats such as LATEX and ConTEXt.

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 `inputTempFileName` macro.

```
\cs_new:Nn
  \@@_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_define_option_command:n
                { ####1 }
```

To make it easier to copy-and-paste options from Pandoc [6] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [7] also show that snake_case is faster to read than camelCase.

```
              \@@_with_various_cases:nn
                { ####1 }
                {
                  \@@_define_option_keyval:nnn
                    { ##1 }
                    { ####1 }
                    { ########1 }
                }
          }
      }
  }
\cs_new:Nn
  \@@_define_option_command:n
  {
```

Use the lt3luabridge library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```
    \str_if_eq:nnTF
      { #1 }
      { outputDir }
      { \@@_define_option_command_output_dir: }
      {
```

Do not override options defined before loading the package.

```
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_if_exist:cF
          { \l_tmpa_tl }
          {
            \@@_get_default_option_value:nN
              { #1 }
              \l_tmpa_tl
            \@@_set_option_value:nV
              { #1 }
              \l_tmpa_tl
          }
      }
  }
\ExplSyntaxOff
\input lt3luabridge.tex
```

Use the lt3luabridge library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since TeX Live 2024.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_define_option_command_output_dir:
  {
    \cs_if_free:NT
      \markdownOptionOutputDir
      {
        \bool_if:nTF
          {
            \cs_if_exist_p:N
              \luabridge_tl_set:Nn &&
            (
              \int_compare_p:nNn
                { \g_luabridge_method_int }
                =
                { \c_luabridge_method_directlua_int } ||
```

```
            \sys_if_shell_unrestricted_p:
          )
        }
        {
```

Set most catcodes to category 12 (other) to ensure that special characters in TEXMF_OUTPUT_DIRECTORY such as backslashes (\) are not interpreted as control sequences.

```
          \group_begin:
          \cctab_select:N
            \c_str_cctab
          \luabridge_tl_set:Nn
            \l_tmpa_tl
            { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
          \tl_gset:NV
            \markdownOptionOutputDir
            \l_tmpa_tl
          \group_end:
        }
        {
          \tl_gset:Nn
            \markdownOptionOutputDir
            { . }
        }
    }
  }
\cs_new:Nn
  \@@_set_option_value:nn
  {
    \@@_define_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNTF
      \c_@@_option_type_counter_tl
      \l_tmpa_tl
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \int_gset:cn
          { \l_tmpa_tl }
          { #2 }
      }
      {
        \@@_option_tl_to_csname:nN
```

```
          { #1 }
          \l_tmpa_tl
        \cs_set:cpn
          { \l_tmpa_tl }
          { #2 }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\cs_new:Nn
  \@@_define_option:n
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cT
      { \l_tmpa_tl }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpb_tl
        \str_if_eq:NNT
          \c_@@_option_type_counter_tl
          \l_tmpb_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \int_new:c
              { \l_tmpa_tl }
          }
      }
  }
\cs_new:Nn
  \@@_define_option_keyval:nnn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \str_if_eq:VVTF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/options }
```

```
        {
```

For boolean options, we also accept `yes` as an alias for `true` and `no` as an alias for `false`.

```
          #3 .code:n = {
            \tl_set:Nx
              \l_tmpa_tl
              {
                \str_case:nnF
                  { ##1 }
                  {
                    { yes } { true }
                    { no } { false }
                  }
                  { ##1 }
              }
            \@@_set_option_value:nV
              { #2 }
              \l_tmpa_tl
          },
          #3 .default:n = { true },
        }
    }
    {
      \keys_define:nn
        { markdown/options }
        {
          #3 .code:n = {
            \@@_set_option_value:nn
              { #2 }
              { ##1 }
          },
        }
    }
```

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

```
  \str_if_eq:VVT
    \l_tmpa_tl
    \c_@@_option_type_clist_tl
    {
      \tl_set:Nn
        \l_tmpa_tl
        { #3 }
```

```
        \tl_reverse:N
          \l_tmpa_tl
        \str_if_eq:enF
          {
            \tl_head:V
              \l_tmpa_tl
          }
          { s }
          {
            \msg_error:nnn
              { markdown }
              { malformed-name-for-clist-option }
              { #3 }
          }
        \tl_set:Nx
          \l_tmpa_tl
          {
            \tl_tail:V
              \l_tmpa_tl
          }
        \tl_reverse:N
          \l_tmpa_tl
        \tl_put_right:Nn
          \l_tmpa_tl
          {
            .code:n = {
              \@@_get_option_value:nN
                { #2 }
                \l_tmpa_tl
              \clist_set:NV
                \l_tmpa_clist
                { \l_tmpa_tl , { ##1 } }
              \@@_set_option_value:nV
                { #2 }
                \l_tmpa_clist
            }
          }
        \keys_define:nV
          { markdown/options }
          \l_tmpa_tl
      }
  }
\cs_generate_variant:Nn
  \clist_set:Nn
  { NV }
\cs_generate_variant:Nn
  \keys_define:nn
```

```
  { nV }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { p, F }
\msg_new:nnn
  { markdown }
  { malformed-name-for-clist-option }
  {
    Clist~option~name~#1~does~not~end~with~-s.
  }
```

If plain TEX is the top layer, we use the `\@@_define_option_commands_and_keyvals:` macro to define plain TEX option macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_option_commands_and_keyvals:
  }
\ExplSyntaxOff
```

2.2.3 Themes

User-defined themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The key–values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩, optionally followed by `@`⟨*theme version*⟩, load a TEX document (further referred to as *a theme*) named `markdowntheme`⟨*munged theme name*⟩`.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (`/`) for an underscore (`_`). The theme name must be *qualified* and contain no underscores or at signs (`@`). Themes are inspired by the Beamer LATEX package, which provides similar functionality with its `\usetheme` macro [10, Section 15.1].

A theme name is qualified if and only if it contains at least one forward slash. Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is ⟨*theme author*⟩`/`⟨*theme purpose*⟩`/`⟨*private naming scheme*⟩, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the TEX directory structure. For example, loading a theme named `witiko/beamer/MU` would load a TEX document package named `markdownthemewitiko_beamer_MU.tex`.

If `@`⟨*theme version*⟩ is specified after ⟨*theme name*⟩, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@`⟨*theme version*⟩ is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [11].

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/options }
  {
    theme .code:n = {
      \@@_set_theme:n
        { #1 }
    },
    import .code:n = {
      \tl_set:Nn
        \l_tmpa_tl
        { #1 }
```

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

```
      \tl_replace_all:NnV
        \l_tmpa_tl
        { / }
        \c_backslash_str
      \keys_set:nV
        { markdown/options/import }
        \l_tmpa_tl
    },
  }
```

To keep track of the current theme when themes are nested, we will maintain the stacks `\g_@@_theme_names_seq` and `\g_@@_theme_versions_seq` stack of theme names and versions, respectively. For convenience, the name of the current theme and version is also available in the macros `\g_@@_current_theme_tl` and `\markdownThemeVersion`, respectively.

```
\seq_new:N
  \g_@@_theme_names_seq
\seq_new:N
  \g_@@_theme_versions_seq
\tl_new:N
```

```
  \g_@@_current_theme_tl
\tl_gset:Nn
  \g_@@_current_theme_tl
  { }
\seq_gput_right:NV
  \g_@@_theme_names_seq
  \g_@@_current_theme_tl
\cs_new:Npn
  \markdownThemeVersion
  { }
\seq_gput_right:NV
  \g_@@_theme_versions_seq
  \g_@@_current_theme_tl
\cs_new:Nn
  \@@_set_theme:n
  {
```

First, we validate the theme name.

```
    \str_if_in:nnF
      { #1 }
      { / }
      {
        \msg_error:nnn
          { markdown }
          { unqualified-theme-name }
          { #1 }
      }
    \str_if_in:nnT
      { #1 }
      { _ }
      {
        \msg_error:nnn
          { markdown }
          { underscores-in-theme-name }
          { #1 }
      }
```

Next, we extract the theme version.

```
    \str_if_in:nnTF
      { #1 }
      { @ }
      {
        \regex_extract_once:nnN
          { (.*) @ (.*) }
          { #1 }
          \l_tmpa_seq
        \seq_gpop_left:NN
          \l_tmpa_seq
```

```
      \l_tmpa_tl
    \seq_gpop_left:NN
      \l_tmpa_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \l_tmpa_tl
    \seq_gpop_left:NN
      \l_tmpa_seq
      \l_tmpa_tl
    \cs_gset:Npe
      \markdownThemeVersion
      {
        \tl_use:N
          \l_tmpa_tl
      }
  }
  {
    \tl_gset:Nn
      \g_@@_current_theme_tl
      { #1 }
    \cs_gset:Npn
      \markdownThemeVersion
      { latest }
  }
```

Next, we munge the theme name.

```
\str_set:NV
  \l_tmpa_str
  \g_@@_current_theme_tl
\str_replace_all:Nnn
  \l_tmpa_str
  { / }
  { _ }
```

Finally, we load the theme. Before loading the theme, we push down the current name and version of the theme on the stack.

```
\tl_set:NV
  \l_tmpa_tl
  \g_@@_current_theme_tl
\tl_put_right:Nn
  \g_@@_current_theme_tl
  { / }
\seq_gput_right:NV
  \g_@@_theme_names_seq
  \g_@@_current_theme_tl
\seq_gput_right:NV
  \g_@@_theme_versions_seq
```

```
      \markdownThemeVersion
    \@@_load_theme:VeV
      \l_tmpa_tl
      { \markdownThemeVersion }
      \l_tmpa_str
```

After the theme has been loaded, we recover the name and version of the previous theme from the stack.

```
    \seq_gpop_right:NN
      \g_@@_theme_names_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_theme_names_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \l_tmpa_tl
    \seq_gpop_right:NN
      \g_@@_theme_versions_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_theme_versions_seq
      \l_tmpa_tl
    \cs_gset:Npe
      \markdownThemeVersion
      {
        \tl_use:N
          \l_tmpa_tl
      }
  }
\msg_new:nnnn
  { markdown }
  { unqualified-theme-name }
  { Won't~load~theme~with~unqualified~name~#1 }
  { Theme~names~must~contain~at~least~one~forward~slash }
\msg_new:nnnn
  { markdown }
  { underscores-in-theme-name }
  { Won't~load~theme~with~an~underscore~in~its~name~#1 }
  { Theme~names~must~not~contain~underscores~in~their~names }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NnV }
\cs_generate_variant:Nn
  \cs_gset:Npn
  { Npe }
```

We also define the prop `\g_@@_plain_tex_built_in_themes_prop` that contains

the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```
\prop_new:N
  \g_@@_plain_tex_built_in_themes_prop
```

Built-in plain TEX themes provided with the Markdown package include:

witiko/diagrams A theme that typesets fenced code blocks with the infostrings `dot`, `mermaid`, and `plantuml` as figures with diagrams produced with the command `dot` from Graphviz tools, the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`, and the command `plantuml` from the package PlantUML, respectively. The key-value attribute `caption` can be used to specify the caption of the figure. The remaining attributes are treated as image attributes.

```
\documentclass{article}
\usepackage[import=witiko/diagrams@v2, relativeReferences]{markdown}
\begin{document}
\begin{markdown}
``` dot {caption="An example directed graph" width=12cm #dot}
digraph tree {
  margin = 0;
  rankdir = "LR";

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

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

```
``` mermaid {caption="An example mindmap" width=9cm #mermaid}
mindmap
    root )base-idea(
        sub<br/>idea 1
            ((?))
        sub<br/>idea 2
            ((?))
        sub<br/>idea 3
            ((?))
        sub<br/>idea 4
            ((?))
```

``` plantuml {caption="An example UML sequence diagram" width=7cm #plantuml}
@startuml
' Define participants (actors)
participant "Client" as C
participant "Server" as S
participant "Database" as DB

' Diagram title
title Simple Request-Response Flow

' Messages
C -> S: Send Request
note over S: Process request

alt Request is valid
    S -> DB: Query Data
    DB -> S: Return Data
    S -> C: Respond with Data
else Request is invalid
    S -> C: Return Error
end
@enduml
```

See the diagrams in figures <#dot>, <#mermaid>, and <#plantuml>.
\end{markdown}
\end{document}
```

Typesetting the above document produces the output shown in figures 4, 5, and 6.

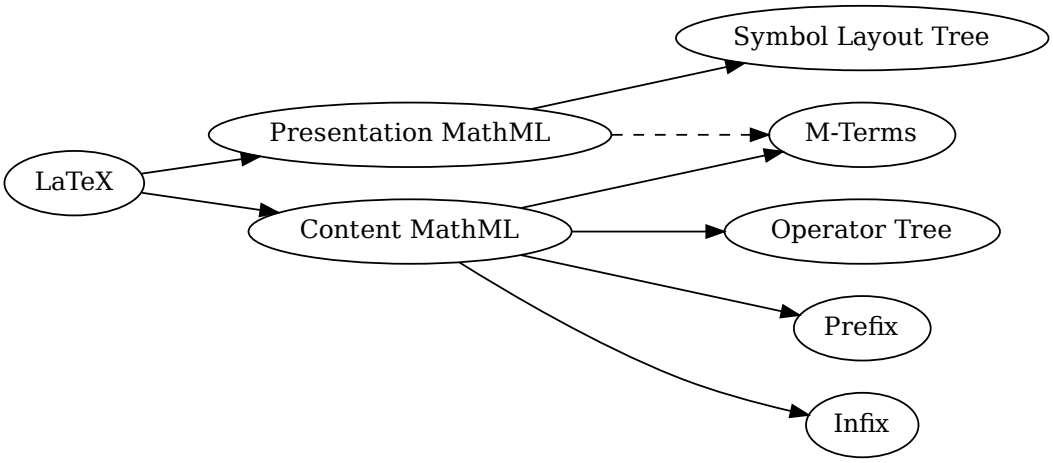

Figure 4: An example directed graph

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package `@mermaid-js/mermaid-cli`, and PlantUML installed. All these packages are already included in the Docker image `witiko/markdown`; consult `Dockerfile` to see how they are installed. The theme also requires shell access unless the `frozenCache` plain TEX option is enabled.

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

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

Typesetting the above document produces the output shown in Figure 7. The theme requires the catchfile LATEX package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `frozenCache` plain TEX option is enabled.

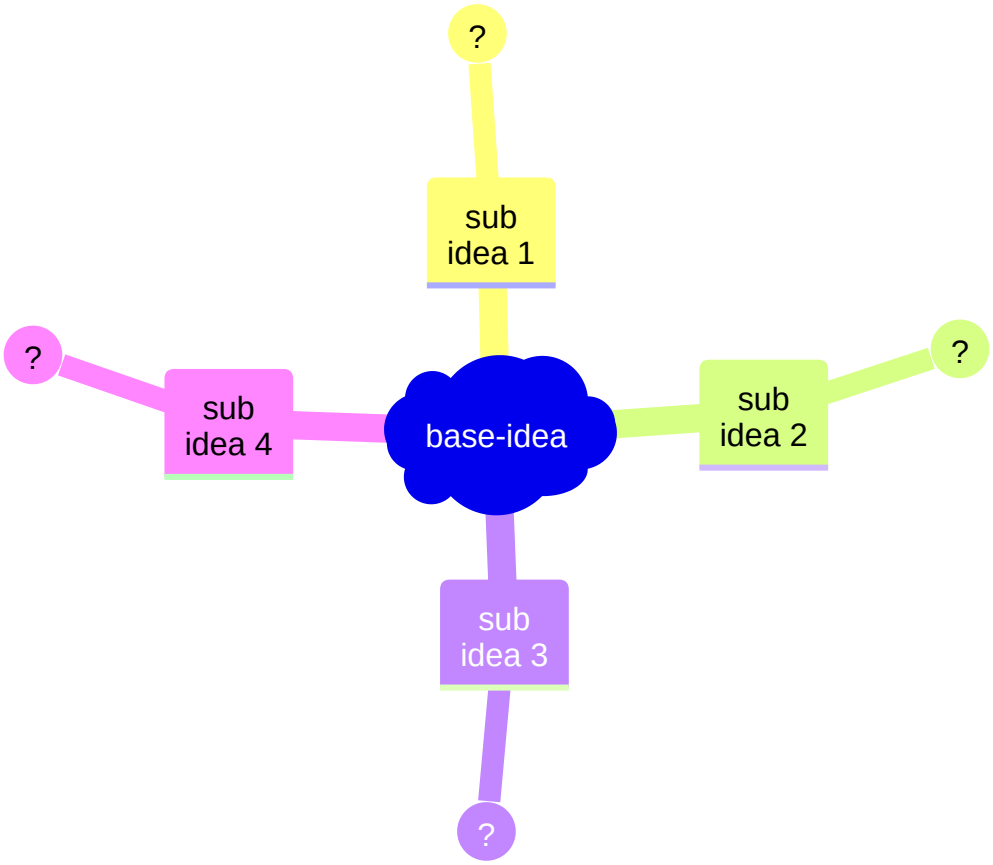

Figure 5: An example mindmap

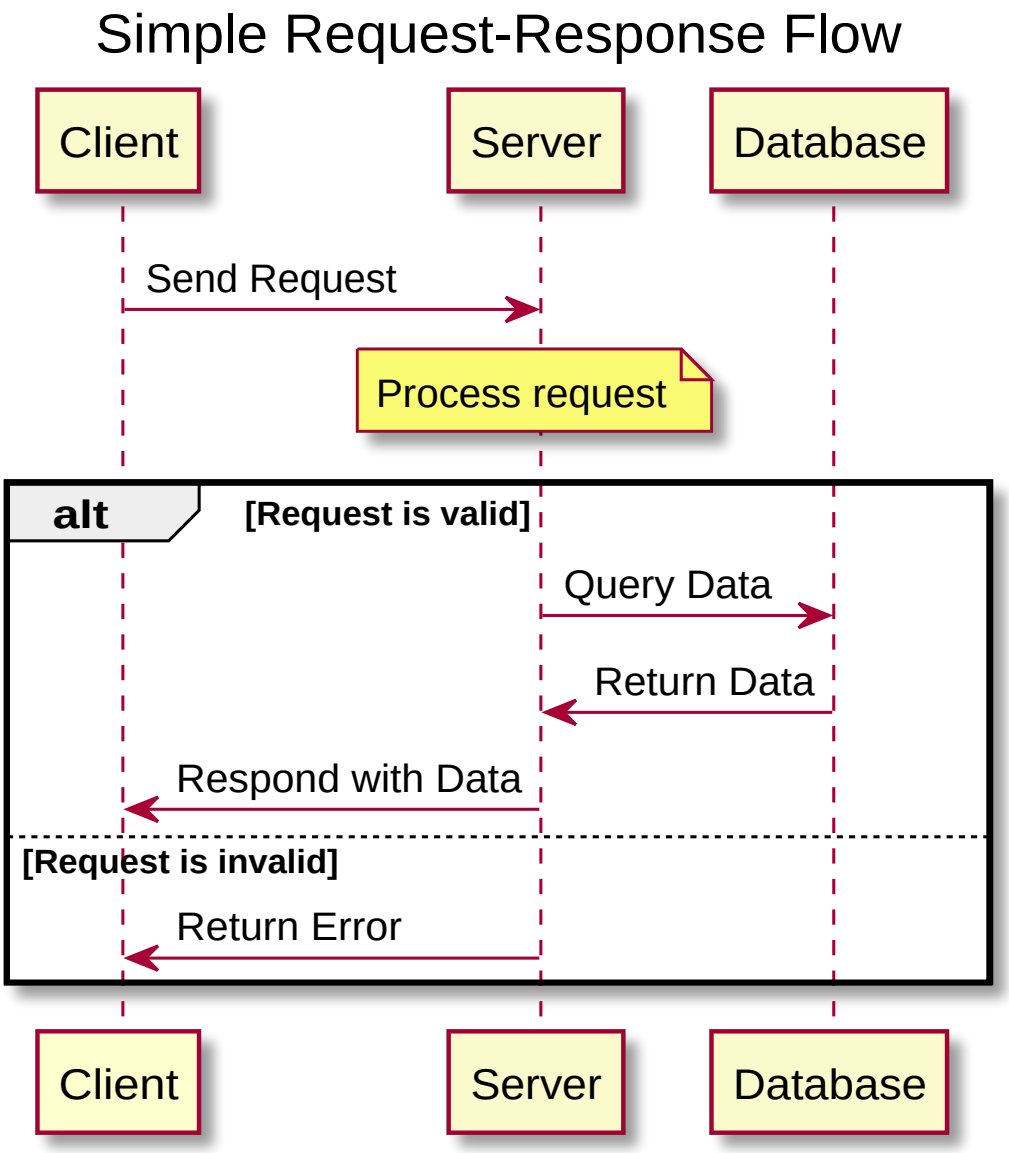

Figure 6: An example UML sequence diagram

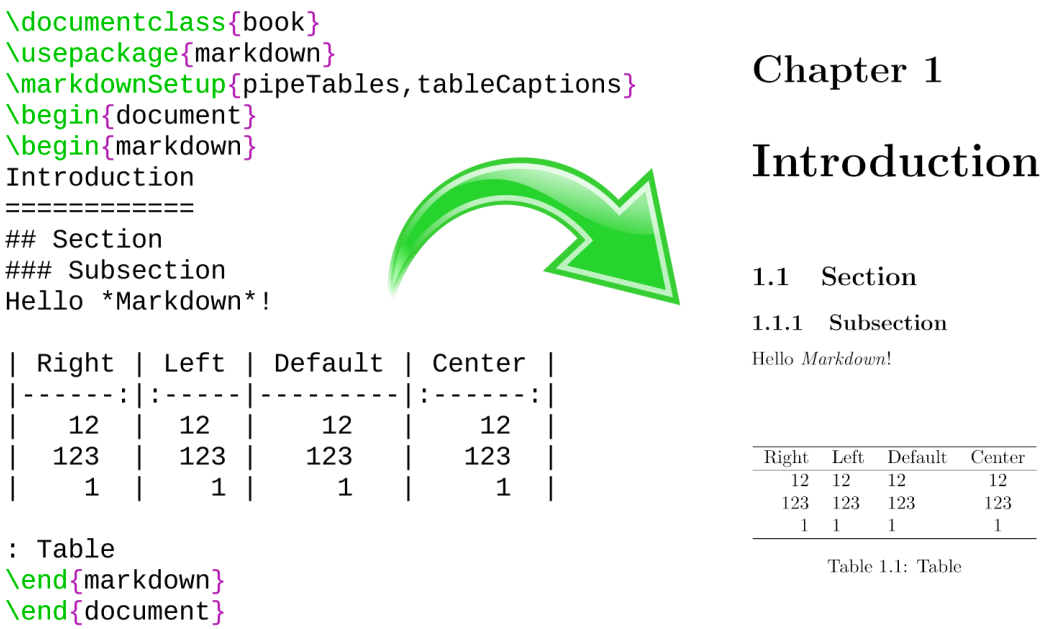

Figure 7: The banner of the Markdown package

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

```
\input markdown
\markdownSetup{import=witiko/tilde}
\markdownBegin
Bartel~Leendert van~der~Waerden
\markdownEnd
\bye
```

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

witiko/markdown/defaults A plain TEX theme with the default definitions of token renderer prototypes for plain TEX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

Please, see Section 3.2.2 for implementation details of the built-in plain TEX themes.

2.2.4 Snippets

We may set up options as *snippets* using the `\markdownSetupSnippet` macro and invoke them later. The `\markdownSetupSnippet` macro receives two arguments: the name of the snippet and the options to store.

```
\prop_new:N
  \g_@@_snippets_prop
\cs_new:Nn
  \@@_setup_snippet:nn
  {
    \tl_if_empty:nT
      { #1 }
      {
        \msg_error:nnn
          { markdown }
          { empty-snippet-name }
          { #1 }
      }
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
      \l_tmpa_tl
      { #1 }
    \@@_if_snippet_exists:nT
      { #1 }
      {
        \msg_warning:nnV
          { markdown }
          { redefined-snippet }
          \l_tmpa_tl
      }
    \keys_precompile:nnN
      { markdown/options }
      { #2 }
      \l_tmpb_tl
    \prop_gput:NVV
      \g_@@_snippets_prop
      \l_tmpa_tl
      \l_tmpb_tl
  }
\cs_gset_eq:NN
  \markdownSetupSnippet
  \@@_setup_snippet:nn
\msg_new:nnnn
  { markdown }
  { empty-snippet-name }
```

```
  { Empty~snippet~name~#1 }
  { Pick~a~non-empty~name~for~your~snippet }
\msg_new:nnn
  { markdown }
  { redefined-snippet }
  { Redefined~snippet~#1 }
```

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

```
\tl_new:N
  \l_@@_current_snippet_tl
\prg_new_conditional:Nnn
  \@@_if_snippet_exists:n
  { TF, T, F }
  {
    \tl_set:NV
      \l_@@_current_snippet_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
      \l_@@_current_snippet_tl
      { #1 }
    \prop_if_in:NVTF
      \g_@@_snippets_prop
      \l_@@_current_snippet_tl
      { \prg_return_true: }
      { \prg_return_false: }
  }
\cs_gset_eq:NN
  \markdownIfSnippetExists
  \@@_if_snippet_exists:nTF
```

The option with key `snippet` invokes a snippet named ⟨*value*⟩.

```
\keys_define:nn
  { markdown/options }
  {
    snippet .code:n = {
      \tl_set:NV
        \l_tmpa_tl
        \g_@@_current_theme_tl
      \tl_put_right:Nn
        \l_tmpa_tl
        { #1 }
      \@@_if_snippet_exists:nTF
        { #1 }
        {
          \prop_get:NVN
            \g_@@_snippets_prop
            \l_tmpa_tl
```

```
            \l_tmpb_tl
          \tl_use:N
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { undefined-snippet }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { undefined-snippet }
  { Can't~invoke~undefined~snippet~#1 }
\ExplSyntaxOff
```

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

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

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

\end{markdown}
\begin{markdown}[snippet=romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

If the `romanNumerals` snippet were defined in the `jdoe/lists` theme, we could import the `jdoe/lists` theme and use the qualified name `jdoe/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdoe/lists}
\begin{markdown}[snippet=jdoe/lists/romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

Alternatively, we can use the extended variant of the `import` LaTeX option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```
\markdownSetup{
  import = {
    jdoe/lists = romanNumerals,
  },
}
\begin{markdown}[snippet=romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

Furthermore, we can also specify the name of the snippet in the current namespace, which can be different from the name of the snippet in the `jdoe/lists` theme. For example, we can make the snippet `jdoe/lists/romanNumerals` available under the name `roman`.

```
\markdownSetup{
  import = {
    jdoe/lists = romanNumerals as roman,
  },
}
\begin{markdown}[snippet=roman]

The following ordered list will be preceded by roman numerals:
```

```
3. tres
4. quattuor

\end{markdown}
```

Several themes and/or snippets can be loaded at once using the extended variant of the `import` LaTeX option:

```
\markdownSetup{
  import = {
    jdoe/longpackagename/lists = {
      arabic as arabic1,
      roman,
      alphabetic,
    },
    jdoe/anotherlongpackagename/lists = {
      arabic as arabic2,
    },
    jdoe/yetanotherlongpackagename,
  },
}
```

```
\ExplSyntaxOn
\tl_new:N
  \l_@@_import_current_theme_tl
\keys_define:nn
  { markdown/options/import }
  {
```

If a theme name is given without a list of snippets to import, we assume that an empty list was given.

```
    unknown .default:n = {},
    unknown .code:n = {
```

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

```
      \tl_set_eq:NN
        \l_@@_import_current_theme_tl
        \l_keys_key_str
```

```
      \tl_replace_all:NVn
        \l_@@_import_current_theme_tl
        \c_backslash_str
        { / }
```

Here, we import the snippets.

```
      \clist_map_inline:nn
        { #1 }
        {
          \regex_extract_once:nnNTF
            { ^(.*?)\s+as\s+(.*?)$ }
            { ##1 }
            \l_tmpa_seq
            {
              \seq_pop:NN
                \l_tmpa_seq
                \l_tmpa_tl
              \seq_pop:NN
                \l_tmpa_seq
                \l_tmpa_tl
              \seq_pop:NN
                \l_tmpa_seq
                \l_tmpb_tl
            }
            {
              \tl_set:Nn
                \l_tmpa_tl
                { ##1 }
              \tl_set:Nn
                \l_tmpb_tl
                { ##1 }
            }
          \tl_put_left:Nn
            \l_tmpa_tl
            { / }
          \tl_put_left:NV
            \l_tmpa_tl
            \l_@@_import_current_theme_tl
          \@@_setup_snippet:Vx
            \l_tmpb_tl
            { snippet = { \l_tmpa_tl } }
        }
```

Here, we load the theme.

```
      \@@_set_theme:V
        \l_@@_import_current_theme_tl
    },
  }
```

```
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NVn }
\cs_generate_variant:Nn
  \@@_set_theme:n
  { V }
\cs_generate_variant:Nn
  \@@_setup_snippet:nn
  { Vx }
```

2.2.5 Token Renderers

The following TEX macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.6).

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

```
\seq_new:N \g_@@_renderers_seq
```

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

```
\prop_new:N \g_@@_renderer_arities_prop
\ExplSyntaxOff
```

2.2.5.1 Attribute Renderers

The following macros are only produced, when at least one of the following options for markdown attributes on different elements is enabled:

- `autoIdentifiers`
- `fencedCodeAttributes`
- `gfmAutoIdentifiers`
- `headerAttributes`
- `inlineCodeAttributes`
- `linkAttributes`

`\markdownRendererAttributeIdentifier` represents the ⟨*identifier*⟩ of a markdown element (`id="`⟨*identifier*⟩`"` in HTML and `#`⟨*identifier*⟩ in markdown attributes). The macro receives a single attribute that corresponds to the ⟨*identifier*⟩.

`\markdownRendererAttributeClassName` represents the ⟨*class name*⟩ of a markdown element (`class="`⟨*class name*⟩ `..."` in HTML and `.`⟨*class name*⟩ in markdown attributes). The macro receives a single attribute that corresponds to the ⟨*class name*⟩.

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererAttributeIdentifier
  {
    \markdownRendererAttributeIdentifierPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeIdentifier }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { attributeIdentifier }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererAttributeClassName
  {
    \markdownRendererAttributeClassNamePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeClassName }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { attributeClassName }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererAttributeKeyValue
  {
    \markdownRendererAttributeKeyValuePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeKeyValue }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { attributeKeyValue }
  { 2 }
\ExplSyntaxOff
```

2.2.5.2 Block Quote Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererBlockQuoteBegin
  {
    \markdownRendererBlockQuoteBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockQuoteBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererBlockQuoteEnd
  {
    \markdownRendererBlockQuoteEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockQuoteEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.3 Bracketed Spans Attribute Context Renderers

The following macros are only produced, when the `bracketedSpans` option is enabled.

The `\markdownRendererBracketedSpanAttributeContextBegin` and `\markdownRendererBrac` macros represent the beginning and the end of a context in which the attributes of an inline bracketed span apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererBracketedSpanAttributeContextBegin
  {
    \markdownRendererBracketedSpanAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererBracketedSpanAttributeContextEnd
  {
    \markdownRendererBracketedSpanAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.4 Bullet List Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererUlBegin
  {
    \markdownRendererUlBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
```

```
  \markdownRendererUlBeginTight
  {
    \markdownRendererUlBeginTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulBeginTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererUlItem
  {
    \markdownRendererUlItemPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulItem }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererUlItemEnd
  {
    \markdownRendererUlItemEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulItemEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererUlEnd 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererUlEnd
  {
    \markdownRendererUlEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererUlEndTight
  {
    \markdownRendererUlEndTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.5.5 Citation Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererCite
  {
    \markdownRendererCitePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { cite }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { cite }
  { 1 }
\ExplSyntaxOff
```

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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererTextCite
  {
    \markdownRendererTextCitePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { textCite }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { textCite }
  { 1 }
\ExplSyntaxOff
```

2.2.5.6 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInputVerbatim
  {
    \markdownRendererInputVerbatimPrototype
  }
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { inputVerbatim }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputVerbatim }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives three arguments that correspond to the filename of a file containing the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInputFencedCode
  {
    \markdownRendererInputFencedCodePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputFencedCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputFencedCode }
  { 3 }
\ExplSyntaxOff
```

2.2.5.7 Code Span Renderer

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererCodeSpan
  {
    \markdownRendererCodeSpanPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpan }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpan }
  { 1 }
\ExplSyntaxOff
```

2.2.5.8 Code Span Attribute Context Renderers

The following macros are only produced, when the `inlineCodeAttributes` option is enabled.

The `\markdownRendererCodeSpanAttributeContextBegin` and `\markdownRendererCodeSpanA` macros represent the beginning and the end of a context in which the attributes of an inline code span apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererCodeSpanAttributeContextBegin
  {
    \markdownRendererCodeSpanAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpanAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpanAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererCodeSpanAttributeContextEnd
  {
    \markdownRendererCodeSpanAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpanAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpanAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.9 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererContentBlock
  {
    \markdownRendererContentBlockPrototype
  }
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { contentBlock }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlock }
  { 4 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererContentBlockOnlineImage
  {
    \markdownRendererContentBlockOnlineImagePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockOnlineImage }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockOnlineImage }
  { 4 }
\ExplSyntaxOff
```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension s. If any `markdown-languages.json` file found by kpathsea[32] contains a record (k, v), then a non-online-image content block with the filename extension s, s`:lower()` $= k$ is considered to be in a known programming language v. The macro receives five arguments: the local file name extension s cast to the lower case, the language v, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place place a `markdown-languages.json` file inside your working directory or inside your local TEX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [5] is a good starting point.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
```

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

```
  \markdownRendererContentBlockCode
  {
    \markdownRendererContentBlockCodePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

2.2.5.10 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlBegin
  {
    \markdownRendererDlBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlBeginTight
  {
    \markdownRendererDlBeginTightPrototype
  }
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { dlBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlBeginTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlItem
  {
    \markdownRendererDlItemPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlItem }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlItemEnd
  {
    \markdownRendererDlItemEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlItemEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlDefinitionBegin
```

```
  {
    \markdownRendererDlDefinitionBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlDefinitionEnd
  {
    \markdownRendererDlDefinitionEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlEnd
  {
    \markdownRendererDlEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDlEndTight
  {
    \markdownRendererDlEndTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.5.11 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererEllipsis
  {
    \markdownRendererEllipsisPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ellipsis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

2.2.5.12 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
```

```
  \markdownRendererEmphasis
  {
    \markdownRendererEmphasisPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { emphasis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { emphasis }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererStrongEmphasis
  {
    \markdownRendererStrongEmphasisPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { strongEmphasis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { strongEmphasis }
  { 1 }
\ExplSyntaxOff
```

2.2.5.13 Fenced Code Attribute Context Renderers

The following macros are only produced, when the `fencedCode` and `fencedCodeAttributes` options are enabled.

The `\markdownRendererFencedCodeAttributeContextBegin` and `\markdownRendererFencedC` macros represent the beginning and the end of a context in which the attributes of a fenced code apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFencedCodeAttributeContextBegin
  {
    \markdownRendererFencedCodeAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
```

```
  { fencedCodeAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererFencedCodeAttributeContextEnd
  {
    \markdownRendererFencedCodeAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedCodeAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.14 Fenced Div Attribute Context Renderers

The following macros are only produced, when the `fencedDiv` option is enabled.

The `\markdownRendererFencedDivAttributeContextBegin` and `\markdownRendererFencedDi` macros represent the beginning and the end of a context in which the attributes of a div apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFencedDivAttributeContextBegin
  {
    \markdownRendererFencedDivAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedDivAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedDivAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererFencedDivAttributeContextEnd
  {
    \markdownRendererFencedDivAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedDivAttributeContextEnd }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { fencedDivAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.15 Header Attribute Context Renderers

The following macros are only produced, when the `autoIdentifiers`, `gfmAutoIdentifiers`, or `headerAttributes` options are enabled.

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeaderAttributeContextBegin
  {
    \markdownRendererHeaderAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererHeaderAttributeContextEnd
  {
    \markdownRendererHeaderAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.16 Heading Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingOne
```

```
  {
    \markdownRendererHeadingOnePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingOne }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingOne }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingTwo
  {
    \markdownRendererHeadingTwoPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingTwo }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingTwo }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingThree
  {
    \markdownRendererHeadingThreePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingThree }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingThree }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingFour
  {
    \markdownRendererHeadingFourPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingFour }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingFour }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingFive
  {
    \markdownRendererHeadingFivePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingFive }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingFive }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHeadingSix
  {
    \markdownRendererHeadingSixPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingSix }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { headingSix }
  { 1 }
\ExplSyntaxOff
```

2.2.5.17 Inline HTML Comment Renderer

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlComment
  {
    \markdownRendererInlineHtmlCommentPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlComment }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlComment }
  { 1 }
\ExplSyntaxOff
```

2.2.5.18 HTML Tag and Element Renderers

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

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlTag
  {
    \markdownRendererInlineHtmlTagPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlTag }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { inlineHtmlTag }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInputBlockHtmlElement
  {
    \markdownRendererInputBlockHtmlElementPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlElement }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlElement }
  { 1 }
\ExplSyntaxOff
```

2.2.5.19 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererImage
  {
    \markdownRendererImagePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { image }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { image }
  { 4 }
\ExplSyntaxOff
```

2.2.5.20 Image Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererImageAttributeContextBegin` and `\markdownRendererImageAttribu` macros represent the beginning and the end of a context in which the attributes of an image apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
```

```
  \markdownRendererImageAttributeContextBegin
  {
    \markdownRendererImageAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { imageAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { imageAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererImageAttributeContextEnd
  {
    \markdownRendererImageAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { imageAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { imageAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.21 Interblock Separator Renderers

The `\markdownRendererInterblockSeparator` macro represents an interblock separator between two markdown block elements. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInterblockSeparator
  {
    \markdownRendererInterblockSeparatorPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { interblockSeparator }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { interblockSeparator }
  { 0 }
\ExplSyntaxOff
```

Users can use more than one blank line to delimit two block to indicate the end of a series of blocks that make up a logical paragraph. This produces a paragraph

separator instead of an interblock separator. Between some blocks, such as markdown paragraphs, a paragraph separator is always produced.

The \markdownRendererParagraphSeparator macro represents a paragraph separator. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererParagraphSeparator
  {
    \markdownRendererParagraphSeparatorPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { paragraphSeparator }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { paragraphSeparator }
  { 0 }
\ExplSyntaxOff
```

2.2.5.22 Line Block Renderers

The following macros are only produced, when the lineBlocks option is enabled.

The \markdownRendererLineBlockBegin and \markdownRendererLineBlockEnd macros represent the beginning and the end of a line block. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererLineBlockBegin
  {
    \markdownRendererLineBlockBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBlockBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBlockBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererLineBlockEnd
  {
    \markdownRendererLineBlockEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBlockEnd }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBlockEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.23 Line Break Renderers

The \markdownRendererSoftLineBreak macro represents a soft line break. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererSoftLineBreak
  {
    \markdownRendererSoftLineBreakPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { softLineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { softLineBreak }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererHardLineBreak macro represents a hard line break. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererHardLineBreak
  {
    \markdownRendererHardLineBreakPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { hardLineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { hardLineBreak }
  { 0 }
\ExplSyntaxOff
```

2.2.5.24 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.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererLink
  {
    \markdownRendererLinkPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { link }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { link }
  { 4 }
\ExplSyntaxOff
```

2.2.5.25 Link Attribute Context Renderers

The following macros are only produced, when the `linkAttributes` option is enabled.

The `\markdownRendererLinkAttributeContextBegin` and `\markdownRendererLinkAttributeC` macros represent the beginning and the end of a context in which the attributes of a hyperlink apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererLinkAttributeContextBegin
  {
    \markdownRendererLinkAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { linkAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { linkAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererLinkAttributeContextEnd
  {
    \markdownRendererLinkAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { linkAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { linkAttributeContextEnd }
  { 0 }
```

```
\ExplSyntaxOff
```

2.2.5.26 Marked Text Renderer

The following macro is only produced, when the `mark` option is enabled.

The `\markdownRendererMark` macro represents a span of marked or highlighted text. The macro receives a single argument that corresponds to the marked text.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererMark
  {
    \markdownRendererMarkPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { mark }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { mark }
  { 1 }
\ExplSyntaxOff
```

2.2.5.27 Markdown Document Renderers

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

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererDocumentBegin
  {
    \markdownRendererDocumentBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererDocumentEnd
```

```
  {
    \markdownRendererDocumentEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.28 Non-Breaking Space Renderer

The `\markdownRendererNbsp` macro represents a non-breaking space.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererNbsp
  {
    \markdownRendererNbspPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { nbsp }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { nbsp }
  { 0 }
\ExplSyntaxOff
```

2.2.5.29 Note Renderer

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

```
\def\markdownRendererNote{%
  \markdownRendererNotePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { note }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { note }
  { 1 }
\ExplSyntaxOff
```

2.2.5.30 Ordered List Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlBegin
  {
    \markdownRendererOlBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlBeginTight
  {
    \markdownRendererOlBeginTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olBeginTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlBegin
  {
    \markdownRendererFancyOlBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBegin }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlBeginTight
  {
    \markdownRendererFancyOlBeginTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBeginTight }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlItem
  {
    \markdownRendererOlItemPrototype
  }
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { olItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItem }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlItemEnd
  {
    \markdownRendererOlItemEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItemEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlItemWithNumber
  {
    \markdownRendererOlItemWithNumberPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olItemWithNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlItem
  {
    \markdownRendererFancyOlItemPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItem }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlItemEnd
  {
    \markdownRendererFancyOlItemEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlItemWithNumber
  {
    \markdownRendererFancyOlItemWithNumberPrototype
  }
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemWithNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlEnd
  {
    \markdownRendererOlEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererOlEndTight
  {
    \markdownRendererOlEndTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olEndTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlEnd
  {
    \markdownRendererFancyOlEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererFancyOlEndTight
  {
    \markdownRendererFancyOlEndTightPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.5.31 Raw Content Renderers

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

```
\ExplSyntaxOn
```

```
\cs_gset_protected:Npn
  \markdownRendererInputRawInline
  {
    \markdownRendererInputRawInlinePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputRawInline }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputRawInline }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInputRawBlock
  {
    \markdownRendererInputRawBlockPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputRawBlock }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputRawBlock }
  { 2 }
\ExplSyntaxOff
```

2.2.5.32 Section Renderers

The `\markdownRendererSectionBegin` and `\markdownRendererSectionEnd` macros represent the beginning and the end of a section based on headings.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererSectionBegin
  {
    \markdownRendererSectionBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { sectionBegin }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { sectionBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererSectionEnd
  {
    \markdownRendererSectionEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { sectionEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { sectionEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.33 Replacement Character Renderers

The `\markdownRendererReplacementCharacter` macro represents the U+0000 and U+FFFD Unicode characters. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererReplacementCharacter
  {
    \markdownRendererReplacementCharacterPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { replacementCharacter }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { replacementCharacter }
  { 0 }
\ExplSyntaxOff
```

2.2.5.34 Special Character Renderers

The following macros replace any special plain TEX characters, including the active pipe character (`|`) of ConTEXt, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererLeftBrace
  {
    \markdownRendererLeftBracePrototype
  }
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { leftBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { leftBrace }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererRightBrace
  {
    \markdownRendererRightBracePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { rightBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { rightBrace }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererDollarSign
  {
    \markdownRendererDollarSignPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dollarSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dollarSign }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererPercentSign
  {
    \markdownRendererPercentSignPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { percentSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { percentSign }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererAmpersand
  {
    \markdownRendererAmpersandPrototype
```

```
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ampersand }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ampersand }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererUnderscore
  {
    \markdownRendererUnderscorePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { underscore }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { underscore }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererHash
  {
    \markdownRendererHashPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { hash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { hash }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererCircumflex
  {
    \markdownRendererCircumflexPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { circumflex }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { circumflex }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererBackslash
  {
```

```
    \markdownRendererBackslashPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { backslash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { backslash }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererTilde
  {
    \markdownRendererTildePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tilde }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tilde }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererPipe
  {
    \markdownRendererPipePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { pipe }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { pipe }
  { 0 }
\ExplSyntaxOff
```

2.2.5.35 Strike-Through Renderer

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererStrikeThrough
  {
    \markdownRendererStrikeThroughPrototype
  }
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { strikeThrough }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { strikeThrough }
  { 1 }
\ExplSyntaxOff
```

2.2.5.36 Subscript Renderer

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererSubscript
  {
    \markdownRendererSubscriptPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { subscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { subscript }
  { 1 }
```

2.2.5.37 Superscript Renderer

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

```
\cs_gset_protected:Npn
  \markdownRendererSuperscript
  {
    \markdownRendererSuperscriptPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { superscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { superscript }
  { 1 }
\ExplSyntaxOff
```

2.2.5.38 Table Attribute Context Renderers

The following macros are only produced, when the `tableCaptions` and `tableAttributes` options are enabled.

The `\markdownRendererTableAttributeContextBegin` and `\markdownRendererTableAttribu` macros represent the beginning and the end of a context in which the attributes of a table apply. The macros receive no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererTableAttributeContextBegin
  {
    \markdownRendererTableAttributeContextBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextBegin }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererTableAttributeContextEnd
  {
    \markdownRendererTableAttributeContextEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.39 Table Renderer

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

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

- `r` – The corresponding column is right-aligned.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererTable
  {
    \markdownRendererTablePrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { table }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { table }
  { 3 }
\ExplSyntaxOff
```

2.2.5.40 TEX Math Renderers

The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display TEX math. Both macros receive a single argument that corresponds to the TEX math content. These macros will only be produced, when the `texMathDollars`, `texMathSingleBackslash`, or `texMathDoubleBackslash` option are enabled.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInlineMath
  {
    \markdownRendererInlineMathPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineMath }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineMath }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererDisplayMath
  {
    \markdownRendererDisplayMathPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { displayMath }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

```
  { displayMath }
  { 1 }
\ExplSyntaxOff
```

2.2.5.41 Thematic Break Renderer

The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererThematicBreak
  {
    \markdownRendererThematicBreakPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { thematicBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { thematicBreak }
  { 0 }
\ExplSyntaxOff
```

2.2.5.42 Tickbox Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererTickedBox
  {
    \markdownRendererTickedBoxPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tickedBox }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererHalfTickedBox
  {
```

```
    \markdownRendererHalfTickedBoxPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { halfTickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { halfTickedBox }
  { 0 }
\cs_gset_protected:Npn
  \markdownRendererUntickedBox
  {
    \markdownRendererUntickedBoxPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { untickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { untickedBox }
  { 0 }
\ExplSyntaxOff
```

2.2.5.43 Warning and Error Renderers

The `\markdownRendererWarning` and `\markdownRendererError` macros represent warnings and errors produced by the markdown parser. Both macros receive four parameters:

1. The fully escaped text of the warning or error that can be directly typeset
2. The raw text of the warning or error that can be used outside typesetting for e.g. logging the warning or error.
3. The fully escaped text with more details about the warning or error that can be directly typeset. Can be empty, unlike the first two parameters.
4. The raw text with more details about the warning or error that can be used outside typesetting for e.g. logging the warning or error. Can be empty, unlike the first two parameters.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererWarning
  {
    \markdownRendererWarningPrototype
  }
\cs_gset_protected:Npn
  \markdownRendererError
  {
```

```
    \markdownRendererErrorPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { warning }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { warning }
  { 4 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { error }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { error }
  { 4 }
\ExplSyntaxOff
```

2.2.5.44 YAML Metadata Renderers

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataBegin
  {
    \markdownRendererJekyllDataBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataEnd
  {
    \markdownRendererJekyllDataEndPrototype
  }
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataMappingBegin
  {
    \markdownRendererJekyllDataMappingBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingBegin }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataMappingEnd
  {
    \markdownRendererJekyllDataMappingEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataSequenceBegin
  {
    \markdownRendererJekyllDataSequenceBeginPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceBegin }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataSequenceEnd
  {
    \markdownRendererJekyllDataSequenceEndPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
```

```
  \markdownRendererJekyllDataBoolean
  {
    \markdownRendererJekyllDataBooleanPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBoolean }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBoolean }
  { 2 }
\ExplSyntaxOff
```

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

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataNumber
  {
    \markdownRendererJekyllDataNumberPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataNumber }
  { 2 }
\ExplSyntaxOff
```

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

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataT` receives the scalar value after all markdown markup and special TEX characters in the string have been replaced by TEX macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicSt` macro is more appropriate for texts that are supposed to be typeset with TEX, such as document titles, author names, or exam questions, the

\markdownRendererJekyllDataProgrammaticString macro is more appropriate for identifiers and other programmatic text that won't be typeset by TEX.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataTypographicString
  {
    \markdownRendererJekyllDataTypographicStringPrototype
  }
\cs_gset_protected:Npn
  \markdownRendererJekyllDataProgrammaticString
  {
    \markdownRendererJekyllDataProgrammaticStringPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataTypographicString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataTypographicString }
  { 2 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataProgrammaticString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataProgrammaticString }
  { 2 }
\ExplSyntaxOff
```

Before Markdown 3.7.0, the \markdownRendererJekyllDataTypographicString macro was named \markdownRendererJekyllDataString and the \markdownRendererJekyllData macro was not produced. The \markdownRendererJekyllDataString has been deprecated and will be removed in Markdown 4.0.0.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererJekyllDataTypographicString
  {
    \cs_if_exist:NTF
      \markdownRendererJekyllDataString
      {
        \@@_if_option:nTF
          { experimental }
          {
            \markdownError
              {
                The~jekyllDataString~renderer~has~been~deprecated,~
```

```
                to~be~removed~in~Markdown~4.0.0
              }
          }
          {
            \markdownWarning
              {
                The~jekyllDataString~renderer~has~been~deprecated,~
                to~be~removed~in~Markdown~4.0.0
              }
            \markdownRendererJekyllDataString
          }
      }
      {
        \cs_if_exist:NTF
          \markdownRendererJekyllDataStringPrototype
          {
            \@@_if_option:nTF
              { experimental }
              {
                \markdownError
                  {
                    The~jekyllDataString~renderer~prototype~
                    has~been~deprecated,~
                    to~be~removed~in~Markdown~4.0.0
                  }
              }
              {
                \markdownWarning
                  {
                    The~jekyllDataString~renderer~prototype~
                    has~been~deprecated,~
                    to~be~removed~in~Markdown~4.0.0
                  }
                \markdownRendererJekyllDataStringPrototype
              }
          }
          {
            \markdownRendererJekyllDataTypographicStringPrototype
          }
      }
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataString }
```

```
  { 2 }
\ExplSyntaxOff
```

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

See also Section 2.2.6.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererJekyllDataEmpty
  {
    \markdownRendererJekyllDataEmptyPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEmpty }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEmpty }
  { 1 }
\ExplSyntaxOff
```

2.2.5.45 Generating Plain TEX Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain TEX macros for token renderers. Furthermore, the `\markdownSetup` macro also accepts the `renderers` and `unprotectedRenderers` keys. The value for these keys must be a list of key–values, where the keys correspond to the markdown token renderer macros and the values are new definitions of these token renderers.

Whereas the key `renderers` defines protected functions, which are usually preferable for typesetting, the key `unprotectedRenderers` defines unprotected functions, which are easier to expand and may be preferable for programming.

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderers:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer:n
          { ##1 }
      }
  }
\cs_new:Nn \@@_define_renderer:n
  {
```

```
    \@@_renderer_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_define_renderer:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
\tl_new:N
  \l_@@_renderer_definition_tl
\bool_new:N
  \g_@@_appending_renderer_bool
\bool_new:N
  \g_@@_unprotected_renderer_bool
\cs_new:Nn \@@_define_renderer:nNn
  {
    \keys_define:nn
      { markdown/options/renderers }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_@@_renderer_definition_tl
            { ##1 }
          \regex_replace_all:nnN
            { \cP\#0 }
            { #1 }
            \l_@@_renderer_definition_tl
          \bool_if:NT
            \g_@@_appending_renderer_bool
            {
```

```
              \@@_tl_set_from_cs:NNn
                \l_tmpa_tl
                #2
                { #3 }
              \tl_put_left:NV
                \l_@@_renderer_definition_tl
                \l_tmpa_tl
            }
          \bool_if:NTF
            \g_@@_unprotected_renderer_bool
            {
              \tl_set:Nn
                \l_tmpa_tl
                { \cs_set:Npn }
            }
            {
              \tl_set:Nn
                \l_tmpa_tl
                { \cs_set_protected:Npn }
            }
          \exp_last_unbraced:NNV
            \cs_generate_from_arg_count:NNnV
            #2
            \l_tmpa_tl
            { #3 }
            \l_@@_renderer_definition_tl
        },
    }
```

If the token renderer macro has been deprecated, we undefine it.

The `\markdownRendererJekyllDataString` macro has been deprecated and will be removed in Markdown 4.0.0.

```
  \str_if_eq:nnT
    { #1 }
    { jekyllDataString }
    {
      \cs_undefine:N
        #2
    }
}
```

We define the function `\@@_tl_set_from_cs:NNn` [12]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```
\cs_new_protected:Nn
  \@@_tl_set_from_cs:NNn
```

```
  {
    \tl_set:Nn
      \l_tmpa_tl
      { #2 }
    \int_step_inline:nn
      { #3 }
      {
        \exp_args:NNc
          \tl_put_right:Nn
          \l_tmpa_tl
          { @@_tl_set_from_cs_parameter_ ##1 }
      }
    \exp_args:NNV
      \tl_set:No
      \l_tmpb_tl
      \l_tmpa_tl
    \regex_replace_all:nnN
      { \cP. }
      { \0\0 }
      \l_tmpb_tl
    \int_step_inline:nn
      { #3 }
      {
        \regex_replace_all:nnN
          { \c { @@_tl_set_from_cs_parameter_ ##1 } }
          { \cP\# ##1 }
          \l_tmpb_tl
      }
    \tl_set:NV
      #1
      \l_tmpb_tl
  }
\cs_generate_variant:Nn
  \@@_define_renderer:nNn
  { ncV }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { NNnV }
\cs_generate_variant:Nn
  \tl_put_left:Nn
  { Nv }
\keys_define:nn
  { markdown/options }
  {
    renderers .code:n = {
      \bool_gset_false:N
        \g_@@_unprotected_renderer_bool
```

```
      \keys_set:nn
        { markdown/options/renderers }
        { #1 }
    },
    unprotectedRenderers .code:n = {
      \bool_gset_true:N
        \g_@@_unprotected_renderer_bool
      \keys_set:nn
        { markdown/options/renderers }
        { #1 }
    },
  }
```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

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

```
\tl_new:N
  \l_@@_renderer_glob_definition_tl
\seq_new:N
  \l_@@_renderer_glob_results_seq
\regex_const:Nn
  \c_@@_appending_key_regex
  { \s*+$ }
\keys_define:nn
  { markdown/options/renderers }
  {
    unknown .code:n = {
```

Besides defining renderers at once, we can also define them incrementally using the appending operator (`+=`). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  renderers = {
    % Start with empty renderers.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
```

```
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
3412        \regex_match:NVTF
3413          \c_@@_appending_key_regex
3414          \l_keys_key_str
3415          {
3416            \bool_gset_true:N
3417              \g_@@_appending_renderer_bool
3418            \tl_set:NV
3419              \l_tmpa_tl
3420              \l_keys_key_str
3421            \regex_replace_once:NnN
3422              \c_@@_appending_key_regex
3423              { }
3424              \l_tmpa_tl
3425            \tl_set:Nx
3426              \l_tmpb_tl
3427              { { \l_tmpa_tl } = }
3428            \tl_put_right:Nn
3429              \l_tmpb_tl
3430              { { #1 } }
3431            \keys_set:nV
3432              { markdown/options/renderers }
3433              \l_tmpb_tl
3434            \bool_gset_false:N
3435              \g_@@_appending_renderer_bool
3436          }
```

In addition to exact token renderer names, we also support wildcards (`*`) and enumerations (`|`) that match multiple token renderer names:

```
\markdownSetup{
  renderers = {
    heading* = {{\bf #1}},    % Render headings using the bold face.
    jekyllData(String|Number) = {%  % Render YAML string and numbers
      {\it #2}%                                    % using italics.
    },
  }
}
```

Wildcards and enumerations can be combined:

```
\markdownSetup{
  renderers = {
    *lItem(|End) = {"},           % Quote ordered/bullet list items.
  }
}
```

To determine the current token renderer, you can use the pseudo-parameter `#0`:

```
\markdownSetup{
  renderers = {
    heading* = {#0: #1},      % Render headings as the renderer name
  }                                  % followed by the heading text.
}
```

```
3437          {
3438            \@@_glob_seq:VnN
3439              \l_keys_key_str
3440              { g_@@_renderers_seq }
3441              \l_@@_renderer_glob_results_seq
3442            \seq_if_empty:NTF
3443              \l_@@_renderer_glob_results_seq
3444              {
3445                \msg_error:nnV
3446                  { markdown }
3447                  { undefined-renderer }
3448                  \l_keys_key_str
3449              }
3450              {
3451                \tl_set:Nn
```

```
                \l_@@_renderer_glob_definition_tl
                { \exp_not:n { #1 } }
              \seq_map_inline:Nn
                \l_@@_renderer_glob_results_seq
                {
                  \tl_set:Nn
                    \l_tmpa_tl
                    { { ##1 } = }
                  \tl_put_right:Nx
                    \l_tmpa_tl
                    { { \l_@@_renderer_glob_definition_tl } }
                  \keys_set:nV
                    { markdown/options/renderers }
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer }
  {
    Renderer~#1~is~undefined.
  }
\cs_generate_variant:Nn
  \@@_glob_seq:nnN
  { VnN }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { cNVV }
\cs_generate_variant:Nn
  \msg_error:nnn
  { nnV }
\prg_generate_conditional_variant:Nnn
  \regex_match:Nn
  { NV }
  { TF }
\prop_new:N
  \g_@@_glob_cache_prop
\tl_new:N
  \l_@@_current_glob_tl
\cs_new:Nn
  \@@_glob_seq:nnN
  {
    \tl_set:Nn
      \l_@@_current_glob_tl
```

```
      { ^ #1 $ }
    \prop_get:NeNTF
      \g_@@_glob_cache_prop
      { #2 / \l_@@_current_glob_tl }
      \l_tmpa_clist
      {
        \seq_set_from_clist:NN
          #3
          \l_tmpa_clist
      }
      {
        \seq_clear:N
          #3
        \regex_replace_all:nnN
          { \* }
          { .* }
          \l_@@_current_glob_tl
        \regex_set:NV
          \l_tmpa_regex
          \l_@@_current_glob_tl
        \seq_map_inline:cn
          { #2 }
          {
            \regex_match:NnT
              \l_tmpa_regex
              { ##1 }
              {
                \seq_put_right:Nn
                  #3
                  { ##1 }
              }
          }
        \clist_set_from_seq:NN
          \l_tmpa_clist
          #3
        \prop_gput:NeV
          \g_@@_glob_cache_prop
          { #2 / \l_@@_current_glob_tl }
          \l_tmpa_clist
      }
  }
\cs_generate_variant:Nn
  \regex_set:Nn
  { NV }
\cs_generate_variant:Nn
  \prop_gput:Nnn
  { NeV }
```

If plain TEX is the top layer, we use the `\@@_define_renderers:` macro to define plain TEX token renderer macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_renderers:
  }
\ExplSyntaxOff
```

2.2.6 Token Renderer Prototypes

2.2.6.1 YAML Metadata Renderer Prototypes

For simple YAML metadata, a simple high-level interface is provided that can be programmed by setting the expl3 key–values [2] for the module `markdown/jekyllData`.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  { }
\ExplSyntaxOff
```

The option `jekyllDataRenderers`=⟨*key–values*⟩ can be used to set the ⟨*key–values*⟩ for the module `markdown/jekyllData` without using the expl3 syntax.

```
\ExplSyntaxOn
\@@_with_various_cases:nn
  { jekyllDataRenderers }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_tmpa_tl
            { ##1 }
```

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

```
          \tl_replace_all:NnV
            \l_tmpa_tl
            { / }
            \c_backslash_str
          \keys_set:nV
```

```
            { markdown/options/jekyll-data-renderers }
            \l_tmpa_tl
        },
      }
  }
\keys_define:nn
  { markdown/options/jekyll-data-renderers }
  {
    unknown .code:n = {
      \tl_set_eq:NN
        \l_tmpa_tl
        \l_keys_key_str
      \tl_replace_all:NVn
        \l_tmpa_tl
        \c_backslash_str
        { / }
      \tl_put_right:Nn
        \l_tmpa_tl
        {
          .code:n = { #1 }
        }
      \keys_define:nV
        { markdown/jekyllData }
        \l_tmpa_tl
    }
  }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\ExplSyntaxOff
```

For complex YAML metadata, the option `jekyllDataKeyValue`=⟨*module*⟩ [13] can be used to route the processing of all YAML metadata in the current TEX group to the key–values from ⟨*module*⟩.

2.2.6.2 Generating Plain TEX Token Renderer Prototype Macros and Key-Values

We define the command `\@@_define_renderer_prototypes:` that defines plain TEX macros for token renderer prototypes. Furthermore, the `\markdownSetup` macro also accepts the `rendererPrototypes` and `unprotectedRendererPrototypes` keys. The value for these keys must be a list of key–values, where the keys correspond to the markdown token renderer prototype macros and the values are new definitions of these token renderer prototypes.

Whereas the key `rendererPrototypes` defines protected functions, which are usually preferable for typesetting, the key `unprotectedRendererPrototypes` de-

fines unprotected functions, which are easier to expand and may be preferable for programming.

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderer_prototypes:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer_prototype:n
          { ##1 }
      }
  }
\cs_new:Nn \@@_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_define_renderer_prototype:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
        Prototype
      }
  }
\tl_new:N
  \l_@@_renderer_prototype_definition_tl
\bool_new:N
  \g_@@_appending_renderer_prototype_bool
\bool_new:N
  \g_@@_unprotected_renderer_prototype_bool
\cs_new:Nn \@@_define_renderer_prototype:nNn
```

```
{
  \keys_define:nn
    { markdown/options/renderer-prototypes }
    {
      #1 .code:n = {
        \tl_set:Nn
          \l_@@_renderer_prototype_definition_tl
          { ##1 }
        \regex_replace_all:nnN
          { \cP\#0 }
          { #1 }
          \l_@@_renderer_prototype_definition_tl
        \bool_if:NT
          \g_@@_appending_renderer_prototype_bool
          {
            \@@_tl_set_from_cs:NNn
              \l_tmpa_tl
              #2
              { #3 }
            \tl_put_left:NV
              \l_@@_renderer_prototype_definition_tl
              \l_tmpa_tl
          }
        \bool_if:NTF
          \g_@@_unprotected_renderer_prototype_bool
          {
            \tl_set:Nn
              \l_tmpa_tl
              { \cs_set:Npn }
          }
          {
            \tl_set:Nn
              \l_tmpa_tl
              { \cs_set_protected:Npn }
          }
        \exp_last_unbraced:NNV
          \cs_generate_from_arg_count:NNnV
          #2
          \l_tmpa_tl
          { #3 }
          \l_@@_renderer_prototype_definition_tl
      },
    }
```

Unless the token renderer prototype macro has already been defined or unless, it has been deprecated, we provide an empty definition.

The \markdownRendererJekyllDataStringPrototype macro has been deprecated and will be removed in Markdown 4.0.0.

```
    \str_if_eq:nnF
      { #1 }
      { jekyllDataString }
      {
        \cs_if_free:NT
          #2
          {
            \cs_generate_from_arg_count:NNnn
              #2
              \cs_gset_protected:Npn
              { #3 }
              { }
          }
      }
  }
\cs_generate_variant:Nn
  \@@_define_renderer_prototype:nNn
  { ncV }
```

The following example code showcases a possible configuration of the \markdownRendererImageProto and \markdownRendererCodeSpanPrototype token renderer prototype macros.

```
\markdownSetup{
  rendererPrototypes = {
    image = {\pdfximage{#2}},    % Embed PDF images in the document.
    codeSpan = {{\tt #1}},     % Render inline code using monospace.
  }
}
```

```
\keys_define:nn
  { markdown/options/renderer-prototypes }
  {
    unknown .code:n = {
```

Besides defining renderer prototypes at once, we can also define them incrementally using the appending operator (+=). This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = {},
    attributeClassName = {},
```

```
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
3714        \regex_match:NVTF
3715          \c_@@_appending_key_regex
3716          \l_keys_key_str
3717          {
3718            \bool_gset_true:N
3719              \g_@@_appending_renderer_prototype_bool
3720            \tl_set:NV
3721              \l_tmpa_tl
3722              \l_keys_key_str
3723            \regex_replace_once:NnN
3724              \c_@@_appending_key_regex
3725              { }
3726              \l_tmpa_tl
3727            \tl_set:Nx
3728              \l_tmpb_tl
3729              { { \l_tmpa_tl } = }
3730            \tl_put_right:Nn
3731              \l_tmpb_tl
3732              { { #1 } }
3733            \keys_set:nV
3734              { markdown/options/renderer-prototypes }
3735              \l_tmpb_tl
3736            \bool_gset_false:N
3737              \g_@@_appending_renderer_prototype_bool
```

```
3738          }
```

In addition to exact token renderer prototype names, we also support wildcards (`*`) and enumerations (`|`) that match multiple token renderer prototype names:

```
\markdownSetup{
  rendererPrototypes = {
    heading* = {{\bf #1}},    % Render headings using the bold face.
    jekyllData(String|Number) = {   % Render YAML string and numbers
      {\it #2}%                                      % using italics.
    },
  }
}
```

Wildcards and enumerations can be combined:

```
\markdownSetup{
  rendererPrototypes = {
    *lItem(|End) = {"},           % Quote ordered/bullet list items.
  }
}
```

To determine the current token renderer prototype, you can use the pseudo-parameter `#0`:

```
\markdownSetup{
  rendererPrototypes = {
    heading* = {#0: #1}, % Render headings as the renderer prototype
  }                                % name followed by the heading text.
}
```

```
3739          {
3740            \@@_glob_seq:VnN
3741              \l_keys_key_str
3742              { g_@@_renderers_seq }
3743              \l_@@_renderer_glob_results_seq
3744            \seq_if_empty:NTF
3745              \l_@@_renderer_glob_results_seq
3746              {
3747                \msg_error:nnV
3748                  { markdown }
3749                  { undefined-renderer-prototype }
3750                  \l_keys_key_str
```

```
            }
            {
              \tl_set:Nn
                \l_@@_renderer_glob_definition_tl
                { \exp_not:n { #1 } }
              \seq_map_inline:Nn
                \l_@@_renderer_glob_results_seq
                {
                  \tl_set:Nn
                    \l_tmpa_tl
                    { { ##1 } = }
                  \tl_put_right:Nx
                    \l_tmpa_tl
                    { { \l_@@_renderer_glob_definition_tl } }
                  \keys_set:nV
                    { markdown/options/renderer-prototypes }
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer-prototype }
  {
    Renderer~prototype~#1~is~undefined.
  }
\@@_with_various_cases:nn
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \bool_gset_false:N
            \g_@@_unprotected_renderer_prototype_bool
          \keys_set:nn
            { markdown/options/renderer-prototypes }
            { ##1 }
        },
      }
  }
\@@_with_various_cases:nn
  { unprotectedRendererPrototypes }
  {
    \keys_define:nn
```

```
      { markdown/options }
      {
        #1 .code:n = {
          \bool_gset_true:N
            \g_@@_unprotected_renderer_prototype_bool
          \keys_set:nn
            { markdown/options/renderer-prototypes }
            { ##1 }
        },
      }
  }
```

If plain TEX is the top layer, we use the `\@@_define_renderer_prototypes:` macro to define plain TEX token renderer prototype macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

2.2.7 Logging Facilities

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros perform logging for the Markdown package. Their first argument specifies the text of the info, warning, or error message. The `\markdownError` macro receives a second argument that provides a help text. You may redefine these macros to redirect and process the info, warning, and error messages.

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros have been deprecated and will be removed in the next major version of the Markdown package.

2.2.8 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*.

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

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will terminate the markdown input when the plain TEX special characters have had

their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
\let\markdownReadAndConvert\relax
\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*.

```
  \catcode`\|=0\catcode`\\=12%
  |gdef|markdownBegin{%
    |markdownReadAndConvert{\markdownEnd}%
                           {|markdownEnd}}%
  |gdef|yamlBegin{%
    |begingroup
    |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
    |markdownReadAndConvert{\yamlEnd}%
                           {|yamlEnd}}%
|endgroup
```

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

The `code` key, which can be used to immediately expand and execute code.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/options }
  {
    code .code:n = { #1 },
  }
\ExplSyntaxOff
```

This can be especially useful in snippets.

2.3 LaTeX Interface

The LaTeX interface provides LaTeX environments for the typesetting of markdown input from within LaTeX, facilities for setting Lua, plain TeX, and LaTeX options 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).

To determine whether LaTeX is the top layer or if there are other layers above LaTeX, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that LaTeX is the top layer.

```
\ExplSyntaxOn
```

```
\tl_const:Nn \c_@@_option_layer_latex_tl { latex }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_latex_tl
  }
\ExplSyntaxOff
\input markdown/markdown
```

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

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

where ⟨*options*⟩ are the LaTeX interface options (see Section 2.3.3). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single LaTeX theme (see Section 2.3.4) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way LaTeX 2_ε parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown`, `markdown*`, and `yaml` LaTeX environments, and redefines the `\markinline`, `\markdownInput`, and `\yamlInput` commands.

2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` LaTeX environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TeX interface.

The `markdown*` environment has been deprecated and will be removed in the next major version of the Markdown package.

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

Furthermore, both environments accept LaTeX interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[smartEllipses]
_Hello_ **world** ...
\end{markdown}
\end{document}
```

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown*}{smartEllipses}
_Hello_ **world** ...
\end{markdown*}
\end{document}
```

You can't directly extend the `markdown` LaTeX environment by using it in other environments as follows:

```
\newenvironment{foo}%
               {code before \begin{markdown}[some, options]}%
               {\end{markdown} code after}
```

This is because the implementation looks for the literal string `\end{markdown}` to stop scanning the markdown text. However, you can work around this limitation by using the `\markdown` and `\markdownEnd` macros directly in the definition as follows:

```
\newenvironment{foo}%
               {code before \markdown[some, options]}%
               {\markdownEnd code after}
```

Specifically, the `\markdown` macro must appear at the end of the replacement before-text and must be followed by text that has not yet been ingested by TeX's input processor.

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` LaTeX environment.

Lastly, you can't nest the other environments. For example, the following definition would be incorrect:

```
\newenvironment{bar}{\begin{foo}}{\end{foo}}
```

In this example, you should use the `\markdown` macro directly in the definition of the environment `bar`:

```
\newenvironment{bar}{\markdown[some, options]}{\markdownEnd}
```

The `yaml` LaTeX environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
\newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts LaTeX interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example LaTeX code showcases the usage of the `yaml` environment:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{yaml}[smartEllipses]
title: _Hello_ **world** ...
author: John Doe
\end{yaml}
\end{document}
```

The above code has the same effect as the below code:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\begin{markdown}[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]
title: _Hello_ **world** ...
author: John Doe
\end{markdown}
\end{document}
```

You can't directly extend the `yaml` LaTeX environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro _must_ be used in or after the replacement after-text.

The \markinline macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain TEX. Unlike the \markinline macro provided by the plain TEX interface, this macro also accepts LATEX interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

2.3.1.2 Typesetting Markdown and YAML from external documents

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 2.3.3) 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}
```

The \yamlInput macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain TEX. Unlike the \yamlInput macro provided by the plain TEX interface, this macro also accepts LATEX interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

The following example LATEX code showcases the usage of the \yamlInput macro:

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

The above code has the same effect as the below code:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
```

```
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]{hello.yml}
\end{document}
```

2.3.2 Using LaTeX hooks with the Markdown package

LaTeX provides an intricate hook management system that allows users to insert extra material before and after certain TeX macros and LaTeX environments, among other things. [14, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before TeX commands and before/after LaTeX environments without restriction:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}
```

Processing the above example with LaTeX will produce the text "markdownfoo emphasis: _bar_ baz!/markdown", as expected.

However, using hooks to insert extra material after TeX commands only works for commands with a fixed number of parameters that don't use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```
\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}
```

```
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}
```

Processing the above example with LaTeX will produce the text "markdownfoo emphasis_bar_/emphasis baz!/markdown", as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

2.3.3 Options

The LaTeX options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` if the =⟨*value*⟩ part has been omitted.

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 2.2.5 and 2.2.6).

The LaTeX options may be specified when loading the LaTeX package, when using the `markdown*` LaTeX environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [15, Section 5.1], which supports the `finalizecache` and `frozencache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain TeX options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizecache` 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 `finalizecache` and `frozencache` package options in the future, so that they can become a standard interface for preparing LaTeX document sources for distribution.

```
\DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
\DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

2.3.3.2 Generating Plain TeX Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

If LaTeX is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain TeX option, token renderer, and token renderer prototype macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_latex_tl
  {
    \@@_define_option_commands_and_keyvals:
    \@@_define_renderers:
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

The following example LaTeX code showcases a possible configuration of plain TeX interface options `hybrid`, `smartEllipses`, and `cacheDir`.

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

2.3.4 Themes

In Section 2.2.3, we described the concept of themes. In LaTeX, we expand on the concept of themes by allowing a theme to be a full-blown LaTeX package. Specifically, the key–values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ load a LaTeX package named `markdowntheme`⟨*munged theme name*⟩`.sty` if it exists and a TeX document named `markdowntheme`⟨*munged theme name*⟩`.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex` *theme file* or the LaTeX-specific `.sty` theme file allows developers to have a single *theme file*, when the theme is small or the difference between TeX formats is unimportant, and

scale up to separate theme files native to different TEX formats for large multi-format themes, where different code is needed for different TEX formats. To enable code reuse, developers can load the `.tex` theme file from the `.sty` theme file using the `\markdownLoadPlainTeXTheme` macro.

If the LATEX option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown LATEX package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, the following code would first load the Markdown package, then the theme `witiko/example/foo`, and finally the theme `witiko/example/bar`:

```
\usepackage[
  import=witiko/example/foo,
  import=witiko/example/bar,
]{markdown}
```

```
\newif\ifmarkdownLaTeXLoaded
  \markdownLaTeXLoadedfalse
```

Due to limitations of LATEX, themes may not be loaded after the beginning of a LATEX document.

We also define the prop `\g_@@_latex_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```
\ExplSyntaxOn
\prop_new:N
  \g_@@_latex_built_in_themes_prop
\ExplSyntaxOff
```

Built-in LATEX themes provided with the Markdown package include:

witiko/markdown/defaults A LATEX theme with the default definitions of token renderer prototypes for plain TEX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
\AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

At the end of the LATEX module, we load the `witiko/markdown/defaults` LATEX theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_latex_tl
  {
```

```
    \use:c
      { ExplSyntaxOff }
    \AtEndOfPackage
      {
        \@@_if_option:nF
          { noDefaults }
          {
            \@@_if_option:nTF
              { experimental }
              {
                \@@_setup:n
                  { theme = witiko/markdown/defaults@experimental }
              }
              {
                \@@_setup:n
                  { theme = witiko/markdown/defaults }
              }
          }
      }
    \use:c
      { ExplSyntaxOn }
  }
\ExplSyntaxOff
```

Please, see Section 3.3.2 for implementation details of the built-in LaTeX themes.

2.4 ConTeXt Interface

To determine whether ConTeXt is the top layer or if there are other layers above ConTeXt, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that ConTeXt is the top layer.

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_context_tl { context }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_context_tl
  }
\ExplSyntaxOff
```

The ConTeXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTeXt and facilities for setting Lua, plain TeX, and

ConTEXt options used during the conversion from markdown to plain TEX. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

```
\writestatus{loading}{ConTeXt User Module / markdown}%
\startmodule[markdown]
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\input markdown/markdown
```

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

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

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

2.4.1 Typesetting Markdown and YAML

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TEX interface.

```
\let\startmarkdown\relax
\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.

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTEXt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TEX interface.

```
\let\startyaml\relax
\let\stopyaml\relax
```

You may prepend your own code to the `\startyaml` macro and append your own code to the `\stopyaml` macro to produce special effects before and after the YAML document.

The macros `\startyaml` and `\stopyaml` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTEXt code showcases the usage of the `\startyaml` and `\stopyaml` macros:

```
\usemodule[t][markdown]
\starttext
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

The above code has the same effect as the below code:

```
\usemodule[t][markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

2.4.1.2 Typesetting Markdown and YAML from external documents

The `\inputmarkdown` macro aliases the macro `\markdownInput` exposed by the plain TEX interface.

```
\let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example ConTEXt code showcases the usage of the `\inputmarkdown` macro:

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

```
\inputmarkdown[smartEllipses]{hello.md}
\stoptext
```

The above code has the same effect as the below code:

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

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain TEX interface.

```
\let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influence this YAML document.

The following example ConTEXt code showcases the usage of the `\inputyaml` macro:

```
\usemodule[t][markdown]
\starttext
\inputyaml[smartEllipses]{hello.yml}
\stoptext
```

The above code has the same effect as the below code:

```
\usemodule[t][markdown]
\starttext
\setupyaml[smartEllipses]
\inputyaml{hello.yml}
\stoptext
```

2.4.2 Options

The ConTEXt options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` (or, equivalently, ⟨*key*⟩=`yes`) if the =⟨*value*⟩ part has been omitted.

ConTEXt options map directly to the options recognized by the plain TEX interface (see Section 2.2.2).

The ConTEXt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

```
\ExplSyntaxOn
\cs_new:Npn
  \setupmarkdown
  [ #1 ]
  {
    \@@_setup:n
      { #1 }
  }
```

The command `\setupyaml` is also available as an alias for the command `\setupmarkdown`.

```
\cs_gset_eq:NN
  \setupyaml
  \setupmarkdown
```

2.4.2.1 Generating Plain TEX Option Macros and Key-Values

Unlike plain TEX, we also accept caseless variants of options in line with the style of ConTEXt.

```
\cs_new:Nn \@@_caseless:N
  {
    \regex_replace_all:nnN
      { ([a-z])([A-Z]) }
      { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
```

If ConTEXt is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to define plain TEX option, token renderer, and token renderer prototype macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_context_tl
  {
    \@@_define_option_commands_and_keyvals:
    \@@_define_renderers:
    \@@_define_renderer_prototypes:
  }
```

2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConTEXt, we expand on the concept of themes by allowing a theme to be a full-blown ConTEXt module. Specifically, the key–values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ load a ConTEXt module named `t-markdowntheme`⟨*munged theme name*⟩`.tex` if it exists and a TEX document named `markdowntheme`⟨*munged theme name*⟩`.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex` *theme file* or the ConTEXt-specific `t-*.tex` theme file allows developers to have a single *theme file*, when the theme is small or the difference between TEX formats is unimportant, and scale up to separate theme files native to different TEX formats for large multi-format themes, where different code is needed for different TEX formats. To enable code reuse, developers can load the `.tex` theme file from the `t-*.tex` theme file using the `\markdownLoadPlainTeXTheme` macro.

For example, to load a theme named `witiko/tilde` in your document:

```
\usemodule[t][markdown]
\setupmarkdown[import=witiko/tilde]
```

We also define the prop `\g_@@_context_built_in_themes_prop` that contains the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```
\prop_new:N
  \g_@@_context_built_in_themes_prop
\ExplSyntaxOff
```

Built-in ConTEXt themes provided with the Markdown package include:

witiko/markdown/defaults A ConTEXt theme with the default definitions of token renderer prototypes for plain TEX. This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
\startmodule[markdownthemewitiko_markdown_defaults]
\unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConTEXt themes.

3 Implementation

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is performed by the Lua layer. The plain

TEX layer provides default definitions for the token renderers. The LATEX and ConTEXt layers correct idiosyncrasies of the respective TEX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain TEX, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

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

```
local upper, format, length =
  string.upper, string.format, string.len
local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
  lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
  lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

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

```
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.

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

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exists, it gets created with `transform(string)` as its content. Regardless, the pathname is then returned.

```
function util.cache(dir, string, salt, transform, suffix)
  local digest = md5.sumhexa(string .. (salt or ""))
  local name = util.pathname(dir, digest .. suffix)
  local file = io.open(name, "r")
  if file == nil then -- If no cache entry exists, create a new one.
    file = assert(io.open(name, "w"),
      [[Could not open file "]] .. name .. [[" for writing]])
    local result = string
```

```
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  return name
end
```

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

```
function util.cache_verbatim(dir, string)
  local name = util.cache(dir, string, nil, nil, ".verbatim")
  return name
end
```

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

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

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

```
function util.encode_json_string(s)
  s = s:gsub([[\]], [[\\]])
  s = s:gsub([["]], [[\"]])
  return [["]] .. s .. [["]]
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 [16, Chapter 21].

```
function util.expand_tabs_in_line(s, tabstop)
  local tab = tabstop or 4
  local corr = 0
  return (s:gsub("()\t", function(p)
            local sp = tab - (p - 1 + corr) % tab
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
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.

```
function util.walk(t, f)
  local typ = type(t)
  if typ == "string" then
    f(t)
  elseif typ == "table" then
    local i = 1
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

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

```
function util.flatten(ary)
  local new = {}
  for _,v in ipairs(ary) do
    if type(v) == "table" then
      for _,w in ipairs(util.flatten(v)) do
        new[#new + 1] = w
      end
    else
      new[#new + 1] = v
    end
  end
  return new
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.

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

```
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.

```
function util.rope_last(rope)
  if #rope == 0 then
    return nil
  else
    local l = rope[#rope]
    if type(l) == "table" then
      return util.rope_last(l)
    else
      return l
    end
  end
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 $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.intersperse(ary, x)
  local new = {}
  local l = #ary
  for i,v in ipairs(ary) do
    local n = #new
    new[n + 1] = v
    if i ~= l then
      new[n + 2] = x
    end
  end
  return new
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 $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.map(ary, f)
  local new = {}
  for i,v in ipairs(ary) do
    new[i] = f(v)
  end
  return new
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.

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

Build a string of escapable characters.

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

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

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

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

$$\sum_{\texttt{(k,v)}\in\texttt{string\_escapes}} \texttt{P(k) / v} + \texttt{escapable}$$

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

```
  if string_escapes then
    for k,v in pairs(string_escapes) do
      escapable = P(k) / v + escapable
    end
  end
```

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

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

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

```
  return function(s)
    return lpeg.match(escape_string, s)
  end
end
```

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

```
function util.pathname(dir, file)
  if #dir == 0 then
    return file
  else
    return dir .. "/" .. file
  end
end
```

The `util.salt` method produces cryptographic salt out of a table of options `options`.

```
function util.salt(options)
  local opt_string = {}
  for k, _ in pairs(defaultOptions) do
    local v = options[k]
    if type(v) == "table" then
      for _, i in ipairs(v) do
        opt_string[#opt_string+1] = k .. "=" .. tostring(i)
      end
```

The `cacheDir` option is disregarded.

```
    elseif k ~= "cacheDir" then
      opt_string[#opt_string+1] = k .. "=" .. tostring(v)
    end
  end
  table.sort(opt_string)
  local salt = table.concat(opt_string, ",")
            .. "," .. metadata.version
  return salt
end
```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```
function util.warning(s)
  io.stderr:write("Warning: " .. s .. "\n")
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.

```
local entities = {}

local character_entities = {
  ["Tab"] = 9,
  ["NewLine"] = 10,
  ["excl"] = 33,
  ["QUOT"] = 34,
  ["quot"] = 34,
  ["num"] = 35,
  ["dollar"] = 36,
  ["percnt"] = 37,
  ["AMP"] = 38,
  ["amp"] = 38,
  ["apos"] = 39,
```

```
4136   ["lpar"] = 40,
4137   ["rpar"] = 41,
4138   ["ast"] = 42,
4139   ["midast"] = 42,
4140   ["plus"] = 43,
4141   ["comma"] = 44,
4142   ["period"] = 46,
4143   ["sol"] = 47,
4144   ["colon"] = 58,
4145   ["semi"] = 59,
4146   ["LT"] = 60,
4147   ["lt"] = 60,
4148   ["nvlt"] = {60, 8402},
4149   ["bne"] = {61, 8421},
4150   ["equals"] = 61,
4151   ["GT"] = 62,
4152   ["gt"] = 62,
4153   ["nvgt"] = {62, 8402},
4154   ["quest"] = 63,
4155   ["commat"] = 64,
4156   ["lbrack"] = 91,
4157   ["lsqb"] = 91,
4158   ["bsol"] = 92,
4159   ["rbrack"] = 93,
4160   ["rsqb"] = 93,
4161   ["Hat"] = 94,
4162   ["UnderBar"] = 95,
4163   ["lowbar"] = 95,
4164   ["DiacriticalGrave"] = 96,
4165   ["grave"] = 96,
4166   ["fjlig"] = {102, 106},
4167   ["lbrace"] = 123,
4168   ["lcub"] = 123,
4169   ["VerticalLine"] = 124,
4170   ["verbar"] = 124,
4171   ["vert"] = 124,
4172   ["rbrace"] = 125,
4173   ["rcub"] = 125,
4174   ["NonBreakingSpace"] = 160,
4175   ["nbsp"] = 160,
4176   ["iexcl"] = 161,
4177   ["cent"] = 162,
4178   ["pound"] = 163,
4179   ["curren"] = 164,
4180   ["yen"] = 165,
4181   ["brvbar"] = 166,
4182   ["sect"] = 167,
```

```
4183   ["Dot"] = 168,
4184   ["DoubleDot"] = 168,
4185   ["die"] = 168,
4186   ["uml"] = 168,
4187   ["COPY"] = 169,
4188   ["copy"] = 169,
4189   ["ordf"] = 170,
4190   ["laquo"] = 171,
4191   ["not"] = 172,
4192   ["shy"] = 173,
4193   ["REG"] = 174,
4194   ["circledR"] = 174,
4195   ["reg"] = 174,
4196   ["macr"] = 175,
4197   ["strns"] = 175,
4198   ["deg"] = 176,
4199   ["PlusMinus"] = 177,
4200   ["plusmn"] = 177,
4201   ["pm"] = 177,
4202   ["sup2"] = 178,
4203   ["sup3"] = 179,
4204   ["DiacriticalAcute"] = 180,
4205   ["acute"] = 180,
4206   ["micro"] = 181,
4207   ["para"] = 182,
4208   ["CenterDot"] = 183,
4209   ["centerdot"] = 183,
4210   ["middot"] = 183,
4211   ["Cedilla"] = 184,
4212   ["cedil"] = 184,
4213   ["sup1"] = 185,
4214   ["ordm"] = 186,
4215   ["raquo"] = 187,
4216   ["frac14"] = 188,
4217   ["frac12"] = 189,
4218   ["half"] = 189,
4219   ["frac34"] = 190,
4220   ["iquest"] = 191,
4221   ["Agrave"] = 192,
4222   ["Aacute"] = 193,
4223   ["Acirc"] = 194,
4224   ["Atilde"] = 195,
4225   ["Auml"] = 196,
4226   ["Aring"] = 197,
4227   ["angst"] = 197,
4228   ["AElig"] = 198,
4229   ["Ccedil"] = 199,
```

```
4230    ["Egrave"] = 200,
4231    ["Eacute"] = 201,
4232    ["Ecirc"] = 202,
4233    ["Euml"] = 203,
4234    ["Igrave"] = 204,
4235    ["Iacute"] = 205,
4236    ["Icirc"] = 206,
4237    ["Iuml"] = 207,
4238    ["ETH"] = 208,
4239    ["Ntilde"] = 209,
4240    ["Ograve"] = 210,
4241    ["Oacute"] = 211,
4242    ["Ocirc"] = 212,
4243    ["Otilde"] = 213,
4244    ["Ouml"] = 214,
4245    ["times"] = 215,
4246    ["Oslash"] = 216,
4247    ["Ugrave"] = 217,
4248    ["Uacute"] = 218,
4249    ["Ucirc"] = 219,
4250    ["Uuml"] = 220,
4251    ["Yacute"] = 221,
4252    ["THORN"] = 222,
4253    ["szlig"] = 223,
4254    ["agrave"] = 224,
4255    ["aacute"] = 225,
4256    ["acirc"] = 226,
4257    ["atilde"] = 227,
4258    ["auml"] = 228,
4259    ["aring"] = 229,
4260    ["aelig"] = 230,
4261    ["ccedil"] = 231,
4262    ["egrave"] = 232,
4263    ["eacute"] = 233,
4264    ["ecirc"] = 234,
4265    ["euml"] = 235,
4266    ["igrave"] = 236,
4267    ["iacute"] = 237,
4268    ["icirc"] = 238,
4269    ["iuml"] = 239,
4270    ["eth"] = 240,
4271    ["ntilde"] = 241,
4272    ["ograve"] = 242,
4273    ["oacute"] = 243,
4274    ["ocirc"] = 244,
4275    ["otilde"] = 245,
4276    ["ouml"] = 246,
```

```
4277   ["div"] = 247,
4278   ["divide"] = 247,
4279   ["oslash"] = 248,
4280   ["ugrave"] = 249,
4281   ["uacute"] = 250,
4282   ["ucirc"] = 251,
4283   ["uuml"] = 252,
4284   ["yacute"] = 253,
4285   ["thorn"] = 254,
4286   ["yuml"] = 255,
4287   ["Amacr"] = 256,
4288   ["amacr"] = 257,
4289   ["Abreve"] = 258,
4290   ["abreve"] = 259,
4291   ["Aogon"] = 260,
4292   ["aogon"] = 261,
4293   ["Cacute"] = 262,
4294   ["cacute"] = 263,
4295   ["Ccirc"] = 264,
4296   ["ccirc"] = 265,
4297   ["Cdot"] = 266,
4298   ["cdot"] = 267,
4299   ["Ccaron"] = 268,
4300   ["ccaron"] = 269,
4301   ["Dcaron"] = 270,
4302   ["dcaron"] = 271,
4303   ["Dstrok"] = 272,
4304   ["dstrok"] = 273,
4305   ["Emacr"] = 274,
4306   ["emacr"] = 275,
4307   ["Edot"] = 278,
4308   ["edot"] = 279,
4309   ["Eogon"] = 280,
4310   ["eogon"] = 281,
4311   ["Ecaron"] = 282,
4312   ["ecaron"] = 283,
4313   ["Gcirc"] = 284,
4314   ["gcirc"] = 285,
4315   ["Gbreve"] = 286,
4316   ["gbreve"] = 287,
4317   ["Gdot"] = 288,
4318   ["gdot"] = 289,
4319   ["Gcedil"] = 290,
4320   ["Hcirc"] = 292,
4321   ["hcirc"] = 293,
4322   ["Hstrok"] = 294,
4323   ["hstrok"] = 295,
```

```
4324   ["Itilde"] = 296,
4325   ["itilde"] = 297,
4326   ["Imacr"] = 298,
4327   ["imacr"] = 299,
4328   ["Iogon"] = 302,
4329   ["iogon"] = 303,
4330   ["Idot"] = 304,
4331   ["imath"] = 305,
4332   ["inodot"] = 305,
4333   ["IJlig"] = 306,
4334   ["ijlig"] = 307,
4335   ["Jcirc"] = 308,
4336   ["jcirc"] = 309,
4337   ["Kcedil"] = 310,
4338   ["kcedil"] = 311,
4339   ["kgreen"] = 312,
4340   ["Lacute"] = 313,
4341   ["lacute"] = 314,
4342   ["Lcedil"] = 315,
4343   ["lcedil"] = 316,
4344   ["Lcaron"] = 317,
4345   ["lcaron"] = 318,
4346   ["Lmidot"] = 319,
4347   ["lmidot"] = 320,
4348   ["Lstrok"] = 321,
4349   ["lstrok"] = 322,
4350   ["Nacute"] = 323,
4351   ["nacute"] = 324,
4352   ["Ncedil"] = 325,
4353   ["ncedil"] = 326,
4354   ["Ncaron"] = 327,
4355   ["ncaron"] = 328,
4356   ["napos"] = 329,
4357   ["ENG"] = 330,
4358   ["eng"] = 331,
4359   ["Omacr"] = 332,
4360   ["omacr"] = 333,
4361   ["Odblac"] = 336,
4362   ["odblac"] = 337,
4363   ["OElig"] = 338,
4364   ["oelig"] = 339,
4365   ["Racute"] = 340,
4366   ["racute"] = 341,
4367   ["Rcedil"] = 342,
4368   ["rcedil"] = 343,
4369   ["Rcaron"] = 344,
4370   ["rcaron"] = 345,
```

```
4371   ["Sacute"] = 346,
4372   ["sacute"] = 347,
4373   ["Scirc"] = 348,
4374   ["scirc"] = 349,
4375   ["Scedil"] = 350,
4376   ["scedil"] = 351,
4377   ["Scaron"] = 352,
4378   ["scaron"] = 353,
4379   ["Tcedil"] = 354,
4380   ["tcedil"] = 355,
4381   ["Tcaron"] = 356,
4382   ["tcaron"] = 357,
4383   ["Tstrok"] = 358,
4384   ["tstrok"] = 359,
4385   ["Utilde"] = 360,
4386   ["utilde"] = 361,
4387   ["Umacr"] = 362,
4388   ["umacr"] = 363,
4389   ["Ubreve"] = 364,
4390   ["ubreve"] = 365,
4391   ["Uring"] = 366,
4392   ["uring"] = 367,
4393   ["Udblac"] = 368,
4394   ["udblac"] = 369,
4395   ["Uogon"] = 370,
4396   ["uogon"] = 371,
4397   ["Wcirc"] = 372,
4398   ["wcirc"] = 373,
4399   ["Ycirc"] = 374,
4400   ["ycirc"] = 375,
4401   ["Yuml"] = 376,
4402   ["Zacute"] = 377,
4403   ["zacute"] = 378,
4404   ["Zdot"] = 379,
4405   ["zdot"] = 380,
4406   ["Zcaron"] = 381,
4407   ["zcaron"] = 382,
4408   ["fnof"] = 402,
4409   ["imped"] = 437,
4410   ["gacute"] = 501,
4411   ["jmath"] = 567,
4412   ["circ"] = 710,
4413   ["Hacek"] = 711,
4414   ["caron"] = 711,
4415   ["Breve"] = 728,
4416   ["breve"] = 728,
4417   ["DiacriticalDot"] = 729,
```

```
4418   ["dot"] = 729,
4419   ["ring"] = 730,
4420   ["ogon"] = 731,
4421   ["DiacriticalTilde"] = 732,
4422   ["tilde"] = 732,
4423   ["DiacriticalDoubleAcute"] = 733,
4424   ["dblac"] = 733,
4425   ["DownBreve"] = 785,
4426   ["Alpha"] = 913,
4427   ["Beta"] = 914,
4428   ["Gamma"] = 915,
4429   ["Delta"] = 916,
4430   ["Epsilon"] = 917,
4431   ["Zeta"] = 918,
4432   ["Eta"] = 919,
4433   ["Theta"] = 920,
4434   ["Iota"] = 921,
4435   ["Kappa"] = 922,
4436   ["Lambda"] = 923,
4437   ["Mu"] = 924,
4438   ["Nu"] = 925,
4439   ["Xi"] = 926,
4440   ["Omicron"] = 927,
4441   ["Pi"] = 928,
4442   ["Rho"] = 929,
4443   ["Sigma"] = 931,
4444   ["Tau"] = 932,
4445   ["Upsilon"] = 933,
4446   ["Phi"] = 934,
4447   ["Chi"] = 935,
4448   ["Psi"] = 936,
4449   ["Omega"] = 937,
4450   ["ohm"] = 937,
4451   ["alpha"] = 945,
4452   ["beta"] = 946,
4453   ["gamma"] = 947,
4454   ["delta"] = 948,
4455   ["epsi"] = 949,
4456   ["epsilon"] = 949,
4457   ["zeta"] = 950,
4458   ["eta"] = 951,
4459   ["theta"] = 952,
4460   ["iota"] = 953,
4461   ["kappa"] = 954,
4462   ["lambda"] = 955,
4463   ["mu"] = 956,
4464   ["nu"] = 957,
```

```
4465    ["xi"] = 958,
4466    ["omicron"] = 959,
4467    ["pi"] = 960,
4468    ["rho"] = 961,
4469    ["sigmaf"] = 962,
4470    ["sigmav"] = 962,
4471    ["varsigma"] = 962,
4472    ["sigma"] = 963,
4473    ["tau"] = 964,
4474    ["upsi"] = 965,
4475    ["upsilon"] = 965,
4476    ["phi"] = 966,
4477    ["chi"] = 967,
4478    ["psi"] = 968,
4479    ["omega"] = 969,
4480    ["thetasym"] = 977,
4481    ["thetav"] = 977,
4482    ["vartheta"] = 977,
4483    ["Upsi"] = 978,
4484    ["upsih"] = 978,
4485    ["phiv"] = 981,
4486    ["straightphi"] = 981,
4487    ["varphi"] = 981,
4488    ["piv"] = 982,
4489    ["varpi"] = 982,
4490    ["Gammad"] = 988,
4491    ["digamma"] = 989,
4492    ["gammad"] = 989,
4493    ["kappav"] = 1008,
4494    ["varkappa"] = 1008,
4495    ["rhov"] = 1009,
4496    ["varrho"] = 1009,
4497    ["epsiv"] = 1013,
4498    ["straightepsilon"] = 1013,
4499    ["varepsilon"] = 1013,
4500    ["backepsilon"] = 1014,
4501    ["bepsi"] = 1014,
4502    ["IOcy"] = 1025,
4503    ["DJcy"] = 1026,
4504    ["GJcy"] = 1027,
4505    ["Jukcy"] = 1028,
4506    ["DScy"] = 1029,
4507    ["Iukcy"] = 1030,
4508    ["YIcy"] = 1031,
4509    ["Jsercy"] = 1032,
4510    ["LJcy"] = 1033,
4511    ["NJcy"] = 1034,
```

```
4512    ["TSHcy"] = 1035,
4513    ["KJcy"] = 1036,
4514    ["Ubrcy"] = 1038,
4515    ["DZcy"] = 1039,
4516    ["Acy"] = 1040,
4517    ["Bcy"] = 1041,
4518    ["Vcy"] = 1042,
4519    ["Gcy"] = 1043,
4520    ["Dcy"] = 1044,
4521    ["IEcy"] = 1045,
4522    ["ZHcy"] = 1046,
4523    ["Zcy"] = 1047,
4524    ["Icy"] = 1048,
4525    ["Jcy"] = 1049,
4526    ["Kcy"] = 1050,
4527    ["Lcy"] = 1051,
4528    ["Mcy"] = 1052,
4529    ["Ncy"] = 1053,
4530    ["Ocy"] = 1054,
4531    ["Pcy"] = 1055,
4532    ["Rcy"] = 1056,
4533    ["Scy"] = 1057,
4534    ["Tcy"] = 1058,
4535    ["Ucy"] = 1059,
4536    ["Fcy"] = 1060,
4537    ["KHcy"] = 1061,
4538    ["TScy"] = 1062,
4539    ["CHcy"] = 1063,
4540    ["SHcy"] = 1064,
4541    ["SHCHcy"] = 1065,
4542    ["HARDcy"] = 1066,
4543    ["Ycy"] = 1067,
4544    ["SOFTcy"] = 1068,
4545    ["Ecy"] = 1069,
4546    ["YUcy"] = 1070,
4547    ["YAcy"] = 1071,
4548    ["acy"] = 1072,
4549    ["bcy"] = 1073,
4550    ["vcy"] = 1074,
4551    ["gcy"] = 1075,
4552    ["dcy"] = 1076,
4553    ["iecy"] = 1077,
4554    ["zhcy"] = 1078,
4555    ["zcy"] = 1079,
4556    ["icy"] = 1080,
4557    ["jcy"] = 1081,
4558    ["kcy"] = 1082,
```

```
4559    ["lcy"] = 1083,
4560    ["mcy"] = 1084,
4561    ["ncy"] = 1085,
4562    ["ocy"] = 1086,
4563    ["pcy"] = 1087,
4564    ["rcy"] = 1088,
4565    ["scy"] = 1089,
4566    ["tcy"] = 1090,
4567    ["ucy"] = 1091,
4568    ["fcy"] = 1092,
4569    ["khcy"] = 1093,
4570    ["tscy"] = 1094,
4571    ["chcy"] = 1095,
4572    ["shcy"] = 1096,
4573    ["shchcy"] = 1097,
4574    ["hardcy"] = 1098,
4575    ["ycy"] = 1099,
4576    ["softcy"] = 1100,
4577    ["ecy"] = 1101,
4578    ["yucy"] = 1102,
4579    ["yacy"] = 1103,
4580    ["iocy"] = 1105,
4581    ["djcy"] = 1106,
4582    ["gjcy"] = 1107,
4583    ["jukcy"] = 1108,
4584    ["dscy"] = 1109,
4585    ["iukcy"] = 1110,
4586    ["yicy"] = 1111,
4587    ["jsercy"] = 1112,
4588    ["ljcy"] = 1113,
4589    ["njcy"] = 1114,
4590    ["tshcy"] = 1115,
4591    ["kjcy"] = 1116,
4592    ["ubrcy"] = 1118,
4593    ["dzcy"] = 1119,
4594    ["ensp"] = 8194,
4595    ["emsp"] = 8195,
4596    ["emsp13"] = 8196,
4597    ["emsp14"] = 8197,
4598    ["numsp"] = 8199,
4599    ["puncsp"] = 8200,
4600    ["ThinSpace"] = 8201,
4601    ["thinsp"] = 8201,
4602    ["VeryThinSpace"] = 8202,
4603    ["hairsp"] = 8202,
4604    ["NegativeMediumSpace"] = 8203,
4605    ["NegativeThickSpace"] = 8203,
```

```
  ["NegativeThinSpace"] = 8203,
  ["NegativeVeryThinSpace"] = 8203,
  ["ZeroWidthSpace"] = 8203,
  ["zwnj"] = 8204,
  ["zwj"] = 8205,
  ["lrm"] = 8206,
  ["rlm"] = 8207,
  ["dash"] = 8208,
  ["hyphen"] = 8208,
  ["ndash"] = 8211,
  ["mdash"] = 8212,
  ["horbar"] = 8213,
  ["Verbar"] = 8214,
  ["Vert"] = 8214,
  ["OpenCurlyQuote"] = 8216,
  ["lsquo"] = 8216,
  ["CloseCurlyQuote"] = 8217,
  ["rsquo"] = 8217,
  ["rsquor"] = 8217,
  ["lsquor"] = 8218,
  ["sbquo"] = 8218,
  ["OpenCurlyDoubleQuote"] = 8220,
  ["ldquo"] = 8220,
  ["CloseCurlyDoubleQuote"] = 8221,
  ["rdquo"] = 8221,
  ["rdquor"] = 8221,
  ["bdquo"] = 8222,
  ["ldquor"] = 8222,
  ["dagger"] = 8224,
  ["Dagger"] = 8225,
  ["ddagger"] = 8225,
  ["bull"] = 8226,
  ["bullet"] = 8226,
  ["nldr"] = 8229,
  ["hellip"] = 8230,
  ["mldr"] = 8230,
  ["permil"] = 8240,
  ["pertenk"] = 8241,
  ["prime"] = 8242,
  ["Prime"] = 8243,
  ["tprime"] = 8244,
  ["backprime"] = 8245,
  ["bprime"] = 8245,
  ["lsaquo"] = 8249,
  ["rsaquo"] = 8250,
  ["OverBar"] = 8254,
  ["oline"] = 8254,
```

```
  ["caret"] = 8257,
  ["hybull"] = 8259,
  ["frasl"] = 8260,
  ["bsemi"] = 8271,
  ["qprime"] = 8279,
  ["MediumSpace"] = 8287,
  ["ThickSpace"] = {8287, 8202},
  ["NoBreak"] = 8288,
  ["ApplyFunction"] = 8289,
  ["af"] = 8289,
  ["InvisibleTimes"] = 8290,
  ["it"] = 8290,
  ["InvisibleComma"] = 8291,
  ["ic"] = 8291,
  ["euro"] = 8364,
  ["TripleDot"] = 8411,
  ["tdot"] = 8411,
  ["DotDot"] = 8412,
  ["Copf"] = 8450,
  ["complexes"] = 8450,
  ["incare"] = 8453,
  ["gscr"] = 8458,
  ["HilbertSpace"] = 8459,
  ["Hscr"] = 8459,
  ["hamilt"] = 8459,
  ["Hfr"] = 8460,
  ["Poincareplane"] = 8460,
  ["Hopf"] = 8461,
  ["quaternions"] = 8461,
  ["planckh"] = 8462,
  ["hbar"] = 8463,
  ["hslash"] = 8463,
  ["planck"] = 8463,
  ["plankv"] = 8463,
  ["Iscr"] = 8464,
  ["imagline"] = 8464,
  ["Ifr"] = 8465,
  ["Im"] = 8465,
  ["image"] = 8465,
  ["imagpart"] = 8465,
  ["Laplacetrf"] = 8466,
  ["Lscr"] = 8466,
  ["lagran"] = 8466,
  ["ell"] = 8467,
  ["Nopf"] = 8469,
  ["naturals"] = 8469,
  ["numero"] = 8470,
```

```
4700    ["copysr"] = 8471,
4701    ["weierp"] = 8472,
4702    ["wp"] = 8472,
4703    ["Popf"] = 8473,
4704    ["primes"] = 8473,
4705    ["Qopf"] = 8474,
4706    ["rationals"] = 8474,
4707    ["Rscr"] = 8475,
4708    ["realine"] = 8475,
4709    ["Re"] = 8476,
4710    ["Rfr"] = 8476,
4711    ["real"] = 8476,
4712    ["realpart"] = 8476,
4713    ["Ropf"] = 8477,
4714    ["reals"] = 8477,
4715    ["rx"] = 8478,
4716    ["TRADE"] = 8482,
4717    ["trade"] = 8482,
4718    ["Zopf"] = 8484,
4719    ["integers"] = 8484,
4720    ["mho"] = 8487,
4721    ["Zfr"] = 8488,
4722    ["zeetrf"] = 8488,
4723    ["iiota"] = 8489,
4724    ["Bernoullis"] = 8492,
4725    ["Bscr"] = 8492,
4726    ["bernou"] = 8492,
4727    ["Cayleys"] = 8493,
4728    ["Cfr"] = 8493,
4729    ["escr"] = 8495,
4730    ["Escr"] = 8496,
4731    ["expectation"] = 8496,
4732    ["Fouriertrf"] = 8497,
4733    ["Fscr"] = 8497,
4734    ["Mellintrf"] = 8499,
4735    ["Mscr"] = 8499,
4736    ["phmmat"] = 8499,
4737    ["order"] = 8500,
4738    ["orderof"] = 8500,
4739    ["oscr"] = 8500,
4740    ["alefsym"] = 8501,
4741    ["aleph"] = 8501,
4742    ["beth"] = 8502,
4743    ["gimel"] = 8503,
4744    ["daleth"] = 8504,
4745    ["CapitalDifferentialD"] = 8517,
4746    ["DD"] = 8517,
```

```
4747   ["DifferentialD"] = 8518,
4748   ["dd"] = 8518,
4749   ["ExponentialE"] = 8519,
4750   ["ee"] = 8519,
4751   ["exponentiale"] = 8519,
4752   ["ImaginaryI"] = 8520,
4753   ["ii"] = 8520,
4754   ["frac13"] = 8531,
4755   ["frac23"] = 8532,
4756   ["frac15"] = 8533,
4757   ["frac25"] = 8534,
4758   ["frac35"] = 8535,
4759   ["frac45"] = 8536,
4760   ["frac16"] = 8537,
4761   ["frac56"] = 8538,
4762   ["frac18"] = 8539,
4763   ["frac38"] = 8540,
4764   ["frac58"] = 8541,
4765   ["frac78"] = 8542,
4766   ["LeftArrow"] = 8592,
4767   ["ShortLeftArrow"] = 8592,
4768   ["larr"] = 8592,
4769   ["leftarrow"] = 8592,
4770   ["slarr"] = 8592,
4771   ["ShortUpArrow"] = 8593,
4772   ["UpArrow"] = 8593,
4773   ["uarr"] = 8593,
4774   ["uparrow"] = 8593,
4775   ["RightArrow"] = 8594,
4776   ["ShortRightArrow"] = 8594,
4777   ["rarr"] = 8594,
4778   ["rightarrow"] = 8594,
4779   ["srarr"] = 8594,
4780   ["DownArrow"] = 8595,
4781   ["ShortDownArrow"] = 8595,
4782   ["darr"] = 8595,
4783   ["downarrow"] = 8595,
4784   ["LeftRightArrow"] = 8596,
4785   ["harr"] = 8596,
4786   ["leftrightarrow"] = 8596,
4787   ["UpDownArrow"] = 8597,
4788   ["updownarrow"] = 8597,
4789   ["varr"] = 8597,
4790   ["UpperLeftArrow"] = 8598,
4791   ["nwarr"] = 8598,
4792   ["nwarrow"] = 8598,
4793   ["UpperRightArrow"] = 8599,
```

```
  ["nearr"] = 8599,
  ["nearrow"] = 8599,
  ["LowerRightArrow"] = 8600,
  ["searr"] = 8600,
  ["searrow"] = 8600,
  ["LowerLeftArrow"] = 8601,
  ["swarr"] = 8601,
  ["swarrow"] = 8601,
  ["nlarr"] = 8602,
  ["nleftarrow"] = 8602,
  ["nrarr"] = 8603,
  ["nrightarrow"] = 8603,
  ["nrarrw"] = {8605, 824},
  ["rarrw"] = 8605,
  ["rightsquigarrow"] = 8605,
  ["Larr"] = 8606,
  ["twoheadleftarrow"] = 8606,
  ["Uarr"] = 8607,
  ["Rarr"] = 8608,
  ["twoheadrightarrow"] = 8608,
  ["Darr"] = 8609,
  ["larrtl"] = 8610,
  ["leftarrowtail"] = 8610,
  ["rarrtl"] = 8611,
  ["rightarrowtail"] = 8611,
  ["LeftTeeArrow"] = 8612,
  ["mapstoleft"] = 8612,
  ["UpTeeArrow"] = 8613,
  ["mapstoup"] = 8613,
  ["RightTeeArrow"] = 8614,
  ["map"] = 8614,
  ["mapsto"] = 8614,
  ["DownTeeArrow"] = 8615,
  ["mapstodown"] = 8615,
  ["hookleftarrow"] = 8617,
  ["larrhk"] = 8617,
  ["hookrightarrow"] = 8618,
  ["rarrhk"] = 8618,
  ["larrlp"] = 8619,
  ["looparrowleft"] = 8619,
  ["looparrowright"] = 8620,
  ["rarrlp"] = 8620,
  ["harrw"] = 8621,
  ["leftrightsquigarrow"] = 8621,
  ["nharr"] = 8622,
  ["nleftrightarrow"] = 8622,
  ["Lsh"] = 8624,
```

```
4841  ["lsh"] = 8624,
4842  ["Rsh"] = 8625,
4843  ["rsh"] = 8625,
4844  ["ldsh"] = 8626,
4845  ["rdsh"] = 8627,
4846  ["crarr"] = 8629,
4847  ["cularr"] = 8630,
4848  ["curvearrowleft"] = 8630,
4849  ["curarr"] = 8631,
4850  ["curvearrowright"] = 8631,
4851  ["circlearrowleft"] = 8634,
4852  ["olarr"] = 8634,
4853  ["circlearrowright"] = 8635,
4854  ["orarr"] = 8635,
4855  ["LeftVector"] = 8636,
4856  ["leftharpoonup"] = 8636,
4857  ["lharu"] = 8636,
4858  ["DownLeftVector"] = 8637,
4859  ["leftharpoondown"] = 8637,
4860  ["lhard"] = 8637,
4861  ["RightUpVector"] = 8638,
4862  ["uharr"] = 8638,
4863  ["upharpoonright"] = 8638,
4864  ["LeftUpVector"] = 8639,
4865  ["uharl"] = 8639,
4866  ["upharpoonleft"] = 8639,
4867  ["RightVector"] = 8640,
4868  ["rharu"] = 8640,
4869  ["rightharpoonup"] = 8640,
4870  ["DownRightVector"] = 8641,
4871  ["rhard"] = 8641,
4872  ["rightharpoondown"] = 8641,
4873  ["RightDownVector"] = 8642,
4874  ["dharr"] = 8642,
4875  ["downharpoonright"] = 8642,
4876  ["LeftDownVector"] = 8643,
4877  ["dharl"] = 8643,
4878  ["downharpoonleft"] = 8643,
4879  ["RightArrowLeftArrow"] = 8644,
4880  ["rightleftarrows"] = 8644,
4881  ["rlarr"] = 8644,
4882  ["UpArrowDownArrow"] = 8645,
4883  ["udarr"] = 8645,
4884  ["LeftArrowRightArrow"] = 8646,
4885  ["leftrightarrows"] = 8646,
4886  ["lrarr"] = 8646,
4887  ["leftleftarrows"] = 8647,
```

```
  ["llarr"] = 8647,
  ["upuparrows"] = 8648,
  ["uuarr"] = 8648,
  ["rightrightarrows"] = 8649,
  ["rrarr"] = 8649,
  ["ddarr"] = 8650,
  ["downdownarrows"] = 8650,
  ["ReverseEquilibrium"] = 8651,
  ["leftrightharpoons"] = 8651,
  ["lrhar"] = 8651,
  ["Equilibrium"] = 8652,
  ["rightleftharpoons"] = 8652,
  ["rlhar"] = 8652,
  ["nLeftarrow"] = 8653,
  ["nlArr"] = 8653,
  ["nLeftrightarrow"] = 8654,
  ["nhArr"] = 8654,
  ["nRightarrow"] = 8655,
  ["nrArr"] = 8655,
  ["DoubleLeftArrow"] = 8656,
  ["Leftarrow"] = 8656,
  ["lArr"] = 8656,
  ["DoubleUpArrow"] = 8657,
  ["Uparrow"] = 8657,
  ["uArr"] = 8657,
  ["DoubleRightArrow"] = 8658,
  ["Implies"] = 8658,
  ["Rightarrow"] = 8658,
  ["rArr"] = 8658,
  ["DoubleDownArrow"] = 8659,
  ["Downarrow"] = 8659,
  ["dArr"] = 8659,
  ["DoubleLeftRightArrow"] = 8660,
  ["Leftrightarrow"] = 8660,
  ["hArr"] = 8660,
  ["iff"] = 8660,
  ["DoubleUpDownArrow"] = 8661,
  ["Updownarrow"] = 8661,
  ["vArr"] = 8661,
  ["nwArr"] = 8662,
  ["neArr"] = 8663,
  ["seArr"] = 8664,
  ["swArr"] = 8665,
  ["Lleftarrow"] = 8666,
  ["lAarr"] = 8666,
  ["Rrightarrow"] = 8667,
  ["rAarr"] = 8667,
```

```
4935  ["zigrarr"] = 8669,
4936  ["LeftArrowBar"] = 8676,
4937  ["larrb"] = 8676,
4938  ["RightArrowBar"] = 8677,
4939  ["rarrb"] = 8677,
4940  ["DownArrowUpArrow"] = 8693,
4941  ["duarr"] = 8693,
4942  ["loarr"] = 8701,
4943  ["roarr"] = 8702,
4944  ["hoarr"] = 8703,
4945  ["ForAll"] = 8704,
4946  ["forall"] = 8704,
4947  ["comp"] = 8705,
4948  ["complement"] = 8705,
4949  ["PartialD"] = 8706,
4950  ["npart"] = {8706, 824},
4951  ["part"] = 8706,
4952  ["Exists"] = 8707,
4953  ["exist"] = 8707,
4954  ["NotExists"] = 8708,
4955  ["nexist"] = 8708,
4956  ["nexists"] = 8708,
4957  ["empty"] = 8709,
4958  ["emptyset"] = 8709,
4959  ["emptyv"] = 8709,
4960  ["varnothing"] = 8709,
4961  ["Del"] = 8711,
4962  ["nabla"] = 8711,
4963  ["Element"] = 8712,
4964  ["in"] = 8712,
4965  ["isin"] = 8712,
4966  ["isinv"] = 8712,
4967  ["NotElement"] = 8713,
4968  ["notin"] = 8713,
4969  ["notinva"] = 8713,
4970  ["ReverseElement"] = 8715,
4971  ["SuchThat"] = 8715,
4972  ["ni"] = 8715,
4973  ["niv"] = 8715,
4974  ["NotReverseElement"] = 8716,
4975  ["notni"] = 8716,
4976  ["notniva"] = 8716,
4977  ["Product"] = 8719,
4978  ["prod"] = 8719,
4979  ["Coproduct"] = 8720,
4980  ["coprod"] = 8720,
4981  ["Sum"] = 8721,
```

```
4982  ["sum"] = 8721,
4983  ["minus"] = 8722,
4984  ["MinusPlus"] = 8723,
4985  ["mnplus"] = 8723,
4986  ["mp"] = 8723,
4987  ["dotplus"] = 8724,
4988  ["plusdo"] = 8724,
4989  ["Backslash"] = 8726,
4990  ["setminus"] = 8726,
4991  ["setmn"] = 8726,
4992  ["smallsetminus"] = 8726,
4993  ["ssetmn"] = 8726,
4994  ["lowast"] = 8727,
4995  ["SmallCircle"] = 8728,
4996  ["compfn"] = 8728,
4997  ["Sqrt"] = 8730,
4998  ["radic"] = 8730,
4999  ["Proportional"] = 8733,
5000  ["prop"] = 8733,
5001  ["propto"] = 8733,
5002  ["varpropto"] = 8733,
5003  ["vprop"] = 8733,
5004  ["infin"] = 8734,
5005  ["angrt"] = 8735,
5006  ["ang"] = 8736,
5007  ["angle"] = 8736,
5008  ["nang"] = {8736, 8402},
5009  ["angmsd"] = 8737,
5010  ["measuredangle"] = 8737,
5011  ["angsph"] = 8738,
5012  ["VerticalBar"] = 8739,
5013  ["mid"] = 8739,
5014  ["shortmid"] = 8739,
5015  ["smid"] = 8739,
5016  ["NotVerticalBar"] = 8740,
5017  ["nmid"] = 8740,
5018  ["nshortmid"] = 8740,
5019  ["nsmid"] = 8740,
5020  ["DoubleVerticalBar"] = 8741,
5021  ["par"] = 8741,
5022  ["parallel"] = 8741,
5023  ["shortparallel"] = 8741,
5024  ["spar"] = 8741,
5025  ["NotDoubleVerticalBar"] = 8742,
5026  ["npar"] = 8742,
5027  ["nparallel"] = 8742,
5028  ["nshortparallel"] = 8742,
```

```
  ["nspar"] = 8742,
  ["and"] = 8743,
  ["wedge"] = 8743,
  ["or"] = 8744,
  ["vee"] = 8744,
  ["cap"] = 8745,
  ["caps"] = {8745, 65024},
  ["cup"] = 8746,
  ["cups"] = {8746, 65024},
  ["Integral"] = 8747,
  ["int"] = 8747,
  ["Int"] = 8748,
  ["iiint"] = 8749,
  ["tint"] = 8749,
  ["ContourIntegral"] = 8750,
  ["conint"] = 8750,
  ["oint"] = 8750,
  ["Conint"] = 8751,
  ["DoubleContourIntegral"] = 8751,
  ["Cconint"] = 8752,
  ["cwint"] = 8753,
  ["ClockwiseContourIntegral"] = 8754,
  ["cwconint"] = 8754,
  ["CounterClockwiseContourIntegral"] = 8755,
  ["awconint"] = 8755,
  ["Therefore"] = 8756,
  ["there4"] = 8756,
  ["therefore"] = 8756,
  ["Because"] = 8757,
  ["becaus"] = 8757,
  ["because"] = 8757,
  ["ratio"] = 8758,
  ["Colon"] = 8759,
  ["Proportion"] = 8759,
  ["dotminus"] = 8760,
  ["minusd"] = 8760,
  ["mDDot"] = 8762,
  ["homtht"] = 8763,
  ["Tilde"] = 8764,
  ["nvsim"] = {8764, 8402},
  ["sim"] = 8764,
  ["thicksim"] = 8764,
  ["thksim"] = 8764,
  ["backsim"] = 8765,
  ["bsim"] = 8765,
  ["race"] = {8765, 817},
  ["ac"] = 8766,
```

```
5076   ["acE"] = {8766, 819},
5077   ["mstpos"] = 8766,
5078   ["acd"] = 8767,
5079   ["VerticalTilde"] = 8768,
5080   ["wr"] = 8768,
5081   ["wreath"] = 8768,
5082   ["NotTilde"] = 8769,
5083   ["nsim"] = 8769,
5084   ["EqualTilde"] = 8770,
5085   ["NotEqualTilde"] = {8770, 824},
5086   ["eqsim"] = 8770,
5087   ["esim"] = 8770,
5088   ["nesim"] = {8770, 824},
5089   ["TildeEqual"] = 8771,
5090   ["sime"] = 8771,
5091   ["simeq"] = 8771,
5092   ["NotTildeEqual"] = 8772,
5093   ["nsime"] = 8772,
5094   ["nsimeq"] = 8772,
5095   ["TildeFullEqual"] = 8773,
5096   ["cong"] = 8773,
5097   ["simne"] = 8774,
5098   ["NotTildeFullEqual"] = 8775,
5099   ["ncong"] = 8775,
5100   ["TildeTilde"] = 8776,
5101   ["ap"] = 8776,
5102   ["approx"] = 8776,
5103   ["asymp"] = 8776,
5104   ["thickapprox"] = 8776,
5105   ["thkap"] = 8776,
5106   ["NotTildeTilde"] = 8777,
5107   ["nap"] = 8777,
5108   ["napprox"] = 8777,
5109   ["ape"] = 8778,
5110   ["approxeq"] = 8778,
5111   ["apid"] = 8779,
5112   ["napid"] = {8779, 824},
5113   ["backcong"] = 8780,
5114   ["bcong"] = 8780,
5115   ["CupCap"] = 8781,
5116   ["asympeq"] = 8781,
5117   ["nvap"] = {8781, 8402},
5118   ["Bumpeq"] = 8782,
5119   ["HumpDownHump"] = 8782,
5120   ["NotHumpDownHump"] = {8782, 824},
5121   ["bump"] = 8782,
5122   ["nbump"] = {8782, 824},
```

```
5123    ["HumpEqual"] = 8783,
5124    ["NotHumpEqual"] = {8783, 824},
5125    ["bumpe"] = 8783,
5126    ["bumpeq"] = 8783,
5127    ["nbumpe"] = {8783, 824},
5128    ["DotEqual"] = 8784,
5129    ["doteq"] = 8784,
5130    ["esdot"] = 8784,
5131    ["nedot"] = {8784, 824},
5132    ["doteqdot"] = 8785,
5133    ["eDot"] = 8785,
5134    ["efDot"] = 8786,
5135    ["fallingdotseq"] = 8786,
5136    ["erDot"] = 8787,
5137    ["risingdotseq"] = 8787,
5138    ["Assign"] = 8788,
5139    ["colone"] = 8788,
5140    ["coloneq"] = 8788,
5141    ["ecolon"] = 8789,
5142    ["eqcolon"] = 8789,
5143    ["ecir"] = 8790,
5144    ["eqcirc"] = 8790,
5145    ["circeq"] = 8791,
5146    ["cire"] = 8791,
5147    ["wedgeq"] = 8793,
5148    ["veeeq"] = 8794,
5149    ["triangleq"] = 8796,
5150    ["trie"] = 8796,
5151    ["equest"] = 8799,
5152    ["questeq"] = 8799,
5153    ["NotEqual"] = 8800,
5154    ["ne"] = 8800,
5155    ["Congruent"] = 8801,
5156    ["bnequiv"] = {8801, 8421},
5157    ["equiv"] = 8801,
5158    ["NotCongruent"] = 8802,
5159    ["nequiv"] = 8802,
5160    ["le"] = 8804,
5161    ["leq"] = 8804,
5162    ["nvle"] = {8804, 8402},
5163    ["GreaterEqual"] = 8805,
5164    ["ge"] = 8805,
5165    ["geq"] = 8805,
5166    ["nvge"] = {8805, 8402},
5167    ["LessFullEqual"] = 8806,
5168    ["lE"] = 8806,
5169    ["leqq"] = 8806,
```

```
5170  ["nlE"] = {8806, 824},
5171  ["nleqq"] = {8806, 824},
5172  ["GreaterFullEqual"] = 8807,
5173  ["NotGreaterFullEqual"] = {8807, 824},
5174  ["gE"] = 8807,
5175  ["geqq"] = 8807,
5176  ["ngE"] = {8807, 824},
5177  ["ngeqq"] = {8807, 824},
5178  ["lnE"] = 8808,
5179  ["lneqq"] = 8808,
5180  ["lvertneqq"] = {8808, 65024},
5181  ["lvnE"] = {8808, 65024},
5182  ["gnE"] = 8809,
5183  ["gneqq"] = 8809,
5184  ["gvertneqq"] = {8809, 65024},
5185  ["gvnE"] = {8809, 65024},
5186  ["Lt"] = 8810,
5187  ["NestedLessLess"] = 8810,
5188  ["NotLessLess"] = {8810, 824},
5189  ["ll"] = 8810,
5190  ["nLt"] = {8810, 8402},
5191  ["nLtv"] = {8810, 824},
5192  ["Gt"] = 8811,
5193  ["NestedGreaterGreater"] = 8811,
5194  ["NotGreaterGreater"] = {8811, 824},
5195  ["gg"] = 8811,
5196  ["nGt"] = {8811, 8402},
5197  ["nGtv"] = {8811, 824},
5198  ["between"] = 8812,
5199  ["twixt"] = 8812,
5200  ["NotCupCap"] = 8813,
5201  ["NotLess"] = 8814,
5202  ["nless"] = 8814,
5203  ["nlt"] = 8814,
5204  ["NotGreater"] = 8815,
5205  ["ngt"] = 8815,
5206  ["ngtr"] = 8815,
5207  ["NotLessEqual"] = 8816,
5208  ["nle"] = 8816,
5209  ["nleq"] = 8816,
5210  ["NotGreaterEqual"] = 8817,
5211  ["nge"] = 8817,
5212  ["ngeq"] = 8817,
5213  ["LessTilde"] = 8818,
5214  ["lesssim"] = 8818,
5215  ["lsim"] = 8818,
5216  ["GreaterTilde"] = 8819,
```

```
5217    ["gsim"] = 8819,
5218    ["gtrsim"] = 8819,
5219    ["NotLessTilde"] = 8820,
5220    ["nlsim"] = 8820,
5221    ["NotGreaterTilde"] = 8821,
5222    ["ngsim"] = 8821,
5223    ["LessGreater"] = 8822,
5224    ["lessgtr"] = 8822,
5225    ["lg"] = 8822,
5226    ["GreaterLess"] = 8823,
5227    ["gl"] = 8823,
5228    ["gtrless"] = 8823,
5229    ["NotLessGreater"] = 8824,
5230    ["ntlg"] = 8824,
5231    ["NotGreaterLess"] = 8825,
5232    ["ntgl"] = 8825,
5233    ["Precedes"] = 8826,
5234    ["pr"] = 8826,
5235    ["prec"] = 8826,
5236    ["Succeeds"] = 8827,
5237    ["sc"] = 8827,
5238    ["succ"] = 8827,
5239    ["PrecedesSlantEqual"] = 8828,
5240    ["prcue"] = 8828,
5241    ["preccurlyeq"] = 8828,
5242    ["SucceedsSlantEqual"] = 8829,
5243    ["sccue"] = 8829,
5244    ["succcurlyeq"] = 8829,
5245    ["PrecedesTilde"] = 8830,
5246    ["precsim"] = 8830,
5247    ["prsim"] = 8830,
5248    ["NotSucceedsTilde"] = {8831, 824},
5249    ["SucceedsTilde"] = 8831,
5250    ["scsim"] = 8831,
5251    ["succsim"] = 8831,
5252    ["NotPrecedes"] = 8832,
5253    ["npr"] = 8832,
5254    ["nprec"] = 8832,
5255    ["NotSucceeds"] = 8833,
5256    ["nsc"] = 8833,
5257    ["nsucc"] = 8833,
5258    ["NotSubset"] = {8834, 8402},
5259    ["nsubset"] = {8834, 8402},
5260    ["sub"] = 8834,
5261    ["subset"] = 8834,
5262    ["vnsub"] = {8834, 8402},
5263    ["NotSuperset"] = {8835, 8402},
```

```
["Superset"] = 8835,
["nsupset"] = {8835, 8402},
["sup"] = 8835,
["supset"] = 8835,
["vnsup"] = {8835, 8402},
["nsub"] = 8836,
["nsup"] = 8837,
["SubsetEqual"] = 8838,
["sube"] = 8838,
["subseteq"] = 8838,
["SupersetEqual"] = 8839,
["supe"] = 8839,
["supseteq"] = 8839,
["NotSubsetEqual"] = 8840,
["nsube"] = 8840,
["nsubseteq"] = 8840,
["NotSupersetEqual"] = 8841,
["nsupe"] = 8841,
["nsupseteq"] = 8841,
["subne"] = 8842,
["subsetneq"] = 8842,
["varsubsetneq"] = {8842, 65024},
["vsubne"] = {8842, 65024},
["supne"] = 8843,
["supsetneq"] = 8843,
["varsupsetneq"] = {8843, 65024},
["vsupne"] = {8843, 65024},
["cupdot"] = 8845,
["UnionPlus"] = 8846,
["uplus"] = 8846,
["NotSquareSubset"] = {8847, 824},
["SquareSubset"] = 8847,
["sqsub"] = 8847,
["sqsubset"] = 8847,
["NotSquareSuperset"] = {8848, 824},
["SquareSuperset"] = 8848,
["sqsup"] = 8848,
["sqsupset"] = 8848,
["SquareSubsetEqual"] = 8849,
["sqsube"] = 8849,
["sqsubseteq"] = 8849,
["SquareSupersetEqual"] = 8850,
["sqsupe"] = 8850,
["sqsupseteq"] = 8850,
["SquareIntersection"] = 8851,
["sqcap"] = 8851,
["sqcaps"] = {8851, 65024},
```

```
5311    ["SquareUnion"] = 8852,
5312    ["sqcup"] = 8852,
5313    ["sqcups"] = {8852, 65024},
5314    ["CirclePlus"] = 8853,
5315    ["oplus"] = 8853,
5316    ["CircleMinus"] = 8854,
5317    ["ominus"] = 8854,
5318    ["CircleTimes"] = 8855,
5319    ["otimes"] = 8855,
5320    ["osol"] = 8856,
5321    ["CircleDot"] = 8857,
5322    ["odot"] = 8857,
5323    ["circledcirc"] = 8858,
5324    ["ocir"] = 8858,
5325    ["circledast"] = 8859,
5326    ["oast"] = 8859,
5327    ["circleddash"] = 8861,
5328    ["odash"] = 8861,
5329    ["boxplus"] = 8862,
5330    ["plusb"] = 8862,
5331    ["boxminus"] = 8863,
5332    ["minusb"] = 8863,
5333    ["boxtimes"] = 8864,
5334    ["timesb"] = 8864,
5335    ["dotsquare"] = 8865,
5336    ["sdotb"] = 8865,
5337    ["RightTee"] = 8866,
5338    ["vdash"] = 8866,
5339    ["LeftTee"] = 8867,
5340    ["dashv"] = 8867,
5341    ["DownTee"] = 8868,
5342    ["top"] = 8868,
5343    ["UpTee"] = 8869,
5344    ["bot"] = 8869,
5345    ["bottom"] = 8869,
5346    ["perp"] = 8869,
5347    ["models"] = 8871,
5348    ["DoubleRightTee"] = 8872,
5349    ["vDash"] = 8872,
5350    ["Vdash"] = 8873,
5351    ["Vvdash"] = 8874,
5352    ["VDash"] = 8875,
5353    ["nvdash"] = 8876,
5354    ["nvDash"] = 8877,
5355    ["nVdash"] = 8878,
5356    ["nVDash"] = 8879,
5357    ["prurel"] = 8880,
```

```
5358   ["LeftTriangle"] = 8882,
5359   ["vartriangleleft"] = 8882,
5360   ["vltri"] = 8882,
5361   ["RightTriangle"] = 8883,
5362   ["vartriangleright"] = 8883,
5363   ["vrtri"] = 8883,
5364   ["LeftTriangleEqual"] = 8884,
5365   ["ltrie"] = 8884,
5366   ["nvltrie"] = {8884, 8402},
5367   ["trianglelefteq"] = 8884,
5368   ["RightTriangleEqual"] = 8885,
5369   ["nvrtrie"] = {8885, 8402},
5370   ["rtrie"] = 8885,
5371   ["trianglerighteq"] = 8885,
5372   ["origof"] = 8886,
5373   ["imof"] = 8887,
5374   ["multimap"] = 8888,
5375   ["mumap"] = 8888,
5376   ["hercon"] = 8889,
5377   ["intcal"] = 8890,
5378   ["intercal"] = 8890,
5379   ["veebar"] = 8891,
5380   ["barvee"] = 8893,
5381   ["angrtvb"] = 8894,
5382   ["lrtri"] = 8895,
5383   ["Wedge"] = 8896,
5384   ["bigwedge"] = 8896,
5385   ["xwedge"] = 8896,
5386   ["Vee"] = 8897,
5387   ["bigvee"] = 8897,
5388   ["xvee"] = 8897,
5389   ["Intersection"] = 8898,
5390   ["bigcap"] = 8898,
5391   ["xcap"] = 8898,
5392   ["Union"] = 8899,
5393   ["bigcup"] = 8899,
5394   ["xcup"] = 8899,
5395   ["Diamond"] = 8900,
5396   ["diam"] = 8900,
5397   ["diamond"] = 8900,
5398   ["sdot"] = 8901,
5399   ["Star"] = 8902,
5400   ["sstarf"] = 8902,
5401   ["divideontimes"] = 8903,
5402   ["divonx"] = 8903,
5403   ["bowtie"] = 8904,
5404   ["ltimes"] = 8905,
```

```
5405    ["rtimes"] = 8906,
5406    ["leftthreetimes"] = 8907,
5407    ["lthree"] = 8907,
5408    ["rightthreetimes"] = 8908,
5409    ["rthree"] = 8908,
5410    ["backsimeq"] = 8909,
5411    ["bsime"] = 8909,
5412    ["curlyvee"] = 8910,
5413    ["cuvee"] = 8910,
5414    ["curlywedge"] = 8911,
5415    ["cuwed"] = 8911,
5416    ["Sub"] = 8912,
5417    ["Subset"] = 8912,
5418    ["Sup"] = 8913,
5419    ["Supset"] = 8913,
5420    ["Cap"] = 8914,
5421    ["Cup"] = 8915,
5422    ["fork"] = 8916,
5423    ["pitchfork"] = 8916,
5424    ["epar"] = 8917,
5425    ["lessdot"] = 8918,
5426    ["ltdot"] = 8918,
5427    ["gtdot"] = 8919,
5428    ["gtrdot"] = 8919,
5429    ["Ll"] = 8920,
5430    ["nLl"] = {8920, 824},
5431    ["Gg"] = 8921,
5432    ["ggg"] = 8921,
5433    ["nGg"] = {8921, 824},
5434    ["LessEqualGreater"] = 8922,
5435    ["leg"] = 8922,
5436    ["lesg"] = {8922, 65024},
5437    ["lesseqgtr"] = 8922,
5438    ["GreaterEqualLess"] = 8923,
5439    ["gel"] = 8923,
5440    ["gesl"] = {8923, 65024},
5441    ["gtreqless"] = 8923,
5442    ["cuepr"] = 8926,
5443    ["curlyeqprec"] = 8926,
5444    ["cuesc"] = 8927,
5445    ["curlyeqsucc"] = 8927,
5446    ["NotPrecedesSlantEqual"] = 8928,
5447    ["nprcue"] = 8928,
5448    ["NotSucceedsSlantEqual"] = 8929,
5449    ["nsccue"] = 8929,
5450    ["NotSquareSubsetEqual"] = 8930,
5451    ["nsqsube"] = 8930,
```

```
5452    ["NotSquareSupersetEqual"] = 8931,
5453    ["nsqsupe"] = 8931,
5454    ["lnsim"] = 8934,
5455    ["gnsim"] = 8935,
5456    ["precnsim"] = 8936,
5457    ["prnsim"] = 8936,
5458    ["scnsim"] = 8937,
5459    ["succnsim"] = 8937,
5460    ["NotLeftTriangle"] = 8938,
5461    ["nltri"] = 8938,
5462    ["ntriangleleft"] = 8938,
5463    ["NotRightTriangle"] = 8939,
5464    ["nrtri"] = 8939,
5465    ["ntriangleright"] = 8939,
5466    ["NotLeftTriangleEqual"] = 8940,
5467    ["nltrie"] = 8940,
5468    ["ntrianglelefteq"] = 8940,
5469    ["NotRightTriangleEqual"] = 8941,
5470    ["nrtrie"] = 8941,
5471    ["ntrianglerighteq"] = 8941,
5472    ["vellip"] = 8942,
5473    ["ctdot"] = 8943,
5474    ["utdot"] = 8944,
5475    ["dtdot"] = 8945,
5476    ["disin"] = 8946,
5477    ["isinsv"] = 8947,
5478    ["isins"] = 8948,
5479    ["isindot"] = 8949,
5480    ["notindot"] = {8949, 824},
5481    ["notinvc"] = 8950,
5482    ["notinvb"] = 8951,
5483    ["isinE"] = 8953,
5484    ["notinE"] = {8953, 824},
5485    ["nisd"] = 8954,
5486    ["xnis"] = 8955,
5487    ["nis"] = 8956,
5488    ["notnivc"] = 8957,
5489    ["notnivb"] = 8958,
5490    ["barwed"] = 8965,
5491    ["barwedge"] = 8965,
5492    ["Barwed"] = 8966,
5493    ["doublebarwedge"] = 8966,
5494    ["LeftCeiling"] = 8968,
5495    ["lceil"] = 8968,
5496    ["RightCeiling"] = 8969,
5497    ["rceil"] = 8969,
5498    ["LeftFloor"] = 8970,
```

```
5499   ["lfloor"] = 8970,
5500   ["RightFloor"] = 8971,
5501   ["rfloor"] = 8971,
5502   ["drcrop"] = 8972,
5503   ["dlcrop"] = 8973,
5504   ["urcrop"] = 8974,
5505   ["ulcrop"] = 8975,
5506   ["bnot"] = 8976,
5507   ["profline"] = 8978,
5508   ["profsurf"] = 8979,
5509   ["telrec"] = 8981,
5510   ["target"] = 8982,
5511   ["ulcorn"] = 8988,
5512   ["ulcorner"] = 8988,
5513   ["urcorn"] = 8989,
5514   ["urcorner"] = 8989,
5515   ["dlcorn"] = 8990,
5516   ["llcorner"] = 8990,
5517   ["drcorn"] = 8991,
5518   ["lrcorner"] = 8991,
5519   ["frown"] = 8994,
5520   ["sfrown"] = 8994,
5521   ["smile"] = 8995,
5522   ["ssmile"] = 8995,
5523   ["cylcty"] = 9005,
5524   ["profalar"] = 9006,
5525   ["topbot"] = 9014,
5526   ["ovbar"] = 9021,
5527   ["solbar"] = 9023,
5528   ["angzarr"] = 9084,
5529   ["lmoust"] = 9136,
5530   ["lmoustache"] = 9136,
5531   ["rmoust"] = 9137,
5532   ["rmoustache"] = 9137,
5533   ["OverBracket"] = 9140,
5534   ["tbrk"] = 9140,
5535   ["UnderBracket"] = 9141,
5536   ["bbrk"] = 9141,
5537   ["bbrktbrk"] = 9142,
5538   ["OverParenthesis"] = 9180,
5539   ["UnderParenthesis"] = 9181,
5540   ["OverBrace"] = 9182,
5541   ["UnderBrace"] = 9183,
5542   ["trpezium"] = 9186,
5543   ["elinters"] = 9191,
5544   ["blank"] = 9251,
5545   ["circledS"] = 9416,
```

```
5546    ["oS"] = 9416,
5547    ["HorizontalLine"] = 9472,
5548    ["boxh"] = 9472,
5549    ["boxv"] = 9474,
5550    ["boxdr"] = 9484,
5551    ["boxdl"] = 9488,
5552    ["boxur"] = 9492,
5553    ["boxul"] = 9496,
5554    ["boxvr"] = 9500,
5555    ["boxvl"] = 9508,
5556    ["boxhd"] = 9516,
5557    ["boxhu"] = 9524,
5558    ["boxvh"] = 9532,
5559    ["boxH"] = 9552,
5560    ["boxV"] = 9553,
5561    ["boxdR"] = 9554,
5562    ["boxDr"] = 9555,
5563    ["boxDR"] = 9556,
5564    ["boxdL"] = 9557,
5565    ["boxDl"] = 9558,
5566    ["boxDL"] = 9559,
5567    ["boxuR"] = 9560,
5568    ["boxUr"] = 9561,
5569    ["boxUR"] = 9562,
5570    ["boxuL"] = 9563,
5571    ["boxUl"] = 9564,
5572    ["boxUL"] = 9565,
5573    ["boxvR"] = 9566,
5574    ["boxVr"] = 9567,
5575    ["boxVR"] = 9568,
5576    ["boxvL"] = 9569,
5577    ["boxVl"] = 9570,
5578    ["boxVL"] = 9571,
5579    ["boxHd"] = 9572,
5580    ["boxhD"] = 9573,
5581    ["boxHD"] = 9574,
5582    ["boxHu"] = 9575,
5583    ["boxhU"] = 9576,
5584    ["boxHU"] = 9577,
5585    ["boxvH"] = 9578,
5586    ["boxVh"] = 9579,
5587    ["boxVH"] = 9580,
5588    ["uhblk"] = 9600,
5589    ["lhblk"] = 9604,
5590    ["block"] = 9608,
5591    ["blk14"] = 9617,
5592    ["blk12"] = 9618,
```

```
5593   ["blk34"] = 9619,
5594   ["Square"] = 9633,
5595   ["squ"] = 9633,
5596   ["square"] = 9633,
5597   ["FilledVerySmallSquare"] = 9642,
5598   ["blacksquare"] = 9642,
5599   ["squarf"] = 9642,
5600   ["squf"] = 9642,
5601   ["EmptyVerySmallSquare"] = 9643,
5602   ["rect"] = 9645,
5603   ["marker"] = 9646,
5604   ["fltns"] = 9649,
5605   ["bigtriangleup"] = 9651,
5606   ["xutri"] = 9651,
5607   ["blacktriangle"] = 9652,
5608   ["utrif"] = 9652,
5609   ["triangle"] = 9653,
5610   ["utri"] = 9653,
5611   ["blacktriangleright"] = 9656,
5612   ["rtrif"] = 9656,
5613   ["rtri"] = 9657,
5614   ["triangleright"] = 9657,
5615   ["bigtriangledown"] = 9661,
5616   ["xdtri"] = 9661,
5617   ["blacktriangledown"] = 9662,
5618   ["dtrif"] = 9662,
5619   ["dtri"] = 9663,
5620   ["triangledown"] = 9663,
5621   ["blacktriangleleft"] = 9666,
5622   ["ltrif"] = 9666,
5623   ["ltri"] = 9667,
5624   ["triangleleft"] = 9667,
5625   ["loz"] = 9674,
5626   ["lozenge"] = 9674,
5627   ["cir"] = 9675,
5628   ["tridot"] = 9708,
5629   ["bigcirc"] = 9711,
5630   ["xcirc"] = 9711,
5631   ["ultri"] = 9720,
5632   ["urtri"] = 9721,
5633   ["lltri"] = 9722,
5634   ["EmptySmallSquare"] = 9723,
5635   ["FilledSmallSquare"] = 9724,
5636   ["bigstar"] = 9733,
5637   ["starf"] = 9733,
5638   ["star"] = 9734,
5639   ["phone"] = 9742,
```

```
  ["female"] = 9792,
  ["male"] = 9794,
  ["spades"] = 9824,
  ["spadesuit"] = 9824,
  ["clubs"] = 9827,
  ["clubsuit"] = 9827,
  ["hearts"] = 9829,
  ["heartsuit"] = 9829,
  ["diamondsuit"] = 9830,
  ["diams"] = 9830,
  ["sung"] = 9834,
  ["flat"] = 9837,
  ["natur"] = 9838,
  ["natural"] = 9838,
  ["sharp"] = 9839,
  ["check"] = 10003,
  ["checkmark"] = 10003,
  ["cross"] = 10007,
  ["malt"] = 10016,
  ["maltese"] = 10016,
  ["sext"] = 10038,
  ["VerticalSeparator"] = 10072,
  ["lbbrk"] = 10098,
  ["rbbrk"] = 10099,
  ["bsolhsub"] = 10184,
  ["suphsol"] = 10185,
  ["LeftDoubleBracket"] = 10214,
  ["lobrk"] = 10214,
  ["RightDoubleBracket"] = 10215,
  ["robrk"] = 10215,
  ["LeftAngleBracket"] = 10216,
  ["lang"] = 10216,
  ["langle"] = 10216,
  ["RightAngleBracket"] = 10217,
  ["rang"] = 10217,
  ["rangle"] = 10217,
  ["Lang"] = 10218,
  ["Rang"] = 10219,
  ["loang"] = 10220,
  ["roang"] = 10221,
  ["LongLeftArrow"] = 10229,
  ["longleftarrow"] = 10229,
  ["xlarr"] = 10229,
  ["LongRightArrow"] = 10230,
  ["longrightarrow"] = 10230,
  ["xrarr"] = 10230,
  ["LongLeftRightArrow"] = 10231,
```

```
  ["longleftrightarrow"] = 10231,
  ["xharr"] = 10231,
  ["DoubleLongLeftArrow"] = 10232,
  ["Longleftarrow"] = 10232,
  ["xlArr"] = 10232,
  ["DoubleLongRightArrow"] = 10233,
  ["Longrightarrow"] = 10233,
  ["xrArr"] = 10233,
  ["DoubleLongLeftRightArrow"] = 10234,
  ["Longleftrightarrow"] = 10234,
  ["xhArr"] = 10234,
  ["longmapsto"] = 10236,
  ["xmap"] = 10236,
  ["dzigrarr"] = 10239,
  ["nvlArr"] = 10498,
  ["nvrArr"] = 10499,
  ["nvHarr"] = 10500,
  ["Map"] = 10501,
  ["lbarr"] = 10508,
  ["bkarow"] = 10509,
  ["rbarr"] = 10509,
  ["lBarr"] = 10510,
  ["dbkarow"] = 10511,
  ["rBarr"] = 10511,
  ["RBarr"] = 10512,
  ["drbkarow"] = 10512,
  ["DDotrahd"] = 10513,
  ["UpArrowBar"] = 10514,
  ["DownArrowBar"] = 10515,
  ["Rarrtl"] = 10518,
  ["latail"] = 10521,
  ["ratail"] = 10522,
  ["lAtail"] = 10523,
  ["rAtail"] = 10524,
  ["larrfs"] = 10525,
  ["rarrfs"] = 10526,
  ["larrbfs"] = 10527,
  ["rarrbfs"] = 10528,
  ["nwarhk"] = 10531,
  ["nearhk"] = 10532,
  ["hksearow"] = 10533,
  ["searhk"] = 10533,
  ["hkswarow"] = 10534,
  ["swarhk"] = 10534,
  ["nwnear"] = 10535,
  ["nesear"] = 10536,
  ["toea"] = 10536,
```

```
5734    ["seswar"] = 10537,
5735    ["tosa"] = 10537,
5736    ["swnwar"] = 10538,
5737    ["nrarrc"] = {10547, 824},
5738    ["rarrc"] = 10547,
5739    ["cudarrr"] = 10549,
5740    ["ldca"] = 10550,
5741    ["rdca"] = 10551,
5742    ["cudarrl"] = 10552,
5743    ["larrpl"] = 10553,
5744    ["curarrm"] = 10556,
5745    ["cularrp"] = 10557,
5746    ["rarrpl"] = 10565,
5747    ["harrcir"] = 10568,
5748    ["Uarrocir"] = 10569,
5749    ["lurdshar"] = 10570,
5750    ["ldrushar"] = 10571,
5751    ["LeftRightVector"] = 10574,
5752    ["RightUpDownVector"] = 10575,
5753    ["DownLeftRightVector"] = 10576,
5754    ["LeftUpDownVector"] = 10577,
5755    ["LeftVectorBar"] = 10578,
5756    ["RightVectorBar"] = 10579,
5757    ["RightUpVectorBar"] = 10580,
5758    ["RightDownVectorBar"] = 10581,
5759    ["DownLeftVectorBar"] = 10582,
5760    ["DownRightVectorBar"] = 10583,
5761    ["LeftUpVectorBar"] = 10584,
5762    ["LeftDownVectorBar"] = 10585,
5763    ["LeftTeeVector"] = 10586,
5764    ["RightTeeVector"] = 10587,
5765    ["RightUpTeeVector"] = 10588,
5766    ["RightDownTeeVector"] = 10589,
5767    ["DownLeftTeeVector"] = 10590,
5768    ["DownRightTeeVector"] = 10591,
5769    ["LeftUpTeeVector"] = 10592,
5770    ["LeftDownTeeVector"] = 10593,
5771    ["lHar"] = 10594,
5772    ["uHar"] = 10595,
5773    ["rHar"] = 10596,
5774    ["dHar"] = 10597,
5775    ["luruhar"] = 10598,
5776    ["ldrdhar"] = 10599,
5777    ["ruluhar"] = 10600,
5778    ["rdldhar"] = 10601,
5779    ["lharul"] = 10602,
5780    ["llhard"] = 10603,
```

```
5781    ["rharul"] = 10604,
5782    ["lrhard"] = 10605,
5783    ["UpEquilibrium"] = 10606,
5784    ["udhar"] = 10606,
5785    ["ReverseUpEquilibrium"] = 10607,
5786    ["duhar"] = 10607,
5787    ["RoundImplies"] = 10608,
5788    ["erarr"] = 10609,
5789    ["simrarr"] = 10610,
5790    ["larrsim"] = 10611,
5791    ["rarrsim"] = 10612,
5792    ["rarrap"] = 10613,
5793    ["ltlarr"] = 10614,
5794    ["gtrarr"] = 10616,
5795    ["subrarr"] = 10617,
5796    ["suplarr"] = 10619,
5797    ["lfisht"] = 10620,
5798    ["rfisht"] = 10621,
5799    ["ufisht"] = 10622,
5800    ["dfisht"] = 10623,
5801    ["lopar"] = 10629,
5802    ["ropar"] = 10630,
5803    ["lbrke"] = 10635,
5804    ["rbrke"] = 10636,
5805    ["lbrkslu"] = 10637,
5806    ["rbrksld"] = 10638,
5807    ["lbrksld"] = 10639,
5808    ["rbrkslu"] = 10640,
5809    ["langd"] = 10641,
5810    ["rangd"] = 10642,
5811    ["lparlt"] = 10643,
5812    ["rpargt"] = 10644,
5813    ["gtlPar"] = 10645,
5814    ["ltrPar"] = 10646,
5815    ["vzigzag"] = 10650,
5816    ["vangrt"] = 10652,
5817    ["angrtvbd"] = 10653,
5818    ["ange"] = 10660,
5819    ["range"] = 10661,
5820    ["dwangle"] = 10662,
5821    ["uwangle"] = 10663,
5822    ["angmsdaa"] = 10664,
5823    ["angmsdab"] = 10665,
5824    ["angmsdac"] = 10666,
5825    ["angmsdad"] = 10667,
5826    ["angmsdae"] = 10668,
5827    ["angmsdaf"] = 10669,
```

```
5828   ["angmsdag"] = 10670,
5829   ["angmsdah"] = 10671,
5830   ["bemptyv"] = 10672,
5831   ["demptyv"] = 10673,
5832   ["cemptyv"] = 10674,
5833   ["raemptyv"] = 10675,
5834   ["laemptyv"] = 10676,
5835   ["ohbar"] = 10677,
5836   ["omid"] = 10678,
5837   ["opar"] = 10679,
5838   ["operp"] = 10681,
5839   ["olcross"] = 10683,
5840   ["odsold"] = 10684,
5841   ["olcir"] = 10686,
5842   ["ofcir"] = 10687,
5843   ["olt"] = 10688,
5844   ["ogt"] = 10689,
5845   ["cirscir"] = 10690,
5846   ["cirE"] = 10691,
5847   ["solb"] = 10692,
5848   ["bsolb"] = 10693,
5849   ["boxbox"] = 10697,
5850   ["trisb"] = 10701,
5851   ["rtriltri"] = 10702,
5852   ["LeftTriangleBar"] = 10703,
5853   ["NotLeftTriangleBar"] = {10703, 824},
5854   ["NotRightTriangleBar"] = {10704, 824},
5855   ["RightTriangleBar"] = 10704,
5856   ["iinfin"] = 10716,
5857   ["infintie"] = 10717,
5858   ["nvinfin"] = 10718,
5859   ["eparsl"] = 10723,
5860   ["smeparsl"] = 10724,
5861   ["eqvparsl"] = 10725,
5862   ["blacklozenge"] = 10731,
5863   ["lozf"] = 10731,
5864   ["RuleDelayed"] = 10740,
5865   ["dsol"] = 10742,
5866   ["bigodot"] = 10752,
5867   ["xodot"] = 10752,
5868   ["bigoplus"] = 10753,
5869   ["xoplus"] = 10753,
5870   ["bigotimes"] = 10754,
5871   ["xotime"] = 10754,
5872   ["biguplus"] = 10756,
5873   ["xuplus"] = 10756,
5874   ["bigsqcup"] = 10758,
```

```
5875    ["xsqcup"] = 10758,
5876    ["iiiint"] = 10764,
5877    ["qint"] = 10764,
5878    ["fpartint"] = 10765,
5879    ["cirfnint"] = 10768,
5880    ["awint"] = 10769,
5881    ["rppolint"] = 10770,
5882    ["scpolint"] = 10771,
5883    ["npolint"] = 10772,
5884    ["pointint"] = 10773,
5885    ["quatint"] = 10774,
5886    ["intlarhk"] = 10775,
5887    ["pluscir"] = 10786,
5888    ["plusacir"] = 10787,
5889    ["simplus"] = 10788,
5890    ["plusdu"] = 10789,
5891    ["plussim"] = 10790,
5892    ["plustwo"] = 10791,
5893    ["mcomma"] = 10793,
5894    ["minusdu"] = 10794,
5895    ["loplus"] = 10797,
5896    ["roplus"] = 10798,
5897    ["Cross"] = 10799,
5898    ["timesd"] = 10800,
5899    ["timesbar"] = 10801,
5900    ["smashp"] = 10803,
5901    ["lotimes"] = 10804,
5902    ["rotimes"] = 10805,
5903    ["otimesas"] = 10806,
5904    ["Otimes"] = 10807,
5905    ["odiv"] = 10808,
5906    ["triplus"] = 10809,
5907    ["triminus"] = 10810,
5908    ["tritime"] = 10811,
5909    ["intprod"] = 10812,
5910    ["iprod"] = 10812,
5911    ["amalg"] = 10815,
5912    ["capdot"] = 10816,
5913    ["ncup"] = 10818,
5914    ["ncap"] = 10819,
5915    ["capand"] = 10820,
5916    ["cupor"] = 10821,
5917    ["cupcap"] = 10822,
5918    ["capcup"] = 10823,
5919    ["cupbrcap"] = 10824,
5920    ["capbrcup"] = 10825,
5921    ["cupcup"] = 10826,
```

```
["capcap"] = 10827,
["ccups"] = 10828,
["ccaps"] = 10829,
["ccupssm"] = 10832,
["And"] = 10835,
["Or"] = 10836,
["andand"] = 10837,
["oror"] = 10838,
["orslope"] = 10839,
["andslope"] = 10840,
["andv"] = 10842,
["orv"] = 10843,
["andd"] = 10844,
["ord"] = 10845,
["wedbar"] = 10847,
["sdote"] = 10854,
["simdot"] = 10858,
["congdot"] = 10861,
["ncongdot"] = {10861, 824},
["easter"] = 10862,
["apacir"] = 10863,
["apE"] = 10864,
["napE"] = {10864, 824},
["eplus"] = 10865,
["pluse"] = 10866,
["Esim"] = 10867,
["Colone"] = 10868,
["Equal"] = 10869,
["ddotseq"] = 10871,
["eDDot"] = 10871,
["equivDD"] = 10872,
["ltcir"] = 10873,
["gtcir"] = 10874,
["ltquest"] = 10875,
["gtquest"] = 10876,
["LessSlantEqual"] = 10877,
["NotLessSlantEqual"] = {10877, 824},
["leqslant"] = 10877,
["les"] = 10877,
["nleqslant"] = {10877, 824},
["nles"] = {10877, 824},
["GreaterSlantEqual"] = 10878,
["NotGreaterSlantEqual"] = {10878, 824},
["geqslant"] = 10878,
["ges"] = 10878,
["ngeqslant"] = {10878, 824},
["nges"] = {10878, 824},
```

```
5969    ["lesdot"] = 10879,
5970    ["gesdot"] = 10880,
5971    ["lesdoto"] = 10881,
5972    ["gesdoto"] = 10882,
5973    ["lesdotor"] = 10883,
5974    ["gesdotol"] = 10884,
5975    ["lap"] = 10885,
5976    ["lessapprox"] = 10885,
5977    ["gap"] = 10886,
5978    ["gtrapprox"] = 10886,
5979    ["lne"] = 10887,
5980    ["lneq"] = 10887,
5981    ["gne"] = 10888,
5982    ["gneq"] = 10888,
5983    ["lnap"] = 10889,
5984    ["lnapprox"] = 10889,
5985    ["gnap"] = 10890,
5986    ["gnapprox"] = 10890,
5987    ["lEg"] = 10891,
5988    ["lesseqqgtr"] = 10891,
5989    ["gEl"] = 10892,
5990    ["gtreqqless"] = 10892,
5991    ["lsime"] = 10893,
5992    ["gsime"] = 10894,
5993    ["lsimg"] = 10895,
5994    ["gsiml"] = 10896,
5995    ["lgE"] = 10897,
5996    ["glE"] = 10898,
5997    ["lesges"] = 10899,
5998    ["gesles"] = 10900,
5999    ["els"] = 10901,
6000    ["eqslantless"] = 10901,
6001    ["egs"] = 10902,
6002    ["eqslantgtr"] = 10902,
6003    ["elsdot"] = 10903,
6004    ["egsdot"] = 10904,
6005    ["el"] = 10905,
6006    ["eg"] = 10906,
6007    ["siml"] = 10909,
6008    ["simg"] = 10910,
6009    ["simlE"] = 10911,
6010    ["simgE"] = 10912,
6011    ["LessLess"] = 10913,
6012    ["NotNestedLessLess"] = {10913, 824},
6013    ["GreaterGreater"] = 10914,
6014    ["NotNestedGreaterGreater"] = {10914, 824},
6015    ["glj"] = 10916,
```

```
["gla"] = 10917,
["ltcc"] = 10918,
["gtcc"] = 10919,
["lescc"] = 10920,
["gescc"] = 10921,
["smt"] = 10922,
["lat"] = 10923,
["smte"] = 10924,
["smtes"] = {10924, 65024},
["late"] = 10925,
["lates"] = {10925, 65024},
["bumpE"] = 10926,
["NotPrecedesEqual"] = {10927, 824},
["PrecedesEqual"] = 10927,
["npre"] = {10927, 824},
["npreceq"] = {10927, 824},
["pre"] = 10927,
["preceq"] = 10927,
["NotSucceedsEqual"] = {10928, 824},
["SucceedsEqual"] = 10928,
["nsce"] = {10928, 824},
["nsucceq"] = {10928, 824},
["sce"] = 10928,
["succeq"] = 10928,
["prE"] = 10931,
["scE"] = 10932,
["precneqq"] = 10933,
["prnE"] = 10933,
["scnE"] = 10934,
["succneqq"] = 10934,
["prap"] = 10935,
["precapprox"] = 10935,
["scap"] = 10936,
["succapprox"] = 10936,
["precnapprox"] = 10937,
["prnap"] = 10937,
["scnap"] = 10938,
["succnapprox"] = 10938,
["Pr"] = 10939,
["Sc"] = 10940,
["subdot"] = 10941,
["supdot"] = 10942,
["subplus"] = 10943,
["supplus"] = 10944,
["submult"] = 10945,
["supmult"] = 10946,
["subedot"] = 10947,
```

```
["supedot"] = 10948,
["nsubE"] = {10949, 824},
["nsubseteqq"] = {10949, 824},
["subE"] = 10949,
["subseteqq"] = 10949,
["nsupE"] = {10950, 824},
["nsupseteqq"] = {10950, 824},
["supE"] = 10950,
["supseteqq"] = 10950,
["subsim"] = 10951,
["supsim"] = 10952,
["subnE"] = 10955,
["subsetneqq"] = 10955,
["varsubsetneqq"] = {10955, 65024},
["vsubnE"] = {10955, 65024},
["supnE"] = 10956,
["supsetneqq"] = 10956,
["varsupsetneqq"] = {10956, 65024},
["vsupnE"] = {10956, 65024},
["csub"] = 10959,
["csup"] = 10960,
["csube"] = 10961,
["csupe"] = 10962,
["subsup"] = 10963,
["supsub"] = 10964,
["subsub"] = 10965,
["supsup"] = 10966,
["suphsub"] = 10967,
["supdsub"] = 10968,
["forkv"] = 10969,
["topfork"] = 10970,
["mlcp"] = 10971,
["Dashv"] = 10980,
["DoubleLeftTee"] = 10980,
["Vdashl"] = 10982,
["Barv"] = 10983,
["vBar"] = 10984,
["vBarv"] = 10985,
["Vbar"] = 10987,
["Not"] = 10988,
["bNot"] = 10989,
["rnmid"] = 10990,
["cirmid"] = 10991,
["midcir"] = 10992,
["topcir"] = 10993,
["nhpar"] = 10994,
["parsim"] = 10995,
```

```
  ["nparsl"] = {11005, 8421},
  ["parsl"] = 11005,
  ["fflig"] = 64256,
  ["filig"] = 64257,
  ["fllig"] = 64258,
  ["ffilig"] = 64259,
  ["ffllig"] = 64260,
  ["Ascr"] = 119964,
  ["Cscr"] = 119966,
  ["Dscr"] = 119967,
  ["Gscr"] = 119970,
  ["Jscr"] = 119973,
  ["Kscr"] = 119974,
  ["Nscr"] = 119977,
  ["Oscr"] = 119978,
  ["Pscr"] = 119979,
  ["Qscr"] = 119980,
  ["Sscr"] = 119982,
  ["Tscr"] = 119983,
  ["Uscr"] = 119984,
  ["Vscr"] = 119985,
  ["Wscr"] = 119986,
  ["Xscr"] = 119987,
  ["Yscr"] = 119988,
  ["Zscr"] = 119989,
  ["ascr"] = 119990,
  ["bscr"] = 119991,
  ["cscr"] = 119992,
  ["dscr"] = 119993,
  ["fscr"] = 119995,
  ["hscr"] = 119997,
  ["iscr"] = 119998,
  ["jscr"] = 119999,
  ["kscr"] = 120000,
  ["lscr"] = 120001,
  ["mscr"] = 120002,
  ["nscr"] = 120003,
  ["pscr"] = 120005,
  ["qscr"] = 120006,
  ["rscr"] = 120007,
  ["sscr"] = 120008,
  ["tscr"] = 120009,
  ["uscr"] = 120010,
  ["vscr"] = 120011,
  ["wscr"] = 120012,
  ["xscr"] = 120013,
  ["yscr"] = 120014,
```

```
  ["zscr"] = 120015,
  ["Afr"] = 120068,
  ["Bfr"] = 120069,
  ["Dfr"] = 120071,
  ["Efr"] = 120072,
  ["Ffr"] = 120073,
  ["Gfr"] = 120074,
  ["Jfr"] = 120077,
  ["Kfr"] = 120078,
  ["Lfr"] = 120079,
  ["Mfr"] = 120080,
  ["Nfr"] = 120081,
  ["Ofr"] = 120082,
  ["Pfr"] = 120083,
  ["Qfr"] = 120084,
  ["Sfr"] = 120086,
  ["Tfr"] = 120087,
  ["Ufr"] = 120088,
  ["Vfr"] = 120089,
  ["Wfr"] = 120090,
  ["Xfr"] = 120091,
  ["Yfr"] = 120092,
  ["afr"] = 120094,
  ["bfr"] = 120095,
  ["cfr"] = 120096,
  ["dfr"] = 120097,
  ["efr"] = 120098,
  ["ffr"] = 120099,
  ["gfr"] = 120100,
  ["hfr"] = 120101,
  ["ifr"] = 120102,
  ["jfr"] = 120103,
  ["kfr"] = 120104,
  ["lfr"] = 120105,
  ["mfr"] = 120106,
  ["nfr"] = 120107,
  ["ofr"] = 120108,
  ["pfr"] = 120109,
  ["qfr"] = 120110,
  ["rfr"] = 120111,
  ["sfr"] = 120112,
  ["tfr"] = 120113,
  ["ufr"] = 120114,
  ["vfr"] = 120115,
  ["wfr"] = 120116,
  ["xfr"] = 120117,
  ["yfr"] = 120118,
```

```
  ["zfr"] = 120119,
  ["Aopf"] = 120120,
  ["Bopf"] = 120121,
  ["Dopf"] = 120123,
  ["Eopf"] = 120124,
  ["Fopf"] = 120125,
  ["Gopf"] = 120126,
  ["Iopf"] = 120128,
  ["Jopf"] = 120129,
  ["Kopf"] = 120130,
  ["Lopf"] = 120131,
  ["Mopf"] = 120132,
  ["Oopf"] = 120134,
  ["Sopf"] = 120138,
  ["Topf"] = 120139,
  ["Uopf"] = 120140,
  ["Vopf"] = 120141,
  ["Wopf"] = 120142,
  ["Xopf"] = 120143,
  ["Yopf"] = 120144,
  ["aopf"] = 120146,
  ["bopf"] = 120147,
  ["copf"] = 120148,
  ["dopf"] = 120149,
  ["eopf"] = 120150,
  ["fopf"] = 120151,
  ["gopf"] = 120152,
  ["hopf"] = 120153,
  ["iopf"] = 120154,
  ["jopf"] = 120155,
  ["kopf"] = 120156,
  ["lopf"] = 120157,
  ["mopf"] = 120158,
  ["nopf"] = 120159,
  ["oopf"] = 120160,
  ["popf"] = 120161,
  ["qopf"] = 120162,
  ["ropf"] = 120163,
  ["sopf"] = 120164,
  ["topf"] = 120165,
  ["uopf"] = 120166,
  ["vopf"] = 120167,
  ["wopf"] = 120168,
  ["xopf"] = 120169,
  ["yopf"] = 120170,
  ["zopf"] = 120171,
}
```

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

```
function entities.dec_entity(s)
  local n = tonumber(s)
  if n == nil then
    return "&#" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.hex_entity(s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#x" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```
function entities.hex_entity_with_x_char(x, s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#" .. x .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.char_entity(s)
  local code_points = character_entities[s]
  if code_points == nil then
    return "&" .. s .. ";"
  end
  if type(code_points) ~= 'table' then
    code_points = {code_points}
  end
  local char_table = {}
    for _, code_point in ipairs(code_points) do
      table.insert(char_table, unicode.utf8.char(code_point))
    end
  return table.concat(char_table)
end
```

3.1.3 Plain TEX Writer

This section documents the `writer` object, which implements the routines for producing the TEX output. The object is an amalgamate of the generic, TEX, LATEX 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.

```
6286 M.writer = {}
```

The `writer.new` method creates and returns a new TEX writer object associated with the Lua interface options (see Section 2.1.3) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

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

```
6287 function M.writer.new(options)
6288   local self = {}
```

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

```
6289   self.options = options
```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```
6290   self.flatten_inlines = false
```

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

```
6291   local slice_specifiers = {}
6292   for specifier in options.slice:gmatch("[^%s]+") do
6293     table.insert(slice_specifiers, specifier)
6294   end
6295
6296   if #slice_specifiers == 2 then
6297     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6298     local slice_begin_type = self.slice_begin:sub(1, 1)
6299     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6300       self.slice_begin = "^" .. self.slice_begin
6301     end
6302     local slice_end_type = self.slice_end:sub(1, 1)
6303     if slice_end_type ~= "^" and slice_end_type ~= "$" then
```

```
      self.slice_end = "$" .. self.slice_end
    end
  elseif #slice_specifiers == 1 then
    self.slice_begin = "^" .. slice_specifiers[1]
    self.slice_end = "$" .. slice_specifiers[1]
  end

  self.slice_begin_type = self.slice_begin:sub(1, 1)
  self.slice_begin_identifier = self.slice_begin:sub(2) or ""
  self.slice_end_type = self.slice_end:sub(1, 1)
  self.slice_end_identifier = self.slice_end:sub(2) or ""

  if self.slice_begin == "^" and self.slice_end ~= "^" then
    self.is_writing = true
  else
    self.is_writing = false
  end
```

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

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

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

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

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

```
  function self.plain(s)
    return s
  end
```

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

```
  function self.paragraph(s)
    if not self.is_writing then return "" end
    return s
  end
```

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

```
  self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
  function self.interblocksep()
    if not self.is_writing then return "" end
    return self.interblocksep_text
  end
```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```
  self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
```

```
  function self.paragraphsep()
    if not self.is_writing then return "" end
    return self.paragraphsep_text
  end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
  self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
  function self.undosep()
    if not self.is_writing then return "" end
    return self.undosep_text
  end
```

Define `writer->soft_line_break` as the output format of a soft line break.

```
  self.soft_line_break = function()
    if self.flatten_inlines then return "\n" end
    return "\\markdownRendererSoftLineBreak\n{}"
  end
```

Define `writer->hard_line_break` as the output format of a hard line break.

```
  self.hard_line_break = function()
    if self.flatten_inlines then return "\n" end
    return "\\markdownRendererHardLineBreak\n{}"
  end
```

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

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

Define `writer->thematic_break` as the output format of a thematic break.

```
  function self.thematic_break()
    if not self.is_writing then return "" end
    return "\\markdownRendererThematicBreak{}"
  end
```

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

```
  self.escaped_uri_chars = {
    ["{"] = "\\markdownRendererLeftBrace{}",
    ["}"] = "\\markdownRendererRightBrace{}",
    ["\\"] = "\\markdownRendererBackslash{}",
    ["\r"] = " ",
    ["\n"] = " ",
  }
  self.escaped_minimal_strings = {
    ["^^"] = "\\markdownRendererCircumflex"
          .. "\\markdownRendererCircumflex ",
    ["☒"] = "\\markdownRendererTickedBox{}",
    ["☐"] = "\\markdownRendererHalfTickedBox{}",
```

```
    ["▫"] = "\\markdownRendererUntickedBox{}",
    [entities.hex_entity('FFFD')]
      = "\\markdownRendererReplacementCharacter{}",
  }
```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```
  self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
  self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
  self.escaped_chars = {
    ["{"] = "\\markdownRendererLeftBrace{}",
    ["}"] = "\\markdownRendererRightBrace{}",
    ["%"] = "\\markdownRendererPercentSign{}",
    ["\\"] = "\\markdownRendererBackslash{}",
    ["#"] = "\\markdownRendererHash{}",
    ["$"] = "\\markdownRendererDollarSign{}",
    ["&"] = "\\markdownRendererAmpersand{}",
    ["_"] = "\\markdownRendererUnderscore{}",
    ["^"] = "\\markdownRendererCircumflex{}",
    ["~"] = "\\markdownRendererTilde{}",
    ["|"] = "\\markdownRendererPipe{}",
    [entities.hex_entity('0000')]
      = "\\markdownRendererReplacementCharacter{}",
  }
```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal_` tables to create the `escape_typographic_text`, `escape_programmatic_text`, and `escape_minimal` local escaper functions.

```
  local function create_escaper(char_escapes, string_escapes)
    local escape = util.escaper(char_escapes, string_escapes)
    return function(s)
      if self.flatten_inlines then return s end
      return escape(s)
    end
  end
  local escape_typographic_text = create_escaper(
    self.escaped_chars, self.escaped_strings)
  local escape_programmatic_text = create_escaper(
    self.escaped_uri_chars, self.escaped_minimal_strings)
  local escape_minimal = create_escaper(
    {}, self.escaped_minimal_strings)
```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.

- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.
- `writer->infostring` transforms a fence code infostring.

```
6404   self.escape = escape_typographic_text
6405   self.math = escape_minimal
6406   if options.hybrid then
6407     self.identifier = escape_minimal
6408     self.string = escape_minimal
6409     self.uri = escape_minimal
6410     self.infostring = escape_minimal
6411   else
6412     self.identifier = escape_programmatic_text
6413     self.string = escape_typographic_text
6414     self.uri = escape_programmatic_text
6415     self.infostring = escape_programmatic_text
6416   end
```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```
6417   function self.warning(t, m)
6418     return {"\\markdownRendererWarning{", self.escape(t), "}{",
6419             escape_minimal(t), "}{", self.escape(m or ""), "}{",
6420             escape_minimal(m or ""), "}"}
6421   end
```

Define `writer->error` as a function that will transform an input error text `t` with optional more error text `m` to the output format.

```
6422   function self.error(t, m)
6423     return {"\\markdownRendererError{", self.escape(t), "}{",
6424             escape_minimal(t), "}{", self.escape(m or ""), "}{",
6425             escape_minimal(m or ""), "}"}
6426   end
```

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

```
6427   function self.code(s, attributes)
6428     if self.flatten_inlines then return s end
6429     local buf = {}
6430     if attributes ~= nil then
6431       table.insert(buf,
6432                    "\\markdownRendererCodeSpanAttributeContextBegin\n")
6433       table.insert(buf, self.attributes(attributes))
```

```
    end
    table.insert(buf,
                 {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererCodeSpanAttributeContextEnd{}")
    end
    return buf
  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, `tit` to the title of the link, and `attributes` to optional attributes.

```
  function self.link(lab, src, tit, attributes)
    if self.flatten_inlines then return lab end
    local buf = {}
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererLinkAttributeContextBegin\n")
      table.insert(buf, self.attributes(attributes))
    end
    table.insert(buf, {"\\markdownRendererLink{",lab,"}",
                       "{",self.escape(src),"}",
                       "{",self.uri(src),"}",
                       "{",self.string(tit or ""),"}"})
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererLinkAttributeContextEnd{}")
    end
    return buf
  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, `tit` to the title of the image, and `attributes` to optional attributes.

```
  function self.image(lab, src, tit, attributes)
    if self.flatten_inlines then return lab end
    local buf = {}
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererImageAttributeContextBegin\n")
      table.insert(buf, self.attributes(attributes))
    end
    table.insert(buf, {"\\markdownRendererImage{",lab,"}",
                       "{",self.string(src),"}",
                       "{",self.uri(src),"}",
                       "{",self.string(tit or ""),"}"})
```

```
6473      if attributes ~= nil then
6474        table.insert(buf,
6475                     "\\markdownRendererImageAttributeContextEnd{}")
6476      end
6477      return buf
6478    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.

```
6479    function self.bulletlist(items,tight)
6480      if not self.is_writing then return "" end
6481      local buffer = {}
6482      for _,item in ipairs(items) do
6483        if item ~= "" then
6484          buffer[#buffer + 1] = self.bulletitem(item)
6485        end
6486      end
6487      local contents = util.intersperse(buffer,"\n")
6488      if tight and options.tightLists then
6489        return {"\\markdownRendererUlBeginTight\n",contents,
6490          "\n\\markdownRendererUlEndTight "}
6491      else
6492        return {"\\markdownRendererUlBegin\n",contents,
6493          "\n\\markdownRendererUlEnd "}
6494      end
6495    end
```

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

```
6496    function self.bulletitem(s)
6497      return {"\\markdownRendererUlItem ",s,
6498              "\\markdownRendererUlItemEnd "}
6499    end
```

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

```
6500    function self.orderedlist(items,tight,startnum)
6501      if not self.is_writing then return "" end
6502      local buffer = {}
6503      local num = startnum
6504      for _,item in ipairs(items) do
6505        if item ~= "" then
6506          buffer[#buffer + 1] = self.ordereditem(item,num)
6507        end
6508        if num ~= nil and item ~= "" then
```

```
6509          num = num + 1
6510        end
6511      end
6512      local contents = util.intersperse(buffer,"\n")
6513      if tight and options.tightLists then
6514        return {"\\markdownRendererOlBeginTight\n",contents,
6515                "\n\\markdownRendererOlEndTight "}
6516      else
6517        return {"\\markdownRendererOlBegin\n",contents,
6518                "\n\\markdownRendererOlEnd "}
6519      end
6520    end
```

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

```
6521    function self.ordereditem(s,num)
6522      if num ~= nil then
6523        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
6524                "\\markdownRendererOlItemEnd "}
6525      else
6526        return {"\\markdownRendererOlItem ",s,
6527                "\\markdownRendererOlItemEnd "}
6528      end
6529    end
```

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

```
6530    function self.inline_html_comment(contents)
6531      if self.flatten_inlines then return contents end
6532      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
6533    end
```

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

```
6534    function self.inline_html_tag(contents)
6535      if self.flatten_inlines then return contents end
6536      return {"\\markdownRendererInlineHtmlTag{",
6537              self.string(contents),"}"}
6538    end
```

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

```
6539    function self.block_html_element(s)
6540      if not self.is_writing then return "" end
```

```
6541      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6542      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
6543    end
```

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

```
6544    function self.emphasis(s)
6545      if self.flatten_inlines then return s end
6546      return {"\\markdownRendererEmphasis{",s,"}"}
6547    end
```

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

```
6548    function self.tickbox(f)
6549      if f == 1.0 then
6550        return "☒ "
6551      elseif f == 0.0 then
6552        return "▫ "
6553      else
6554        return "⊡ "
6555      end
6556    end
```

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

```
6557    function self.strong(s)
6558      if self.flatten_inlines then return s end
6559      return {"\\markdownRendererStrongEmphasis{",s,"}"}
6560    end
```

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

```
6561    function self.blockquote(s)
6562      if not self.is_writing then return "" end
6563      return {"\\markdownRendererBlockQuoteBegin\n",s,
6564        "\\markdownRendererBlockQuoteEnd "}
6565    end
```

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

```
6566    function self.verbatim(s)
6567      if not self.is_writing then return "" end
6568      s = s:gsub("\n$", "")
6569      local name = util.cache_verbatim(options.cacheDir, s)
6570      return {"\\markdownRendererInputVerbatim{",name,"}"}
6571    end
```

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

```
function self.document(d)
  local buf = {"\\markdownRendererDocumentBegin\n"}

  -- warn against the `hybrid` option
  if options.hybrid then
    local text = "The `hybrid` option has been soft-deprecated."
    local more = "Consider using one of the following better options "
              .. "for mixing TeX and markdown: `contentBlocks`, "
              .. "`rawAttribute`, `texComments`, `texMathDollars`, "
              .. "`texMathSingleBackslash`, and "
              .. "`texMathDoubleBackslash`. "
              .. "For more information, see the user manual at "
              .. "<https://witiko.github.io/markdown/>."
    table.insert(buf, self.warning(text, more))
  end

  -- insert the text of the document
  table.insert(buf, d)

  -- pop all attributes
  table.insert(buf, self.pop_attributes())

  table.insert(buf, "\\markdownRendererDocumentEnd")

  return buf
end
```

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```
local seen_identifiers = {}
local key_value_regex = "([^= ]+)%s*=%s*(.*)"
local function normalize_attributes(attributes, auto_identifiers)
  -- normalize attributes
  local normalized_attributes = {}
  local has_explicit_identifiers = false
  local key, value
  for _, attribute in ipairs(attributes or {}) do
    if attribute:sub(1, 1) == "#" then
      table.insert(normalized_attributes, attribute)
      has_explicit_identifiers = true
      seen_identifiers[attribute:sub(2)] = true
    elseif attribute:sub(1, 1) == "." then
      table.insert(normalized_attributes, attribute)
    else
      key, value = attribute:match(key_value_regex)
      if key:lower() == "id" then
        table.insert(normalized_attributes, "#" .. value)
```

```
      elseif key:lower() == "class" then
        local classes = {}
        for class in value:gmatch("%S+") do
          table.insert(classes, class)
        end
        table.sort(classes)
        for _, class in ipairs(classes) do
          table.insert(normalized_attributes, "." .. class)
        end
      else
        table.insert(normalized_attributes, attribute)
      end
    end
  end

  -- if no explicit identifiers exist, add auto identifiers
  if not has_explicit_identifiers and auto_identifiers ~= nil then
    local seen_auto_identifiers = {}
    for _, auto_identifier in ipairs(auto_identifiers) do
      if seen_auto_identifiers[auto_identifier] == nil then
        seen_auto_identifiers[auto_identifier] = true
        if seen_identifiers[auto_identifier] == nil then
          seen_identifiers[auto_identifier] = true
          table.insert(normalized_attributes,
                       "#" .. auto_identifier)
        else
          local auto_identifier_number = 1
          while true do
            local numbered_auto_identifier = auto_identifier .. "-"
                                             .. auto_identifier_number
            if seen_identifiers[numbered_auto_identifier] == nil then
              seen_identifiers[numbered_auto_identifier] = true
              table.insert(normalized_attributes,
                           "#" .. numbered_auto_identifier)
              break
            end
            auto_identifier_number = auto_identifier_number + 1
          end
        end
      end
    end
  end

  -- sort and deduplicate normalized attributes
  table.sort(normalized_attributes)
  local seen_normalized_attributes = {}
  local deduplicated_normalized_attributes = {}
```

```
6663    for _, attribute in ipairs(normalized_attributes) do
6664      if seen_normalized_attributes[attribute] == nil then
6665        seen_normalized_attributes[attribute] = true
6666        table.insert(deduplicated_normalized_attributes, attribute)
6667      end
6668    end
6669
6670    return deduplicated_normalized_attributes
6671  end
6672
6673  function self.attributes(attributes, should_normalize_attributes)
6674    local normalized_attributes
6675    if should_normalize_attributes == false then
6676      normalized_attributes = attributes
6677    else
6678      normalized_attributes = normalize_attributes(attributes)
6679    end
6680
6681    local buf = {}
6682    local key, value
6683    for _, attribute in ipairs(normalized_attributes) do
6684      if attribute:sub(1, 1) == "#" then
6685        table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6686                           attribute:sub(2), "}"})
6687      elseif attribute:sub(1, 1) == "." then
6688        table.insert(buf, {"\\markdownRendererAttributeClassName{",
6689                           attribute:sub(2), "}"})
6690      else
6691        key, value = attribute:match(key_value_regex)
6692        table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6693                           key, "}{", value, "}"})
6694      end
6695    end
6696
6697    return buf
6698  end
```

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

```
6699  self.active_attributes = {}
```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```
6700  self.attribute_type_levels = {}
6701  setmetatable(self.attribute_type_levels,
6702               { __index = function() return 0 end })
```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```
local function apply_attributes()
  local buf = {}
  for i = 1, #self.active_attributes do
    local start_output = self.active_attributes[i][3]
    if start_output ~= nil then
      table.insert(buf, start_output)
    end
  end
  return buf
end

local function tear_down_attributes()
  local buf = {}
  for i = #self.active_attributes, 1, -1 do
    local end_output = self.active_attributes[i][4]
    if end_output ~= nil then
      table.insert(buf, end_output)
    end
  end
  return buf
end
```

The `writer->push_attributes` method adds `attributes` of type `attribute_type` to `writer->active_attributes`. The `start_output` string is used to construct a rope that will be returned by this function, together with output produced as a result of slicing (see `slice`). The `end_output` string is stored together with `attributes` and is used to construct the return value of the `writer->pop_attributes` method.

```
function self.push_attributes(attribute_type, attributes,
                              start_output, end_output)
  local attribute_type_level
    = self.attribute_type_levels[attribute_type]
  self.attribute_type_levels[attribute_type]
    = attribute_type_level + 1

  -- index attributes in a hash table for easy lookup
  attributes = attributes or {}
  for i = 1, #attributes do
    attributes[attributes[i]] = true
  end

  local buf = {}
  -- handle slicing
  if attributes["#" .. self.slice_end_identifier] ~= nil and
     self.slice_end_type == "^" then
```

```
6741        if self.is_writing then
6742          table.insert(buf, self.undosep())
6743          table.insert(buf, tear_down_attributes())
6744        end
6745        self.is_writing = false
6746      end
6747      if attributes["#" .. self.slice_begin_identifier] ~= nil and
6748         self.slice_begin_type == "^" then
6749        table.insert(buf, apply_attributes())
6750        self.is_writing = true
6751      end
6752      if self.is_writing and start_output ~= nil then
6753        table.insert(buf, start_output)
6754      end
6755      table.insert(self.active_attributes,
6756                   {attribute_type, attributes,
6757                    start_output, end_output})
6758      return buf
6759    end
6760
```

The writer->pop_attributes method removes the most current of active block-level attributes from writer->active_attributes until attributes of type attribute_type have been removed. The method returns a rope constructed from the end_output string specified in the calls of writer->push_attributes that produced the most current attributes, and also from output produced as a result of slicing (see slice).

```
6761    function self.pop_attributes(attribute_type)
6762      local buf = {}
6763      -- pop attributes until we find attributes of correct type
6764      -- or until no attributes remain
6765      local current_attribute_type = false
6766      while current_attribute_type ~= attribute_type and
6767            #self.active_attributes > 0 do
6768        local attributes, _, end_output
6769        current_attribute_type, attributes, _, end_output = table.unpack(
6770          self.active_attributes[#self.active_attributes])
6771        local attribute_type_level
6772          = self.attribute_type_levels[current_attribute_type]
6773        self.attribute_type_levels[current_attribute_type]
6774          = attribute_type_level - 1
6775        if self.is_writing and end_output ~= nil then
6776          table.insert(buf, end_output)
6777        end
6778        table.remove(self.active_attributes, #self.active_attributes)
6779        -- handle slicing
6780        if attributes["#" .. self.slice_end_identifier] ~= nil
```

```
6781          and self.slice_end_type == "$" then
6782        if self.is_writing then
6783          table.insert(buf, self.undosep())
6784          table.insert(buf, tear_down_attributes())
6785        end
6786        self.is_writing = false
6787      end
6788      if attributes["#" .. self.slice_begin_identifier] ~= nil and
6789         self.slice_begin_type == "$" then
6790        self.is_writing = true
6791        table.insert(buf, apply_attributes())
6792      end
6793    end
6794    return buf
6795  end
```

Create an auto identifier string by stripping and converting characters from string `s`.

```
6796  local function create_auto_identifier(s)
6797    local buffer = {}
6798    local prev_space = false
6799    local letter_found = false
6800    local normalized_s = s
6801    if not options.unicodeNormalization
6802       or options.unicodeNormalizationForm ~= "nfc" then
6803      normalized_s = uni_algos.normalize.NFC(normalized_s)
6804    end
6805
6806    for _, code in utf8.codes(normalized_s) do
6807      local char = utf8.char(code)
6808
6809      -- Remove everything up to the first letter.
6810      if not letter_found then
6811        local is_letter = unicode.utf8.match(char, "%a")
6812        if is_letter then
6813          letter_found = true
6814        else
6815          goto continue
6816        end
6817      end
6818
6819      -- Remove all non-alphanumeric characters, except underscores,
6820      -- hyphens, and periods.
6821      if not unicode.utf8.match(char, "[%w_%-%.%s]") then
6822        goto continue
6823      end
6824
6825      -- Replace all spaces and newlines with hyphens.
6826      if unicode.utf8.match(char, "[%s\n]") then
```

```
6827          char = "-"
6828          if prev_space then
6829            goto continue
6830          else
6831            prev_space = true
6832          end
6833        else
6834          -- Convert all alphabetic characters to lowercase.
6835          char = unicode.utf8.lower(char)
6836          prev_space = false
6837        end
6838
6839        table.insert(buffer, char)
6840
6841        ::continue::
6842      end
6843
6844      if prev_space then
6845        table.remove(buffer)
6846      end
6847
6848      local identifier = #buffer == 0 and "section"
6849                      or table.concat(buffer, "")
6850      return identifier
6851    end
```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string s.

```
6852    local function create_gfm_auto_identifier(s)
6853      local buffer = {}
6854      local prev_space = false
6855      local letter_found = false
6856      local normalized_s = s
6857      if not options.unicodeNormalization
6858         or options.unicodeNormalizationForm ~= "nfc" then
6859        normalized_s = uni_algos.normalize.NFC(normalized_s)
6860      end
6861
6862      for _, code in utf8.codes(normalized_s) do
6863        local char = utf8.char(code)
6864
6865        -- Remove everything up to the first non-space.
6866        if not letter_found then
6867          local is_letter = unicode.utf8.match(char, "%S")
6868          if is_letter then
6869            letter_found = true
6870          else
```

```
        goto continue
      end
    end

    -- Remove all non-alphanumeric characters, except underscores
    -- and hyphens.
    if not unicode.utf8.match(char, "[%w_%-%s]") then
      prev_space = false
      goto continue
    end

    -- Replace all spaces and newlines with hyphens.
    if unicode.utf8.match(char, "[%s\n]") then
      char = "-"
      if prev_space then
        goto continue
      else
        prev_space = true
      end
    else
      -- Convert all alphabetic characters to lowercase.
      char = unicode.utf8.lower(char)
      prev_space = false
    end

    table.insert(buffer, char)

    ::continue::
  end

  if prev_space then
    table.remove(buffer)
  end

  local identifier = #buffer == 0 and "section"
                  or table.concat(buffer, "")
  return identifier
end
```

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

```
self.secbegin_text = "\\markdownRendererSectionBegin\n"
self.secend_text = "\n\\markdownRendererSectionEnd "
function self.heading(s, level, attributes)
  local buf = {}
  local flat_text, inlines = table.unpack(s)

```

```
6915      -- push empty attributes for implied sections
6916      while self.attribute_type_levels["heading"] < level - 1 do
6917        table.insert(buf,
6918                     self.push_attributes("heading",
6919                                          nil,
6920                                          self.secbegin_text,
6921                                          self.secend_text))
6922      end
6923
6924      -- pop attributes for sections that have ended
6925      while self.attribute_type_levels["heading"] >= level do
6926        table.insert(buf, self.pop_attributes("heading"))
6927      end
6928
6929      -- construct attributes for the new section
6930      local auto_identifiers = {}
6931      if self.options.autoIdentifiers then
6932        table.insert(auto_identifiers, create_auto_identifier(flat_text))
6933      end
6934      if self.options.gfmAutoIdentifiers then
6935        table.insert(auto_identifiers,
6936                     create_gfm_auto_identifier(flat_text))
6937      end
6938      local normalized_attributes = normalize_attributes(attributes,
6939                                                         auto_identifiers)
6940
6941      -- push attributes for the new section
6942      local start_output = {}
6943      local end_output = {}
6944      table.insert(start_output, self.secbegin_text)
6945      table.insert(end_output, self.secend_text)
6946
6947      table.insert(buf, self.push_attributes("heading",
6948                                             normalized_attributes,
6949                                             start_output,
6950                                             end_output))
6951      assert(self.attribute_type_levels["heading"] == level)
6952
6953      -- render the heading and its attributes
6954      if self.is_writing and #normalized_attributes > 0 then
6955        table.insert(buf,
6956                     "\\markdownRendererHeaderAttributeContextBegin\n")
6957        table.insert(buf, self.attributes(normalized_attributes, false))
6958      end
6959
6960      local cmd
6961      level = level + options.shiftHeadings
```

```
    if level <= 1 then
      cmd = "\\markdownRendererHeadingOne"
    elseif level == 2 then
      cmd = "\\markdownRendererHeadingTwo"
    elseif level == 3 then
      cmd = "\\markdownRendererHeadingThree"
    elseif level == 4 then
      cmd = "\\markdownRendererHeadingFour"
    elseif level == 5 then
      cmd = "\\markdownRendererHeadingFive"
    elseif level >= 6 then
      cmd = "\\markdownRendererHeadingSix"
    else
      cmd = ""
    end
    if self.is_writing then
      table.insert(buf, {cmd, "{", inlines, "}"})
    end

    if self.is_writing and #normalized_attributes > 0 then
      table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
    end

    return buf
  end
```

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

```
  function self.get_state()
    return {
      is_writing=self.is_writing,
      flatten_inlines=self.flatten_inlines,
      active_attributes={table.unpack(self.active_attributes)},
    }
  end
```

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

```
  function self.set_state(s)
    local previous_state = self.get_state()
    for key, value in pairs(s) do
      self[key] = value
    end
    return previous_state
  end
```

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

```
  function self.defer_call(f)
    local previous_state = self.get_state()
    return function(...)
      local state = self.set_state(previous_state)
      local return_value = f(...)
      self.set_state(state)
      return return_value
    end
  end

  return self
end
```

3.1.4 Parsers

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

```
local parsers                 = {}
```

3.1.4.1 Basic Parsers

```
parsers.percent               = P("%")
parsers.at                    = P("@")
parsers.comma                 = P(",")
parsers.asterisk              = P("*")
parsers.dash                  = P("-")
parsers.plus                  = P("+")
parsers.underscore            = P("_")
parsers.period                = P(".")
parsers.hash                  = P("#")
parsers.dollar                = P("$")
parsers.ampersand             = P("&")
parsers.backtick              = P("`")
parsers.less                  = P("<")
parsers.more                  = P(">")
parsers.space                 = P(" ")
parsers.squote                = P("'")
parsers.dquote                = P('"')
parsers.lparent               = P("(")
parsers.rparent               = P(")")
parsers.lbracket              = P("[")
parsers.rbracket              = P("]")
parsers.lbrace                = P("{")
```

```
parsers.rbrace               = P("}")
parsers.circumflex           = P("^")
parsers.slash                = P("/")
parsers.equal                = P("=")
parsers.colon                = P(":")
parsers.semicolon            = P(";")
parsers.exclamation          = P("!")
parsers.pipe                 = P("|")
parsers.tilde                = P("~")
parsers.backslash            = P("\\")
parsers.tab                  = P("\t")
parsers.newline              = P("\n")

parsers.digit                = R("09")
parsers.hexdigit             = R("09","af","AF")
parsers.letter               = R("AZ","az")
parsers.alphanumeric         = R("AZ","az","09")
parsers.keyword              = parsers.letter
                             * (parsers.alphanumeric + parsers.dash)^0

parsers.doubleasterisks      = P("**")
parsers.doubleunderscores    = P("__")
parsers.doubletildes         = P("~~")
parsers.fourspaces           = P("    ")

parsers.any                  = P(1)
parsers.succeed              = P(true)
parsers.fail                 = P(false)

parsers.internal_punctuation = S(":;,.?")
parsers.ascii_punctuation    = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
```

3.1.5 Unicode punctuation

This section documents the Unicode punctuation[33] recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

```
                    (CommonMark Spec, Version 0.31.2 (2024-01-28))
```

All code from this section will be executed during the compilation of the Markdown package and the standard output will be stored in a file named `markdown-unicode-data.lua` with the precompiled parser of Unicode punctuation.

[33] See https://spec.commonmark.org/0.31.2/#unicode-punctuation-character.

```
;(function()
  local pathname = assert(kpse.find_file("UnicodeData.txt"),
    [[Could not locate file "UnicodeData.txt"]])
  local file = assert(io.open(pathname, "r"),
    [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct a prefix tree of UTF-8 encodings for all codepoints of a given code length.

```
  local prefix_trees = {}
  for line in file:lines() do
    local codepoint, major_category = line:match("^(%x+);[^;]*;(%a)")
    if major_category == "P" or major_category == "S" then
      local code = unicode.utf8.char(tonumber(codepoint, 16))
      if prefix_trees[#code] == nil then
        prefix_trees[#code] = {}
      end
      local node = prefix_trees[#code]
      for i = 1, #code do
        local byte = code:sub(i, i)
        if i < #code then
          if node[byte] == nil then
            node[byte] = {}
          end
          node = node[byte]
        else
          table.insert(node, byte)
        end
      end
    end
  end
  assert(file:close())

```

Next, we will construct a parser out of the prefix tree.

```
  local function depth_first_search(node, path, visit, leave)
    visit(node, path)
    for label, child in pairs(node) do
      if type(child) == "table" then
        depth_first_search(child, path .. label, visit, leave)
      else
        visit(child, path)
      end
    end
    leave(node, path)
  end

  print("M.punctuation = {}")
  print("local P = lpeg.P")
```

```
  print("-- luacheck: push no max line length")
  for length, prefix_tree in pairs(prefix_trees) do
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if type(node) == "string" then
        local suffix
        if node == "]" then
          suffix = "P('" .. node .. "')"
        else
          suffix = "P([[" .. node .. "]])"
        end
        if subparsers[path] ~= nil then
          subparsers[path] = subparsers[path] .. " + " .. suffix
        else
          subparsers[path] = suffix
        end
      end
    end, function(_, path)
      if #path > 0 then
        local byte = path:sub(#path, #path)
        local parent_path = path:sub(1, #path-1)
        local prefix
        if byte == "]" then
          prefix = "P('" .. byte .. "')"
        else
          prefix = "P([[" .. byte .. "]])"
        end
        local suffix
        if subparsers[path]:find(" %+ ") then
          suffix = prefix .. " * (" .. subparsers[path] .. ")"
        else
          suffix = prefix .. " * " .. subparsers[path]
        end
        if subparsers[parent_path] ~= nil then
          subparsers[parent_path] = subparsers[parent_path]
                                    .. " + " .. suffix
        else
          subparsers[parent_path] = suffix
        end
      else
        print("M.punctuation[" .. length .. "] = " .. subparsers[path])
      end
    end)
  end
  print("-- luacheck: pop")
end)()
print("return M")
```

Back in the Markdown package, we will load the precompiled parser of Unicode punctuation.

```
local unicode_data = require("markdown-unicode-data")
if metadata.version ~= unicode_data.metadata.version then
  util.warning(
    "markdown.lua " .. metadata.version .. " used with " ..
    "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
  )
end
parsers.punctuation = unicode_data.punctuation

parsers.escapable              = parsers.ascii_punctuation
parsers.anyescaped             = parsers.backslash / ""
                               * parsers.escapable
                               + parsers.any

parsers.spacechar              = S("\t ")
parsers.spacing                = S(" \n\r\t")
parsers.nonspacechar           = parsers.any - parsers.spacing
parsers.optionalspace          = parsers.spacechar^0

parsers.normalchar             = parsers.any - (V("SpecialChar")
                                               + parsers.spacing)
parsers.eof                    = -parsers.any
parsers.nonindentspace         = parsers.space^-3 * - parsers.spacechar
parsers.indent                 = parsers.space^-3 * parsers.tab
                               + parsers.fourspaces / ""
parsers.linechar               = P(1 - parsers.newline)

parsers.blankline              = parsers.optionalspace
                               * parsers.newline / "\n"
parsers.blanklines             = parsers.blankline^0
parsers.skipblanklines         = ( parsers.optionalspace
                                 * parsers.newline)^0
parsers.indentedline           = parsers.indent    /""
                               * C( parsers.linechar^1
                                  * parsers.newline^-1)
parsers.optionallyindentedline = parsers.indent^-1 /""
                               * C( parsers.linechar^1
                                  * parsers.newline^-1)
parsers.sp                     = parsers.spacing^0
parsers.spnl                   = parsers.optionalspace
                               * ( parsers.newline
                                 * parsers.optionalspace)^-1
parsers.line                   = parsers.linechar^0 * parsers.newline
parsers.nonemptyline           = parsers.line - parsers.blankline
```

3.1.5.1 Parsers Used for Indentation

```
7202
7203 parsers.leader      = parsers.space^-3
7204
```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```
7205 local function has_trail(indent_table)
7206   return indent_table ~= nil and
7207     indent_table.trail ~= nil and
7208     next(indent_table.trail) ~= nil
7209 end
7210
```

Check if indent table `indent_table` has any indents.

```
7211 local function has_indents(indent_table)
7212   return indent_table ~= nil and
7213     indent_table.indents ~= nil and
7214     next(indent_table.indents) ~= nil
7215 end
7216
```

Add a trail `trail_info` to the indent table `indent_table`.

```
7217 local function add_trail(indent_table, trail_info)
7218   indent_table.trail = trail_info
7219   return indent_table
7220 end
7221
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
7222 local function remove_trail(indent_table)
7223   indent_table.trail = nil
7224   return indent_table
7225 end
7226
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
7227 local function update_indent_table(indent_table, new_indent, add)
7228   indent_table = remove_trail(indent_table)
7229
7230   if not has_indents(indent_table) then
7231     indent_table.indents = {}
7232   end
7233
7234
7235   if add then
7236     indent_table.indents[#indent_table.indents + 1] = new_indent
7237   else
7238     if indent_table.indents[#indent_table.indents].name
7239        == new_indent.name then
```

```
      indent_table.indents[#indent_table.indents] = nil
    end
  end

  return indent_table
end

```

Remove an indent by its name `name`.

```
local function remove_indent(name)
  local remove_indent_level =
    function(s, i, indent_table) -- luacheck: ignore s i
      indent_table = update_indent_table(indent_table, {name=name},
                                         false)
      return true, indent_table
    end

  return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
end

```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```
local function process_starter_spacing(indent, spacing,
                                       minimum, left_strip_length)
  left_strip_length = left_strip_length or 0

  local count = 0
  local tab_value = 4 - (indent) % 4

  local code_started, minimum_found = false, false
  local code_start, minimum_remainder = "", ""

  local left_total_stripped = 0
  local full_remainder = ""

  if spacing ~= nil then
    for i = 1, #spacing do
      local character = spacing:sub(i, i)

      if character == "\t" then
        count = count + tab_value
        tab_value = 4
      elseif character == " " then
```

```
        count = count + 1
        tab_value = 4 - (1 - tab_value) % 4
      end

      if (left_strip_length ~= 0) then
        local possible_to_strip = math.min(count, left_strip_length)
        count = count - possible_to_strip
        left_strip_length = left_strip_length - possible_to_strip
        left_total_stripped = left_total_stripped + possible_to_strip
      else
        full_remainder =  full_remainder .. character
      end

      if (minimum_found) then
        minimum_remainder = minimum_remainder .. character
      elseif (count >= minimum) then
        minimum_found = true
        minimum_remainder = minimum_remainder
                          .. string.rep(" ", count - minimum)
      end

      if (code_started) then
        code_start = code_start .. character
      elseif (count >= minimum + 4) then
        code_started = true
        code_start = code_start
                   .. string.rep(" ", count - (minimum + 4))
      end
    end
  end

  local remainder
  if (code_started) then
    remainder = code_start
  else
    remainder = string.rep(" ", count - minimum)
  end

  local is_minimum = count >= minimum
  return {
    is_code = code_started,
    remainder = remainder,
    left_total_stripped = left_total_stripped,
    is_minimum = is_minimum,
    minimum_remainder = minimum_remainder,
    total_length = count,
    full_remainder = full_remainder
```

```
  }
end

```

Count the total width of all indents in the indent table `indent_table`.

```
local function count_indent_tab_level(indent_table)
  local count = 0
  if not has_indents(indent_table) then
    return count
  end

  for i=1, #indent_table.indents do
    count = count + indent_table.indents[i].length
  end
  return count
end

```

Count the total width of a delimiter `delimiter`.

```
local function total_delimiter_length(delimiter)
  local count = 0
  if type(delimiter) == "string" then return #delimiter end
  for _, value in pairs(delimiter) do
    count = count + total_delimiter_length(value)
  end
  return count
end

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```
local function process_starter_indent(_, _, indent_table, starter,
                                       is_blank, indent_type, breakable)
  local last_trail = starter[1]
  local delimiter = starter[2]
  local raw_new_trail = starter[3]

  if indent_type == "bq" and not breakable then
    indent_table.ignore_blockquote_blank = true
  end

  if has_trail(indent_table) then
    local trail = indent_table.trail
    if trail.is_code then
      return false
    end
    last_trail = trail.remainder
  else
```

```
    local sp = process_starter_spacing(0, last_trail, 0, 0)

    if sp.is_code then
      return false
    end
    last_trail = sp.remainder
  end

  local preceding_indentation = count_indent_tab_level(indent_table) % 4
  local last_trail_length = #last_trail
  local delimiter_length = total_delimiter_length(delimiter)

  local total_indent_level = preceding_indentation + last_trail_length
                           + delimiter_length

  local sp = {}
  if not is_blank then
    sp = process_starter_spacing(total_indent_level, raw_new_trail,
                                 0, 1)
  end

  local del_trail_length = sp.left_total_stripped
  if is_blank then
    del_trail_length = 1
  elseif not sp.is_code then
    del_trail_length = del_trail_length + #sp.remainder
  end

  local indent_length = last_trail_length + delimiter_length
                      + del_trail_length
  local new_indent_info = {name=indent_type, length=indent_length}

  indent_table = update_indent_table(indent_table, new_indent_info,
                                     true)
  indent_table = add_trail(indent_table,
                           {is_code=sp.is_code,
                            remainder=sp.remainder,
                            total_length=sp.total_length,
                            full_remainder=sp.full_remainder})

  return true, indent_table
end
```

Return the pattern corresponding with the indent name `name`.

```
local function decode_pattern(name)
  local delimeter = parsers.succeed
  if name == "bq" then
```

```
    delimeter = parsers.more
  end

  return C(parsers.optionalspace) * C(delimeter)
      * C(parsers.optionalspace) * Cp()
end

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```
local function left_blank_starter(indent_table)
  local blank_starter_index

  if not has_indents(indent_table) then
    return
  end

  for i = #indent_table.indents,1,-1 do
    local value = indent_table.indents[i]
    if value.name == "li" then
      blank_starter_index = i
    else
      break
    end
  end

  return blank_starter_index
end

```

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```
local function traverse_indent(s, i, indent_table, is_optional,
                               is_blank, current_line_indents)
  local new_index = i

  local preceding_indentation = 0
  local current_trail = {}

  local blank_starter = left_blank_starter(indent_table)

  if current_line_indents == nil then
    current_line_indents = {}
  end
```

```
7451
7452  for index = 1,#indent_table.indents do
7453    local value = indent_table.indents[index]
7454    local pattern = decode_pattern(value.name)
7455
7456    -- match decoded pattern
7457    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7458    if new_indent_info == nil then
7459      local blankline_end = lpeg.match(
7460        Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7461      if is_optional or not indent_table.ignore_blockquote_blank
7462         or not blankline_end then
7463        return is_optional, new_index, current_trail,
7464               current_line_indents
7465      end
7466
7467      return traverse_indent(s, tonumber(blankline_end.pos),
7468                             indent_table, is_optional, is_blank,
7469                             current_line_indents)
7470    end
7471
7472    local raw_last_trail = new_indent_info[1]
7473    local delimiter = new_indent_info[2]
7474    local raw_new_trail = new_indent_info[3]
7475    local next_index = new_indent_info[4]
7476
7477    local space_only = delimiter == ""
7478
7479    -- check previous trail
7480    if not space_only and next(current_trail) == nil then
7481      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7482      current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7483                       total_length=sp.total_length,
7484                       full_remainder=sp.full_remainder}
7485    end
7486
7487    if next(current_trail) ~= nil then
7488      if not space_only and current_trail.is_code then
7489        return is_optional, new_index, current_trail,
7490               current_line_indents
7491      end
7492      if current_trail.internal_remainder ~= nil then
7493        raw_last_trail = current_trail.internal_remainder
7494      end
7495    end
7496
7497    local raw_last_trail_length = 0
```

```
    local delimiter_length = 0

    if not space_only then
      delimiter_length = #delimiter
      raw_last_trail_length = #raw_last_trail
    end

    local total_indent_level = preceding_indentation
                             + raw_last_trail_length + delimiter_length

    local spacing_to_process
    local minimum = 0
    local left_strip_length = 0

    if not space_only then
      spacing_to_process = raw_new_trail
      left_strip_length = 1
    else
      spacing_to_process = raw_last_trail
      minimum = value.length
    end

    local sp = process_starter_spacing(total_indent_level,
                                       spacing_to_process, minimum,
                                       left_strip_length)

    if space_only and not sp.is_minimum then
      return is_optional or (is_blank and blank_starter <= index),
             new_index, current_trail, current_line_indents
    end

    local indent_length = raw_last_trail_length + delimiter_length
                        + sp.left_total_stripped

    -- update info for the next pattern
    if not space_only then
      preceding_indentation = preceding_indentation + indent_length
    else
      preceding_indentation = preceding_indentation + value.length
    end

    current_trail = {is_code=sp.is_code, remainder=sp.remainder,
                     internal_remainder=sp.minimum_remainder,
                     total_length=sp.total_length,
                     full_remainder=sp.full_remainder}

    current_line_indents[#current_line_indents + 1] = new_indent_info
```

```
    new_index = next_index
  end

  return true, new_index, current_trail, current_line_indents
end

```

Check if a code trail is expected.

```
local function check_trail(expect_code, is_code)
  return (expect_code and is_code) or (not expect_code and not is_code)
end

```

Check if the current trail of the `indent_table` would produce code if it is expected `expect_code` or it would not if it is not. If there is no trail, process and check the current spacing `spacing`.

```
local check_trail_joined =
  function(s, i, indent_table, -- luacheck: ignore s i
           spacing, expect_code, omit_remainder)
    local is_code
    local remainder

    if has_trail(indent_table) then
      local trail = indent_table.trail
      is_code = trail.is_code
      if is_code then
        remainder = trail.remainder
      else
        remainder = trail.full_remainder
      end
    else
      local sp = process_starter_spacing(0, spacing, 0, 0)
      is_code = sp.is_code
      if is_code then
        remainder = sp.remainder
      else
        remainder = sp.full_remainder
      end
    end

    local result = check_trail(expect_code, is_code)
    if omit_remainder then
      return result
    end
    return result, remainder
  end

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
local check_trail_length =
  function(s, i, indent_table, -- luacheck: ignore s i
           spacing, min, max)
    local trail

    if has_trail(indent_table) then
      trail = indent_table.trail
    else
      trail = process_starter_spacing(0, spacing, 0, 0)
    end

    local total_length = trail.total_length
    if total_length == nil then
      return false
    end

    return min <= total_length and total_length <= max
  end

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
local function check_continuation_indentation(s, i, indent_table,
                                              is_optional, is_blank)
  if not has_indents(indent_table) then
    return true
  end

  local passes, new_index, current_trail, current_line_indents =
    traverse_indent(s, i, indent_table, is_optional, is_blank)

  if passes then
    indent_table.current_line_indents = current_line_indents
    indent_table = add_trail(indent_table, current_trail)
    return new_index, indent_table
  end
  return false
end

```

Get name of the last indent from the `indent_table`.

```
local function get_last_indent_name(indent_table)
  if has_indents(indent_table) then
    return indent_table.indents[#indent_table.indents].name
  end
end

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
local function remove_remainder_if_blank(indent_table, remainder)
  if get_last_indent_name(indent_table) == "li" then
    return ""
  end
  return remainder
end

```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
local check_trail_type =
  function(s, i, -- luacheck: ignore s i
           trail, spacing, trail_type)
    if trail == nil then
      trail = process_starter_spacing(0, spacing, 0, 0)
    end

    if trail_type == "non-code" then
      return check_trail(false, trail.is_code)
    end
    if trail_type == "code" then
      return check_trail(true, trail.is_code)
    end
    if trail_type == "full-code" then
      if (trail.is_code) then
        return i, trail.remainder
      end
      return i, ""
    end
    if trail_type == "full-any" then
      return i, trail.internal_remainder
    end
  end

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
local trail_freezing =
  function(s, i, -- luacheck: ignore s i
           indent_table, is_freezing)
    if is_freezing then
      if indent_table.is_trail_frozen then
        indent_table.trail = indent_table.frozen_trail
      else
        indent_table.frozen_trail = indent_table.trail
        indent_table.is_trail_frozen = true
```

```
      end
    else
      indent_table.frozen_trail = nil
      indent_table.is_trail_frozen = false
    end
    return true, indent_table
  end

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```
local check_continuation_indentation_and_trail =
  function (s, i, indent_table, is_optional, is_blank, trail_type,
            reset_rem, omit_remainder)
    if not has_indents(indent_table) then
      local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
                                           * Cp(), s, i)
      local result, remainder = check_trail_type(s, i,
        indent_table.trail, spacing, trail_type)
      if remainder == nil then
        if result then
          return new_index
        end
        return false
      end
      if result then
        return new_index, remainder
      end
      return false
    end

    local passes, new_index, current_trail = traverse_indent(s, i,
      indent_table, is_optional, is_blank)

    if passes then
      local spacing
      if current_trail == nil then
        local newer_spacing, newer_index = lpeg.match(
          C(parsers.spacechar^0) * Cp(), s, i)
        current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
        new_index = newer_index
        spacing = newer_spacing
      else
        spacing = current_trail.remainder
      end
      local result, remainder = check_trail_type(s, new_index,
```

```
        current_trail, spacing, trail_type)
      if remainder == nil or omit_remainder then
        if result then
          return new_index
        end
        return false
      end

      if is_blank and reset_rem then
        remainder = remove_remainder_if_blank(indent_table, remainder)
      end
      if result then
        return new_index, remainder
      end
      return false
    end
    return false
  end

```

The following patterns check whitespace indentation at the start of a block.

```
parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
                         * Cc(false), check_trail_joined)

parsers.check_trail_no_rem = Cmt( Cb("indent_info")
                                * C(parsers.spacechar^0) * Cc(false)
                                * Cc(true), check_trail_joined)

parsers.check_code_trail  = Cmt( Cb("indent_info")
                               * C(parsers.spacechar^0)
                               * Cc(true), check_trail_joined)

parsers.check_trail_length_range  = function(min, max)
  return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
            * Cc(max), check_trail_length)
end

parsers.check_trail_length = function(n)
  return parsers.check_trail_length_range(n, n)
end

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```
parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
                         * Cc(true), trail_freezing), "indent_info")

parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
```

```
                                trail_freezing), "indent_info")

```

The following patterns check indentation in continuation lines as defined by the container start.

```
parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
                                   check_continuation_indentation)

parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
                                    check_continuation_indentation)

parsers.check_minimal_blank_indent
  = Cmt( Cb("indent_info") * Cc(false)
       * Cc(true)
       , check_continuation_indentation)

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

parsers.check_minimal_indent_and_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("code") * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_full_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_any_trail_no_rem =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
```

```
      , check_continuation_indentation_and_trail)

parsers.check_optional_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_optional_blank_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)
```

The following patterns specify behaviour around newlines.

```
parsers.spnlc_noexc = parsers.optionalspace
                    * ( parsers.newline
                      * parsers.check_minimal_indent_and_any_trail)^-1

parsers.spnlc = parsers.optionalspace
              * (V("EndlineNoSub"))^-1

parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
                   + parsers.spacechar^1

parsers.only_blank = parsers.spacechar^0
                   * (parsers.newline + parsers.eof)
```

The `parsers.commented_line^1` parser recognizes the regular language of TEX comments, see an equivalent finite automaton in Figure 8.

```
parsers.commented_line_letter  = parsers.linechar
                               + parsers.newline
                               - parsers.backslash
                               - parsers.percent
parsers.commented_line = Cg(Cc(""), "backslashes")
                       * ((#(parsers.commented_line_letter
                            - parsers.newline)
                          * Cb("backslashes")
                          * Cs(parsers.commented_line_letter
                            - parsers.newline)^1  -- initial
                          * Cg(Cc(""), "backslashes"))
                         + #( parsers.backslash
                            * (parsers.backslash + parsers.newline))
                         * Cg((parsers.backslash  -- even backslash
                              * ( parsers.backslash
                                + #parsers.newline))^1, "backslashes")
                         + (parsers.backslash
```

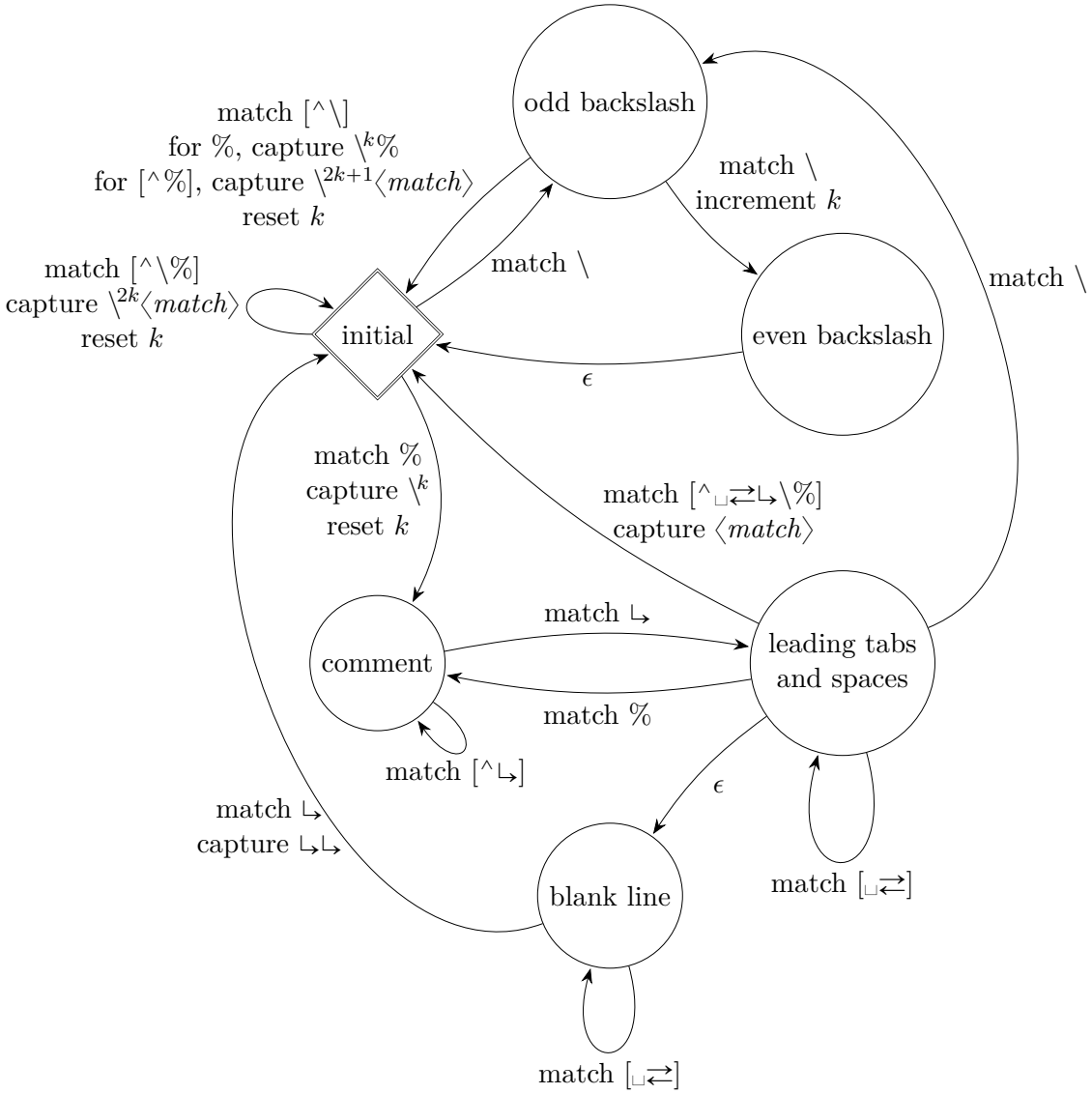

Figure 8: A pushdown automaton that recognizes TEX comments

```
                            * (#parsers.percent
                              * Cb("backslashes")
                              / function(backslashes)
                                return string.rep("\\", #backslashes / 2)
                              end
                              * C(parsers.percent)
                              + #parsers.commented_line_letter
                              * Cb("backslashes")
                              * Cc("\\")
                              * C(parsers.commented_line_letter))
                            * Cg(Cc(""), "backslashes")))^0
                          * (#parsers.percent
                            * Cb("backslashes")
                            / function(backslashes)
                              return string.rep("\\", #backslashes / 2)
                            end
                            * ((parsers.percent  -- comment
                               * parsers.line
                               * #parsers.blankline) -- blank line
                              / "\n"
                              + parsers.percent  -- comment
                              * parsers.line
                              * parsers.optionalspace)  -- leading spaces
                            + #(parsers.newline)
                            * Cb("backslashes")
                            * C(parsers.newline))

parsers.chunk = parsers.line * (parsers.optionallyindentedline
                                      - parsers.blankline)^0

parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
parsers.attribute_key = (parsers.attribute_key_char
                         - parsers.dash - parsers.digit)
                       * parsers.attribute_key_char^0
parsers.attribute_value = ( (parsers.dquote / "")
                            * (parsers.anyescaped - parsers.dquote)^0
                            * (parsers.dquote / ""))
                          + ( (parsers.squote / "")
                            * (parsers.anyescaped - parsers.squote)^0
                            * (parsers.squote / ""))
                          + ( parsers.anyescaped
                            - parsers.dquote
                            - parsers.rbrace
                            - parsers.space)^0
parsers.attribute_identifier = parsers.attribute_key_char^1
parsers.attribute_classname = parsers.letter
```

```
                            * parsers.attribute_key_char^0
parsers.attribute_raw = parsers.attribute_raw_char^1

parsers.attribute = (parsers.dash * Cc(".unnumbered"))
                  + C( parsers.hash
                     * parsers.attribute_identifier)
                  + C( parsers.period
                     * parsers.attribute_classname)
                  + Cs( parsers.attribute_key
                      * parsers.optionalspace
                      * parsers.equal
                      * parsers.optionalspace
                      * parsers.attribute_value)
parsers.attributes = parsers.lbrace
                   * parsers.optionalspace
                   * parsers.attribute
                   * (parsers.spacechar^1
                     * parsers.attribute)^0
                   * parsers.optionalspace
                   * parsers.rbrace

parsers.raw_attribute = parsers.lbrace
                      * parsers.optionalspace
                      * parsers.equal
                      * C(parsers.attribute_raw)
                      * parsers.optionalspace
                      * parsers.rbrace

-- block followed by 0 or more optionally
-- indented blocks with first line indented.
parsers.indented_blocks = function(bl)
  return Cs( bl
         * ( parsers.blankline^1
           * parsers.indent
           * -parsers.blankline
           * bl)^0
         * (parsers.blankline^1 + parsers.eof) )
end
```

3.1.5.2 Parsers Used for HTML Entities

```
local function repeat_between(pattern, min, max)
  return -pattern^(max + 1) * pattern^min
end

parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
                  * C(repeat_between(parsers.hexdigit, 1, 6))
```

```
                  * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                  * C(repeat_between(parsers.digit, 1, 7))
                  * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
                  * parsers.semicolon

parsers.html_entities
  = parsers.hexentity / entities.hex_entity_with_x_char
  + parsers.decentity / entities.dec_entity
  + parsers.tagentity / entities.char_entity
```

3.1.5.3 Parsers Used for Markdown Lists

```
parsers.bullet = function(bullet_char, interrupting)
  local allowed_end
  if interrupting then
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    allowed_end = C(parsers.spacechar^1)
                + #(parsers.newline + parsers.eof)
  end
  return parsers.check_trail
        * Ct(C(bullet_char) * Cc(""))
        * allowed_end
end

local function tickbox(interior)
  return parsers.optionalspace * parsers.lbracket
      * interior * parsers.rbracket * parsers.spacechar^1
end

parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)

```

3.1.5.4 Parsers Used for Markdown Code Spans

```
parsers.openticks   = Cg(parsers.backtick^1, "ticks")

local function captures_equal_length(_,i,a,b)
  return #a == #b and i
end

parsers.closeticks  = Cmt(C(parsers.backtick^1)
                          * Cb("ticks"), captures_equal_length)

```

```
parsers.intickschar = (parsers.any - S("\n\r`"))
                    + V("NoSoftLineBreakEndline")
                    + (parsers.backtick^1 - parsers.closeticks)

local function process_inticks(s)
  s = s:gsub("\n", " ")
  s = s:gsub("^ (.*) $", "%1")
  return s
end

parsers.inticks = parsers.openticks
                * C(parsers.space^0)
                * parsers.closeticks
                + parsers.openticks
                * Cs(Cs(parsers.intickschar^0) / process_inticks)
                * parsers.closeticks

```

3.1.5.5 Parsers Used for HTML

```
-- case-insensitive match (we assume s is lowercase)
-- must be single byte encoding
parsers.keyword_exact = function(s)
  local parser = P(0)
  for i=1,#s do
    local c = s:sub(i,i)
    local m = c .. upper(c)
    parser = parser * S(m)
  end
  return parser
end

parsers.special_block_keyword =
    parsers.keyword_exact("pre") +
    parsers.keyword_exact("script") +
    parsers.keyword_exact("style") +
    parsers.keyword_exact("textarea")

parsers.block_keyword =
    parsers.keyword_exact("address") +
    parsers.keyword_exact("article") +
    parsers.keyword_exact("aside") +
    parsers.keyword_exact("base") +
    parsers.keyword_exact("basefont") +
    parsers.keyword_exact("blockquote") +
    parsers.keyword_exact("body") +
    parsers.keyword_exact("caption") +
```

```
    parsers.keyword_exact("center") +
    parsers.keyword_exact("col") +
    parsers.keyword_exact("colgroup") +
    parsers.keyword_exact("dd") +
    parsers.keyword_exact("details") +
    parsers.keyword_exact("dialog") +
    parsers.keyword_exact("dir") +
    parsers.keyword_exact("div") +
    parsers.keyword_exact("dl") +
    parsers.keyword_exact("dt") +
    parsers.keyword_exact("fieldset") +
    parsers.keyword_exact("figcaption") +
    parsers.keyword_exact("figure") +
    parsers.keyword_exact("footer") +
    parsers.keyword_exact("form") +
    parsers.keyword_exact("frame") +
    parsers.keyword_exact("frameset") +
    parsers.keyword_exact("h1") +
    parsers.keyword_exact("h2") +
    parsers.keyword_exact("h3") +
    parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") +
    parsers.keyword_exact("h6") +
    parsers.keyword_exact("head") +
    parsers.keyword_exact("header") +
    parsers.keyword_exact("hr") +
    parsers.keyword_exact("html") +
    parsers.keyword_exact("iframe") +
    parsers.keyword_exact("legend") +
    parsers.keyword_exact("li") +
    parsers.keyword_exact("link") +
    parsers.keyword_exact("main") +
    parsers.keyword_exact("menu") +
    parsers.keyword_exact("menuitem") +
    parsers.keyword_exact("nav") +
    parsers.keyword_exact("noframes") +
    parsers.keyword_exact("ol") +
    parsers.keyword_exact("optgroup") +
    parsers.keyword_exact("option") +
    parsers.keyword_exact("p") +
    parsers.keyword_exact("param") +
    parsers.keyword_exact("section") +
    parsers.keyword_exact("source") +
    parsers.keyword_exact("summary") +
    parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") +
    parsers.keyword_exact("td") +
```

```
    parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("th") +
    parsers.keyword_exact("thead") +
    parsers.keyword_exact("title") +
    parsers.keyword_exact("tr") +
    parsers.keyword_exact("track") +
    parsers.keyword_exact("ul")

-- end conditions
parsers.html_blankline_end_condition
  = parsers.linechar^0
  * ( parsers.newline
    * (parsers.check_minimal_blank_indent_and_any_trail
      * #parsers.blankline
      + parsers.check_minimal_indent_and_any_trail)
    * parsers.linechar^1)^0
  * (parsers.newline^-1 / "")

local function remove_trailing_blank_lines(s)
  return s:gsub("[\n\r]+%s*$", "")
end

parsers.html_until_end = function(end_marker)
  return Cs(Cs((parsers.newline
           * (parsers.check_minimal_blank_indent_and_any_trail
             * #parsers.blankline
             + parsers.check_minimal_indent_and_any_trail)
           + parsers.linechar - end_marker)^0
           * parsers.linechar^0 * parsers.newline^-1)
         / remove_trailing_blank_lines)
end

-- attributes
parsers.html_attribute_spacing  = parsers.optionalspace
                                * V("NoSoftLineBreakEndline")
                                * parsers.optionalspace
                                + parsers.spacechar^1

parsers.html_attribute_name = ( parsers.letter
                              + parsers.colon
                              + parsers.underscore)
                            * ( parsers.alphanumeric
                              + parsers.colon
                              + parsers.underscore
                            + parsers.period
                            + parsers.dash)^0

```

```
parsers.html_attribute_value  = parsers.squote
                              * (parsers.linechar - parsers.squote)^0
                              * parsers.squote
                              + parsers.dquote
                              * (parsers.linechar - parsers.dquote)^0
                              * parsers.dquote
                              + ( parsers.any
                                - parsers.spacechar
                                - parsers.newline
                                - parsers.dquote
                                - parsers.squote
                                - parsers.backtick
                                - parsers.equal
                                - parsers.less
                                - parsers.more)^1

parsers.html_inline_attribute_value = parsers.squote
                                    * (V("NoSoftLineBreakEndline")
                                      + parsers.any
                                      - parsers.blankline^2
                                      - parsers.squote)^0
                                    * parsers.squote
                                    + parsers.dquote
                                    * (V("NoSoftLineBreakEndline")
                                      + parsers.any
                                      - parsers.blankline^2
                                      - parsers.dquote)^0
                                    * parsers.dquote
                                    + (parsers.any
                                      - parsers.spacechar
                                      - parsers.newline
                                      - parsers.dquote
                                      - parsers.squote
                                      - parsers.backtick
                                      - parsers.equal
                                      - parsers.less
                                      - parsers.more)^1

parsers.html_attribute_value_specification
  = parsers.optionalspace
  * parsers.equal
  * parsers.optionalspace
  * parsers.html_attribute_value

parsers.html_spnl = parsers.optionalspace
                  * (V("NoSoftLineBreakEndline")
                  * parsers.optionalspace)^-1
```

```
8157
8158 parsers.html_inline_attribute_value_specification
8159   = parsers.html_spnl
8160   * parsers.equal
8161   * parsers.html_spnl
8162   * parsers.html_inline_attribute_value
8163
8164 parsers.html_attribute
8165   = parsers.html_attribute_spacing
8166   * parsers.html_attribute_name
8167   * parsers.html_inline_attribute_value_specification^-1
8168
8169 parsers.html_non_newline_attribute
8170   = parsers.spacechar^1
8171   * parsers.html_attribute_name
8172   * parsers.html_attribute_value_specification^-1
8173
8174 parsers.nested_breaking_blank = parsers.newline
8175                               * parsers.check_minimal_blank_indent
8176                               * parsers.blankline
8177
8178 parsers.html_comment_start = P("<!--")
8179
8180 parsers.html_comment_end = P("-->")
8181
8182 parsers.html_comment
8183   = Cs( parsers.html_comment_start
8184       * parsers.html_until_end(parsers.html_comment_end))
8185
8186 parsers.html_inline_comment = (parsers.html_comment_start / "")
8187                             * -P(">") * -P("->")
8188                             * Cs(( V("NoSoftLineBreakEndline")
8189                                  + parsers.any
8190                                  - parsers.nested_breaking_blank
8191                                  - parsers.html_comment_end)^0)
8192                             * (parsers.html_comment_end / "")
8193
8194 parsers.html_cdatasection_start = P("<![CDATA[")
8195
8196 parsers.html_cdatasection_end = P("]]>")
8197
8198 parsers.html_cdatasection
8199   = Cs( parsers.html_cdatasection_start
8200       * parsers.html_until_end(parsers.html_cdatasection_end))
8201
8202 parsers.html_inline_cdatasection
8203   = parsers.html_cdatasection_start
```

```
  * Cs(V("NoSoftLineBreakEndline") + parsers.any
      - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
  * parsers.html_cdatasection_end

parsers.html_declaration_start = P("<!") * parsers.letter

parsers.html_declaration_end = P(">")

parsers.html_declaration
  = Cs( parsers.html_declaration_start
      * parsers.html_until_end(parsers.html_declaration_end))

parsers.html_inline_declaration
  = parsers.html_declaration_start
  * Cs(V("NoSoftLineBreakEndline") + parsers.any
      - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
  * parsers.html_declaration_end

parsers.html_instruction_start = P("<?")

parsers.html_instruction_end = P("?>")

parsers.html_instruction
  = Cs( parsers.html_instruction_start
      * parsers.html_until_end(parsers.html_instruction_end))

parsers.html_inline_instruction = parsers.html_instruction_start
                                * Cs( V("NoSoftLineBreakEndline")
                                    + parsers.any
                                    - parsers.nested_breaking_blank
                                    - parsers.html_instruction_end)^0
                                * parsers.html_instruction_end

parsers.html_blankline  = parsers.newline
                        * parsers.optionalspace
                        * parsers.newline

parsers.html_tag_start = parsers.less

parsers.html_tag_closing_start  = parsers.less
                                * parsers.slash

parsers.html_tag_end  = parsers.html_spnl
                      * parsers.more

parsers.html_empty_tag_end  = parsers.html_spnl
                            * parsers.slash
```

```
                              * parsers.more

-- opening tags
parsers.html_any_open_inline_tag  = parsers.html_tag_start
                                  * parsers.keyword
                                  * parsers.html_attribute^0
                                  * parsers.html_tag_end

parsers.html_any_open_tag = parsers.html_tag_start
                          * parsers.keyword
                          * parsers.html_non_newline_attribute^0
                          * parsers.html_tag_end

parsers.html_open_tag = parsers.html_tag_start
                      * parsers.block_keyword
                      * parsers.html_attribute^0
                      * parsers.html_tag_end

parsers.html_open_special_tag = parsers.html_tag_start
                              * parsers.special_block_keyword
                              * parsers.html_attribute^0
                              * parsers.html_tag_end

-- incomplete tags
parsers.incomplete_tag_following  = parsers.spacechar
                                  + parsers.more
                                  + parsers.slash * parsers.more
                                  + #(parsers.newline + parsers.eof)

parsers.incomplete_special_tag_following = parsers.spacechar
                                         + parsers.more
                                         + #( parsers.newline
                                            + parsers.eof)

parsers.html_incomplete_open_tag  = parsers.html_tag_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

parsers.html_incomplete_open_special_tag
  = parsers.html_tag_start
  * parsers.special_block_keyword
  * parsers.incomplete_special_tag_following

parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

```

```
parsers.html_incomplete_close_special_tag
  = parsers.html_tag_closing_start
  * parsers.special_block_keyword
  * parsers.incomplete_tag_following

-- closing tags
parsers.html_close_tag  = parsers.html_tag_closing_start
                        * parsers.block_keyword
                        * parsers.html_tag_end

parsers.html_any_close_tag  = parsers.html_tag_closing_start
                            * parsers.keyword
                            * parsers.html_tag_end

parsers.html_close_special_tag = parsers.html_tag_closing_start
                               * parsers.special_block_keyword
                               * parsers.html_tag_end

-- empty tags
parsers.html_any_empty_inline_tag = parsers.html_tag_start
                                  * parsers.keyword
                                  * parsers.html_attribute^0
                                  * parsers.html_empty_tag_end

parsers.html_any_empty_tag  = parsers.html_tag_start
                            * parsers.keyword
                            * parsers.html_non_newline_attribute^0
                            * parsers.optionalspace
                            * parsers.slash
                            * parsers.more

parsers.html_empty_tag  = parsers.html_tag_start
                        * parsers.block_keyword
                        * parsers.html_attribute^0
                        * parsers.html_empty_tag_end

parsers.html_empty_special_tag  = parsers.html_tag_start
                                * parsers.special_block_keyword
                                * parsers.html_attribute^0
                                * parsers.html_empty_tag_end

parsers.html_incomplete_blocks
  = parsers.html_incomplete_open_tag
  + parsers.html_incomplete_open_special_tag
  + parsers.html_incomplete_close_tag

-- parse special html blocks
```

```
parsers.html_blankline_ending_special_block_opening
  = ( parsers.html_close_special_tag
    + parsers.html_empty_special_tag)
  * #( parsers.optionalspace
     * (parsers.newline + parsers.eof))

parsers.html_blankline_ending_special_block
  = parsers.html_blankline_ending_special_block_opening
  * parsers.html_blankline_end_condition

parsers.html_special_block_opening
  = parsers.html_incomplete_open_special_tag
  - parsers.html_empty_special_tag

parsers.html_closing_special_block
  = parsers.html_special_block_opening
  * parsers.html_until_end(parsers.html_close_special_tag)

parsers.html_special_block
  = parsers.html_blankline_ending_special_block
  + parsers.html_closing_special_block

-- parse html blocks
parsers.html_block_opening  = parsers.html_incomplete_open_tag
                            + parsers.html_incomplete_close_tag

parsers.html_block  = parsers.html_block_opening
                    * parsers.html_blankline_end_condition

-- parse any html blocks
parsers.html_any_block_opening
  = ( parsers.html_any_open_tag
    + parsers.html_any_close_tag
    + parsers.html_any_empty_tag)
  * #(parsers.optionalspace * (parsers.newline + parsers.eof))

parsers.html_any_block  = parsers.html_any_block_opening
                        * parsers.html_blankline_end_condition

parsers.html_inline_comment_full  = parsers.html_comment_start
                                  * -P(">") * -P("->")
                                  * Cs(( V("NoSoftLineBreakEndline")
                                       + parsers.any - P("--")
                                       - parsers.nested_breaking_blank
                                       - parsers.html_comment_end)^0)
                                  * parsers.html_comment_end

```

```
parsers.html_inline_tags  = parsers.html_inline_comment_full
                          + parsers.html_any_empty_inline_tag
                          + parsers.html_inline_instruction
                          + parsers.html_inline_cdatasection
                          + parsers.html_inline_declaration
                          + parsers.html_any_open_inline_tag
                          + parsers.html_any_close_tag

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```
parsers.urlchar = parsers.anyescaped
                - parsers.newline
                - parsers.more

parsers.auto_link_scheme_part = parsers.alphanumeric
                              + parsers.plus
                              + parsers.period
                              + parsers.dash

parsers.auto_link_scheme  = parsers.letter
                          * parsers.auto_link_scheme_part
                          * parsers.auto_link_scheme_part^-30

parsers.absolute_uri  = parsers.auto_link_scheme * parsers.colon
                      * ( parsers.any - parsers.spacing
                        - parsers.less - parsers.more)^0

parsers.printable_characters = S(".!#$%&'*+/=?^_`{|}~-")

parsers.email_address_local_part_char = parsers.alphanumeric
                                      + parsers.printable_characters

parsers.email_address_local_part
  = parsers.email_address_local_part_char^1

parsers.email_address_dns_label = parsers.alphanumeric
                                * ( parsers.alphanumeric
                                  + parsers.dash)^-62
                                * B(parsers.alphanumeric)

parsers.email_address_domain  = parsers.email_address_dns_label
                              * ( parsers.period
                                * parsers.email_address_dns_label)^0

parsers.email_address = parsers.email_address_local_part
                      * parsers.at
```

```
                      * parsers.email_address_domain

parsers.auto_link_url = parsers.less
                      * C(parsers.absolute_uri)
                      * parsers.more

parsers.auto_link_email = parsers.less
                        * C(parsers.email_address)
                        * parsers.more

parsers.auto_link_relative_reference = parsers.less
                                     * C(parsers.urlchar^1)
                                     * parsers.more

parsers.autolink  = parsers.auto_link_url
                  + parsers.auto_link_email

-- content in balanced brackets, parentheses, or quotes:
parsers.bracketed   = P{ parsers.lbracket
                       * (( parsers.backslash / "" * parsers.rbracket
                          + parsers.any - (parsers.lbracket
                                         + parsers.rbracket
                                         + parsers.blankline^2)
                         ) + V(1))^0
                       * parsers.rbracket }

parsers.inparens    = P{ parsers.lparent
                       * ((parsers.anyescaped - (parsers.lparent
                                               + parsers.rparent
                                               + parsers.blankline^2)
                         ) + V(1))^0
                       * parsers.rparent }

parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.squote
                                               + parsers.blankline^2)
                         ) + V(1))^0
                       * parsers.squote }

parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.dquote
                                               + parsers.blankline^2)
                         ) + V(1))^0
                       * parsers.dquote }

parsers.link_text  = parsers.lbracket
                   * Cs((parsers.alphanumeric^1
```

```
                        + parsers.bracketed
                        + parsers.inticks
                        + parsers.autolink
                        + V("InlineHtml")
                        + ( parsers.backslash * parsers.backslash)
                        + ( parsers.backslash
                          * ( parsers.lbracket
                            + parsers.rbracket)
                          + V("NoSoftLineBreakSpace")
                          + V("NoSoftLineBreakEndline")
                          + (parsers.any
                            - ( parsers.newline
                              + parsers.lbracket
                              + parsers.rbracket
                              + parsers.blankline^2))))^0)
                    * parsers.rbracket

parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
                        * #( ( parsers.any
                             - parsers.rbracket)^-999
                           * parsers.rbracket)
                        * Cs((parsers.alphanumeric^1
                            + parsers.inticks
                            + parsers.autolink
                            + V("InlineHtml")
                            + ( parsers.backslash * parsers.backslash)
                            + ( parsers.backslash
                              * ( parsers.lbracket
                                + parsers.rbracket)
                              + V("NoSoftLineBreakSpace")
                              + V("NoSoftLineBreakEndline")
                              + (parsers.any
                                - ( parsers.newline
                                  + parsers.lbracket
                                  + parsers.rbracket
                                  + parsers.blankline^2))))^1)

parsers.link_label  = parsers.lbracket
                    * parsers.link_label_body
                    * parsers.rbracket

parsers.inparens_url  = P{ parsers.lparent
                         * ((parsers.anyescaped - (parsers.lparent
                                                 + parsers.rparent
                                                 + parsers.spacing)
                           ) + V(1))^0
                         * parsers.rparent }
```

```
8530
8531 -- url for markdown links, allowing nested brackets:
8532 parsers.url         = parsers.less * Cs((parsers.anyescaped
8533                                        - parsers.newline
8534                                        - parsers.less
8535                                        - parsers.more)^0)
8536                                  * parsers.more
8537                     + -parsers.less
8538                     * Cs((parsers.inparens_url + (parsers.anyescaped
8539                                                   - parsers.spacing
8540                                                   - parsers.lparent
8541                                                   - parsers.rparent))^1)
8542
8543 -- quoted text:
8544 parsers.title_s     = parsers.squote
8545                     * Cs((parsers.html_entities
8546                         + V("NoSoftLineBreakSpace")
8547                         + V("NoSoftLineBreakEndline")
8548                         + ( parsers.anyescaped
8549                           - parsers.newline
8550                           - parsers.squote
8551                           - parsers.blankline^2))^0)
8552                     * parsers.squote
8553
8554 parsers.title_d     = parsers.dquote
8555                     * Cs((parsers.html_entities
8556                         + V("NoSoftLineBreakSpace")
8557                         + V("NoSoftLineBreakEndline")
8558                         + ( parsers.anyescaped
8559                           - parsers.newline
8560                           - parsers.dquote
8561                           - parsers.blankline^2))^0)
8562                     * parsers.dquote
8563
8564 parsers.title_p     = parsers.lparent
8565                     * Cs((parsers.html_entities
8566                         + V("NoSoftLineBreakSpace")
8567                         + V("NoSoftLineBreakEndline")
8568                         + ( parsers.anyescaped
8569                           - parsers.newline
8570                           - parsers.lparent
8571                           - parsers.rparent
8572                           - parsers.blankline^2))^0)
8573                     * parsers.rparent
8574
8575 parsers.title
8576   = parsers.title_d + parsers.title_s + parsers.title_p
```

```
8577
8578 parsers.optionaltitle
8579   = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8580
```

3.1.5.7 Helpers for Links and Link Reference Definitions

```
8581 -- parse a reference definition:  [foo]: /bar "title"
8582 parsers.define_reference_parser = (parsers.check_trail / "")
8583                                  * parsers.link_label * parsers.colon
8584                                  * parsers.spnlc * parsers.url
8585                                  * ( parsers.spnlc_sep * parsers.title
8586                                    * parsers.only_blank
8587                                    + Cc("") * parsers.only_blank)
```

3.1.5.8 Inline Elements

```
8588 parsers.Inline         = V("Inline")
8589
8590 -- parse many p between starter and ender
8591 parsers.between = function(p, starter, ender)
8592   local ender2 = B(parsers.nonspacechar) * ender
8593   return ( starter
8594          * #parsers.nonspacechar
8595          * Ct(p * (p - ender2)^0)
8596          * ender2)
8597 end
8598
```

3.1.5.9 Block Elements

```
8599 parsers.lineof = function(c)
8600     return ( parsers.check_trail_no_rem
8601            * (P(c) * parsers.optionalspace)^3
8602            * (parsers.newline + parsers.eof))
8603 end
8604
8605 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8606                              + parsers.lineof(parsers.dash)
8607                              + parsers.lineof(parsers.underscore)
```

3.1.5.10 Headings

```
8608 -- parse Atx heading start and return level
8609 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8610                       * -parsers.hash / length
8611
8612 -- parse setext header ending and return level
8613 parsers.heading_level
```

```
  = parsers.nonindentspace * parsers.equal^1
  * parsers.optionalspace * #parsers.newline * Cc(1)
  + parsers.nonindentspace * parsers.dash^1
  * parsers.optionalspace * #parsers.newline * Cc(2)

local function strip_atx_end(s)
  return s:gsub("%s+#*%s*\n$","")
end

parsers.atx_heading = parsers.check_trail_no_rem
                    * Cg(parsers.heading_start, "level")
                    * (C( parsers.optionalspace
                        * parsers.hash^0
                        * parsers.optionalspace
                        * parsers.newline)
                      + parsers.spacechar^1
                      * C(parsers.line))
```

3.1.6 Markdown Reader

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

The `reader.new` method creates and returns a new TEX reader object associated with the Lua interface options (see Section 2.1.3) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

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

```
M.reader = {}
function M.reader.new(writer, options)
  local self = {}
```

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

```
  self.writer = writer
  self.options = options
```

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

```
  self.parsers = {}
  (function(parsers)
    setmetatable(self.parsers, {
      __index = function (_, key)
        return parsers[key]
```

```
8641        end
8642      })
8643    end)(parsers)
```

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

```
8644    local parsers = self.parsers
```

3.1.6.1 Top-Level Helper Functions

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

```
8645    function self.normalize_tag(tag)
8646      tag = util.rope_to_string(tag)
8647      tag = tag:gsub("[ \n\r\t]+", " ")
8648      tag = tag:gsub("^ ", ""):gsub(" $", "")
8649      tag = uni_algos.case.casefold(tag, true, false)
8650      return tag
8651    end
```

Define `iterlines` as a function that iterates over the lines of the input string `s`, transforms them using an input function `f`, and reassembles them into a new string, which it returns.

```
8652    local function iterlines(s, f)
8653      local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8654      return util.rope_to_string(rope)
8655    end
```

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

```
8656    if options.preserveTabs then
8657      self.expandtabs = function(s) return s end
8658    else
8659      self.expandtabs = function(s)
8660                          if s:find("\t") then
8661                            return iterlines(s, util.expand_tabs_in_line)
8662                          else
8663                            return s
8664                          end
8665                        end
8666    end
```

3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using

grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
  self.parser_functions = {}
  self.create_parser = function(name, grammar, toplevel)
    self.parser_functions[name] = function(str)
```

If the parser function is top-level and the `stripIndent` Lua option is enabled, we will first expand tabs in the input string `str` into spaces and then we will count the minimum indent across all lines, skipping blank lines. Next, we will remove the minimum indent from all lines.

```
      if toplevel and options.stripIndent then
          local min_prefix_length, min_prefix = nil, ''
          str = iterlines(str, function(line)
              if lpeg.match(parsers.nonemptyline, line) == nil then
                  return line
              end
              line = util.expand_tabs_in_line(line)
              local prefix = lpeg.match(C(parsers.optionalspace), line)
              local prefix_length = #prefix
              local is_shorter = min_prefix_length == nil
              if not is_shorter then
                is_shorter = prefix_length < min_prefix_length
              end
              if is_shorter then
                min_prefix_length, min_prefix = prefix_length, prefix
              end
              return line
          end)
          str = str:gsub('^' .. min_prefix, '')
      end
```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain TEX comments from the input string `str` together with the trailing newline characters.

```
      if toplevel and (options.texComments or options.hybrid) then
        str = lpeg.match(Ct(parsers.commented_line^1), str)
        str = util.rope_to_string(str)
      end
      local res = lpeg.match(grammar(), str)
      if res == nil then
        return writer.error(
          format("Parser `%s` failed to process the input text.", name),
          format("Here are the first 20 characters of the remaining "
              .. "unprocessed text: `%s`.", str:sub(1,20))
        )
      else
        return res
```

```
      end
    end
  end

  self.create_parser("parse_blocks",
                     function()
                       return parsers.blocks
                     end, true)

  self.create_parser("parse_blocks_nested",
                     function()
                       return parsers.blocks_nested
                     end, false)

  self.create_parser("parse_inlines",
                     function()
                       return parsers.inlines
                     end, false)

  self.create_parser("parse_inlines_no_inline_note",
                     function()
                       return parsers.inlines_no_inline_note
                     end, false)

  self.create_parser("parse_inlines_no_html",
                     function()
                       return parsers.inlines_no_html
                     end, false)

  self.create_parser("parse_inlines_nbsp",
                     function()
                       return parsers.inlines_nbsp
                     end, false)
  self.create_parser("parse_inlines_no_link_or_emphasis",
                    function()
                      return parsers.inlines_no_link_or_emphasis
                    end, false)
```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
  parsers.minimally_indented_blankline
    = parsers.check_minimal_indent * (parsers.blankline / "")

  parsers.minimally_indented_block
    = parsers.check_minimal_indent * V("Block")

```

```
parsers.minimally_indented_block_or_paragraph
  = parsers.check_minimal_indent * V("BlockOrParagraph")

parsers.minimally_indented_paragraph
  = parsers.check_minimal_indent * V("Paragraph")

parsers.minimally_indented_plain
  = parsers.check_minimal_indent * V("Plain")

parsers.minimally_indented_par_or_plain
  = parsers.minimally_indented_paragraph
  + parsers.minimally_indented_plain

parsers.minimally_indented_par_or_plain_no_blank
  = parsers.minimally_indented_par_or_plain
  - parsers.minimally_indented_blankline

parsers.minimally_indented_ref
  = parsers.check_minimal_indent * V("Reference")

parsers.minimally_indented_blank
  = parsers.check_minimal_indent * V("Blank")

parsers.conditionally_indented_blankline
  = parsers.check_minimal_blank_indent * (parsers.blankline / "")

parsers.minimally_indented_ref_or_block
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block
  - parsers.minimally_indented_blankline

parsers.minimally_indented_ref_or_block_or_par
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block_or_paragraph
  - parsers.minimally_indented_blankline

```

The following pattern parses the properly indented content that follows the initial container start.

```

function parsers.separator_loop(separated_block, paragraph,
                                block_separator, paragraph_separator)
  return  separated_block
        + block_separator
          * paragraph
          * separated_block
        + paragraph_separator
```

```
        * paragraph
end

function parsers.create_loop_body_pair(separated_block, paragraph,
                                       block_separator,
                                       paragraph_separator)
  return {
    block = parsers.separator_loop(separated_block, paragraph,
                                   block_separator, block_separator),
    par = parsers.separator_loop(separated_block, paragraph,
                                 block_separator, paragraph_separator)
  }
end

parsers.block_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + ( blank^2 / writer.paragraphsep
          + blank^0 / writer.interblocksep
          )
end

parsers.par_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + blank^0 / writer.paragraphsep
end

parsers.sep_group_no_output = function(blank)
  return  blank^0 * parsers.eof
        + blank^0
end

parsers.content_blank = parsers.minimally_indented_blankline

parsers.ref_or_block_separated
  = parsers.sep_group_no_output(parsers.content_blank)
  * ( parsers.minimally_indented_ref
    - parsers.content_blank)
  + parsers.block_sep_group(parsers.content_blank)
  * ( parsers.minimally_indented_block
    - parsers.content_blank)

parsers.loop_body_pair  =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_separated,
    parsers.minimally_indented_par_or_plain_no_blank,
    parsers.block_sep_group(parsers.content_blank),
    parsers.par_sep_group(parsers.content_blank))
```

```
8837
8838  parsers.content_loop  = ( V("Block")
8839                          * parsers.loop_body_pair.block^0
8840                          + (V("Paragraph") + V("Plain"))
8841                          * parsers.ref_or_block_separated
8842                          * parsers.loop_body_pair.block^0
8843                          + (V("Paragraph") + V("Plain"))
8844                          * parsers.loop_body_pair.par^0)
8845                        * parsers.content_blank^0
8846
8847  parsers.indented_content = function()
8848    return  Ct( (V("Reference") + (parsers.blankline / ""))
8849              * parsers.content_blank^0
8850              * parsers.check_minimal_indent
8851              * parsers.content_loop
8852              + (V("Reference") + (parsers.blankline / ""))
8853              * parsers.content_blank^0
8854              + parsers.content_loop)
8855  end
8856
8857  parsers.add_indent = function(pattern, name, breakable)
8858    return  Cg(Cmt( Cb("indent_info")
8859                  * Ct(pattern)
8860                  * ( #parsers.linechar  -- check if starter is blank
8861                    * Cc(false) + Cc(true))
8862                  * Cc(name)
8863                  * Cc(breakable),
8864              process_starter_indent), "indent_info")
8865  end
8866
```

3.1.6.4 Parsers Used for Markdown Lists (local)

```
8867  if options.hashEnumerators then
8868    parsers.dig = parsers.digit + parsers.hash
8869  else
8870    parsers.dig = parsers.digit
8871  end
8872
8873  parsers.enumerator = function(delimiter_type, interrupting)
8874    local delimiter_range
8875    local allowed_end
8876    if interrupting then
8877      delimiter_range = P("1")
8878      allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8879    else
8880      delimiter_range = parsers.dig * parsers.dig^-8
```

```
8881        allowed_end = C(parsers.spacechar^1)
8882                    + #(parsers.newline + parsers.eof)
8883      end
8884
8885      return parsers.check_trail
8886               * Ct(C(delimiter_range) * C(delimiter_type))
8887               * allowed_end
8888    end
8889
8890    parsers.starter = parsers.bullet(parsers.dash)
8891                    + parsers.bullet(parsers.asterisk)
8892                    + parsers.bullet(parsers.plus)
8893                    + parsers.enumerator(parsers.period)
8894                    + parsers.enumerator(parsers.rparent)
8895
```

3.1.6.5 Parsers Used for Blockquotes (local)

```
8896    parsers.blockquote_start
8897      = parsers.check_trail
8898      * C(parsers.more)
8899      * C(parsers.spacechar^0)
8900
8901    parsers.blockquote_body
8902      = parsers.add_indent(parsers.blockquote_start, "bq", true)
8903      * parsers.indented_content()
8904      * remove_indent("bq")
8905
8906    if not options.breakableBlockquotes then
8907      parsers.blockquote_body
8908        = parsers.add_indent(parsers.blockquote_start, "bq", false)
8909        * parsers.indented_content()
8910        * remove_indent("bq")
8911    end
```

3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```
8912    local function parse_content_part(content_part)
8913      local rope = util.rope_to_string(content_part)
8914      local parsed
8915        = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8916      parsed.indent_info = nil
8917      return parsed
8918    end
8919
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local collect_emphasis_content =
  function(t, opening_index, closing_index)
    local content = {}

    local content_part = {}
    for i = opening_index, closing_index do
      local value = t[i]

      if value.rendered ~= nil then
        content[#content + 1] = parse_content_part(content_part)
        content_part = {}
        content[#content + 1] = value.rendered
        value.rendered = nil
      else
        if value.type == "delimiter"
            and value.element == "emphasis" then
          if value.is_active then
            content_part[#content_part + 1]
              = string.rep(value.character, value.current_count)
          end
        else
          content_part[#content_part + 1] = value.content
        end
        value.content = ''
        value.is_active = false
      end
    end

    if next(content_part) ~= nil then
      content[#content + 1] = parse_content_part(content_part)
    end

    return content
  end

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
local function fill_emph(t, opening_index, closing_index)
  local content
    = collect_emphasis_content(t, opening_index + 1,
                               closing_index - 1)
  t[opening_index + 1].is_active = true
  t[opening_index + 1].rendered = writer.emphasis(content)
end
```

```
8962
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
8963  local function fill_strong(t, opening_index, closing_index)
8964    local content
8965      = collect_emphasis_content(t, opening_index + 1,
8966                                 closing_index - 1)
8967    t[opening_index + 1].is_active = true
8968    t[opening_index + 1].rendered = writer.strong(content)
8969  end
8970
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
8971  local function breaks_three_rule(opening_delimiter, closing_delimiter)
8972    return ( opening_delimiter.is_closing
8973          or closing_delimiter.is_opening)
8974       and (( opening_delimiter.original_count
8975            + closing_delimiter.original_count) % 3 == 0)
8976       and ( opening_delimiter.original_count % 3 ~= 0
8977          or closing_delimiter.original_count % 3 ~= 0)
8978  end
8979
```

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```
8980  local find_emphasis_opener = function(t, bottom_index, latest_index,
8981                                        character, closing_delimiter)
8982    for i = latest_index, bottom_index, -1 do
8983      local value = t[i]
8984      if value.is_active and
8985         value.is_opening and
8986         value.type == "delimiter" and
8987         value.element == "emphasis" and
8988         (value.character == character) and
8989         (value.current_count > 0) then
8990        if not breaks_three_rule(value, closing_delimiter) then
8991          return i
8992        end
8993      end
8994    end
8995  end
8996
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```
local function process_emphasis(t, opening_index, closing_index)
  for i = opening_index, closing_index do
    local value = t[i]
    if value.type == "delimiter" and value.element == "emphasis" then
        local delimiter_length = string.len(value.content)
        value.character = string.sub(value.content, 1, 1)
        value.current_count = delimiter_length
        value.original_count = delimiter_length
    end
  end

  local openers_bottom = {
    ['*'] = {
      [true] = {opening_index, opening_index, opening_index},
      [false] = {opening_index, opening_index, opening_index}
    },
    ['_'] = {
      [true] = {opening_index, opening_index, opening_index},
      [false] = {opening_index, opening_index, opening_index}
    }
  }

  local current_position = opening_index
  local max_position = closing_index

  while current_position <= max_position do
    local value = t[current_position]

    if value.type ~= "delimiter" or
      value.element ~= "emphasis" or
      not value.is_active or
      not value.is_closing or
      (value.current_count <= 0) then
      current_position = current_position + 1
      goto continue
    end

    local character = value.character
    local is_opening = value.is_opening
    local closing_length_modulo_three = value.original_count % 3

    local current_openers_bottom
      = openers_bottom[character][is_opening]
                      [closing_length_modulo_three + 1]

    local opener_position
      = find_emphasis_opener(t, current_openers_bottom,
```

```
                                current_position - 1, character, value)

      if (opener_position == nil) then
        openers_bottom[character][is_opening]
                      [closing_length_modulo_three + 1]
          = current_position
        current_position = current_position + 1
        goto continue
      end

      local opening_delimiter = t[opener_position]

      local current_opening_count = opening_delimiter.current_count
      local current_closing_count = t[current_position].current_count

      if (current_opening_count >= 2)
        and (current_closing_count >= 2) then
        opening_delimiter.current_count = current_opening_count - 2
        t[current_position].current_count = current_closing_count - 2
        fill_strong(t, opener_position, current_position)
      else
        opening_delimiter.current_count = current_opening_count - 1
        t[current_position].current_count = current_closing_count - 1
        fill_emph(t, opener_position, current_position)
      end

      ::continue::
    end
  end

  local cont = lpeg.R("\128\191") -- continuation byte

```

Match a UTF-8 character of byte length `n`.

```
  local function utf8_by_byte_count(n)
    if (n == 1) then
      return lpeg.R("\0\127")
    end
    if (n == 2) then
      return lpeg.R("\194\223") * cont
    end
    if (n == 3) then
      return lpeg.R("\224\239") * cont * cont
    end
    if (n == 4) then
      return lpeg.R("\240\244") * cont * cont * cont
    end
  end
```

Check if a there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```
local function check_unicode_type(s, i, start_pos, end_pos, chartype)
  local c
  local char_length
  for pos = start_pos, end_pos, 1 do
    if (start_pos < 0) then
      char_length = -pos
    else
      char_length = pos + 1
    end

    if (chartype == "punctuation") then
      if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
        return i
      end
    else
      c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
      if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
        return i
      end
    end
  end
end

local function check_preceding_unicode_punctuation(s, i)
  return check_unicode_type(s, i, -4, -1, "punctuation")
end

local function check_preceding_unicode_whitespace(s, i)
  return check_unicode_type(s, i, -4, -1, "%s")
end

local function check_following_unicode_punctuation(s, i)
  return check_unicode_type(s, i, 0, 3, "punctuation")
end

local function check_following_unicode_whitespace(s, i)
  return check_unicode_type(s, i, 0, 3, "%s")
end

parsers.unicode_preceding_punctuation
  = B(parsers.escapable)
  + Cmt(parsers.succeed, check_preceding_unicode_punctuation)

parsers.unicode_preceding_whitespace
  = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
```

```
9135
9136  parsers.unicode_following_punctuation
9137    = #parsers.escapable
9138    + Cmt(parsers.succeed, check_following_unicode_punctuation)
9139
9140  parsers.unicode_following_whitespace
9141    = Cmt(parsers.succeed, check_following_unicode_whitespace)
9142
9143  parsers.delimiter_run = function(character)
9144    return  (B(parsers.backslash * character) + -B(character))
9145          * character^1
9146          * -#character
9147  end
9148
9149  parsers.left_flanking_delimiter_run = function(character)
9150    return  (B( parsers.any)
9151              * ( parsers.unicode_preceding_punctuation
9152                + parsers.unicode_preceding_whitespace)
9153             + -B(parsers.any))
9154            * parsers.delimiter_run(character)
9155            * parsers.unicode_following_punctuation
9156            + parsers.delimiter_run(character)
9157            * -#( parsers.unicode_following_punctuation
9158                + parsers.unicode_following_whitespace
9159                + parsers.eof)
9160  end
9161
9162  parsers.right_flanking_delimiter_run = function(character)
9163    return  parsers.unicode_preceding_punctuation
9164          * parsers.delimiter_run(character)
9165          * ( parsers.unicode_following_punctuation
9166            + parsers.unicode_following_whitespace
9167            + parsers.eof)
9168          + (B(parsers.any)
9169            * -( parsers.unicode_preceding_punctuation
9170               + parsers.unicode_preceding_whitespace))
9171          * parsers.delimiter_run(character)
9172  end
9173
9174  if options.underscores then
9175    parsers.emph_start
9176      = parsers.left_flanking_delimiter_run(parsers.asterisk)
9177      + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
9178        + ( parsers.unicode_preceding_punctuation
9179          * #parsers.right_flanking_delimiter_run(parsers.underscore)))
9180      * parsers.left_flanking_delimiter_run(parsers.underscore)
9181
```

```
  parsers.emph_end
    = parsers.right_flanking_delimiter_run(parsers.asterisk)
    + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
      + #( parsers.left_flanking_delimiter_run(parsers.underscore)
         * parsers.unicode_following_punctuation))
    * parsers.right_flanking_delimiter_run(parsers.underscore)
else
  parsers.emph_start
    = parsers.left_flanking_delimiter_run(parsers.asterisk)

  parsers.emph_end
    = parsers.right_flanking_delimiter_run(parsers.asterisk)
end

parsers.emph_capturing_open_and_close
  = #parsers.emph_start * #parsers.emph_end
  * Ct( Cg(Cc("delimiter"), "type")
      * Cg(Cc("emphasis"), "element")
      * Cg(C(parsers.emph_start), "content")
      * Cg(Cc(true), "is_opening")
      * Cg(Cc(true), "is_closing"))

parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
                                * Cg(Cc("emphasis"), "element")
                                * Cg(C(parsers.emph_start), "content")
                                * Cg(Cc(true), "is_opening")
                                * Cg(Cc(false), "is_closing"))

parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
                                 * Cg(Cc("emphasis"), "element")
                                 * Cg(C(parsers.emph_end), "content")
                                 * Cg(Cc(false), "is_opening")
                                 * Cg(Cc(true), "is_closing"))

parsers.emph_open_or_close  = parsers.emph_capturing_open_and_close
                            + parsers.emph_capturing_open
                            + parsers.emph_capturing_close

parsers.emph_open = parsers.emph_capturing_open_and_close
                  + parsers.emph_capturing_open

parsers.emph_close  = parsers.emph_capturing_open_and_close
                    + parsers.emph_capturing_close
```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
  -- List of references defined in the document
  local references

  -- List of note references defined in the document
  parsers.rawnotes = {}

```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```
  function self.register_link(_, tag, url, title,
                              attributes)
    local normalized_tag = self.normalize_tag(tag)
      if references[normalized_tag] == nil then
        references[normalized_tag] = {
          url = url,
          title = title,
          attributes = attributes
        }
      end
    return ""
  end

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
  function self.lookup_reference(tag)
    return references[self.normalize_tag(tag)]
  end

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
  function self.lookup_note_reference(tag)
    return parsers.rawnotes[self.normalize_tag(tag)]
  end

  parsers.title_s_direct_ref  = parsers.squote
                              * Cs((parsers.html_entities
                                  + ( parsers.anyescaped
                                    - parsers.squote
                                    - parsers.blankline^2))^0)
                              * parsers.squote

  parsers.title_d_direct_ref  = parsers.dquote
                              * Cs((parsers.html_entities
                                  + ( parsers.anyescaped
                                    - parsers.dquote
                                    - parsers.blankline^2))^0)
```

```
                              * parsers.dquote

parsers.title_p_direct_ref  = parsers.lparent
                              * Cs((parsers.html_entities
                                  + ( parsers.anyescaped
                                    - parsers.lparent
                                    - parsers.rparent
                                    - parsers.blankline^2))^0)
                              * parsers.rparent

parsers.title_direct_ref  = parsers.title_s_direct_ref
                          + parsers.title_d_direct_ref
                          + parsers.title_p_direct_ref

parsers.inline_direct_ref_inside  = parsers.lparent * parsers.spnl
                                    * Cg(parsers.url + Cc(""), "url")
                                    * parsers.spnl
                                    * Cg( parsers.title_direct_ref
                                        + Cc(""), "title")
                                    * parsers.spnl * parsers.rparent

parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
                            * Cg(parsers.url + Cc(""), "url")
                            * parsers.spnlc
                            * Cg(parsers.title + Cc(""), "title")
                            * parsers.spnlc * parsers.rparent

parsers.empty_link  = parsers.lbracket
                      * parsers.rbracket

parsers.inline_link = parsers.link_text
                      * parsers.inline_direct_ref

parsers.full_link = parsers.link_text
                  * parsers.link_label

parsers.shortcut_link = parsers.link_label
                        * -(parsers.empty_link + parsers.link_label)

parsers.collapsed_link  = parsers.link_label
                          * parsers.empty_link

parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
                        * Cg(Cc("inline"), "link_type")
                        + #(parsers.exclamation * parsers.full_link)
                        * Cg(Cc("full"), "link_type")
                        + #( parsers.exclamation
```

```
                           * parsers.collapsed_link)
                        * Cg(Cc("collapsed"), "link_type")
                        + #(parsers.exclamation * parsers.shortcut_link)
                        * Cg(Cc("shortcut"), "link_type")
                        + #(parsers.exclamation * parsers.empty_link)
                        * Cg(Cc("empty"), "link_type")

parsers.link_opening  = #parsers.inline_link
                        * Cg(Cc("inline"), "link_type")
                        + #parsers.full_link
                        * Cg(Cc("full"), "link_type")
                        + #parsers.collapsed_link
                        * Cg(Cc("collapsed"), "link_type")
                        + #parsers.shortcut_link
                        * Cg(Cc("shortcut"), "link_type")
                        + #parsers.empty_link
                        * Cg(Cc("empty_link"), "link_type")
                        + #parsers.link_text
                        * Cg(Cc("link_text"), "link_type")

parsers.note_opening  = #(parsers.circumflex * parsers.link_text)
                        * Cg(Cc("note_inline"), "link_type")

parsers.raw_note_opening  = #( parsers.lbracket
                             * parsers.circumflex
                             * parsers.link_label_body
                             * parsers.rbracket)
                           * Cg(Cc("raw_note"), "link_type")

local inline_note_element = Cg(Cc("note"), "element")
                           * parsers.note_opening
                           * Cg( parsers.circumflex
                               * parsers.lbracket, "content")

local image_element = Cg(Cc("image"), "element")
                     * parsers.image_opening
                     * Cg( parsers.exclamation
                         * parsers.lbracket, "content")

local note_element  = Cg(Cc("note"), "element")
                     * parsers.raw_note_opening
                     * Cg( parsers.lbracket
                         * parsers.circumflex, "content")

local link_element  = Cg(Cc("link"), "element")
                     * parsers.link_opening
                     * Cg(parsers.lbracket, "content")
```

```
9359
9360  local opening_elements = parsers.fail
9361
9362  if options.inlineNotes then
9363    opening_elements = opening_elements + inline_note_element
9364  end
9365
9366  opening_elements = opening_elements + image_element
9367
9368  if options.notes then
9369    opening_elements = opening_elements + note_element
9370  end
9371
9372  opening_elements = opening_elements + link_element
9373
9374  parsers.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
9375                                  * Cg(Cc(true), "is_opening")
9376                                  * Cg(Cc(false), "is_closing")
9377                                  * opening_elements)
9378
9379  parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
9380                                  * Cg(Cc("link"), "element")
9381                                  * Cg(Cc(false), "is_opening")
9382                                  * Cg(Cc(true), "is_closing")
9383                                  * ( Cg(Cc(true), "is_direct")
9384                                    * Cg( parsers.rbracket
9385                                        * #parsers.inline_direct_ref,
9386                                         "content")
9387                                    + Cg(Cc(false), "is_direct")
9388                                    * Cg(parsers.rbracket, "content")))
9389
9390  parsers.link_image_open_or_close  = parsers.link_image_opening
9391                                    + parsers.link_image_closing
9392
9393  if options.html then
9394    parsers.link_emph_precedence  = parsers.inticks
9395                                  + parsers.autolink
9396                                  + parsers.html_inline_tags
9397  else
9398    parsers.link_emph_precedence  = parsers.inticks
9399                                  + parsers.autolink
9400  end
9401
9402  parsers.link_and_emph_endline = parsers.newline
9403                                * ((parsers.check_minimal_indent
9404                                  * -V("EndlineExceptions")
9405                                  + parsers.check_optional_indent
```

```
                                    * -V("EndlineExceptions")
                                    * -V("ListStarter")) / "")
                                  * parsers.spacechar^0 / "\n"

parsers.link_and_emph_content
  = Ct( Cg(Cc("content"), "type")
      * Cg(Cs(( parsers.link_emph_precedence
              + parsers.backslash * parsers.linechar
              + parsers.link_and_emph_endline
              + (parsers.linechar
                - parsers.blankline^2
                - parsers.link_image_open_or_close
                - parsers.emph_open_or_close))^0), "content"))

parsers.link_and_emph_table
  = (parsers.link_image_opening + parsers.emph_open)
  * parsers.link_and_emph_content
  * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
    * parsers.link_and_emph_content)^1

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local function collect_link_content(t, opening_index, closing_index)
  local content = {}
  for i = opening_index, closing_index do
    content[#content + 1] = t[i].content
  end
  return util.rope_to_string(content)
end

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
local function find_link_opener(t, bottom_index, latest_index)
  for i = latest_index, bottom_index, -1 do
    local value = t[i]
    if value.type == "delimiter" and
       value.is_opening and
       ( value.element == "link"
      or value.element == "image"
      or value.element == "note")
       and not value.removed then
      if value.is_active then
        return i
      end
      value.removed = true
      return nil
```

```
      end
    end
  end

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
  local function find_next_link_closing_index(t, latest_index)
    for i = latest_index, #t do
      local value = t[i]
      if value.is_closing and
         value.element == "link" and
         not value.removed then
        return i
      end
    end
  end

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```
  local function disable_previous_link_openers(t, opening_index)
    if t[opening_index].element == "image" then
      return
    end

    for i = opening_index, 1, -1 do
      local value = t[i]
      if value.is_active and
         value.type == "delimiter" and
         value.is_opening and
         value.element == "link" then
        value.is_active = false
      end
    end
  end

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
  local function disable_range(t, opening_index, closing_index)
    for i = opening_index, closing_index do
      local value = t[i]
      if value.is_active then
        value.is_active = false
        if value.type == "delimiter" then
          value.removed = true
        end
```

```
      end
    end
  end

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
  local delete_parsed_content_in_range =
    function(t, opening_index, closing_index)
      for i = opening_index, closing_index do
        t[i].rendered = nil
      end
    end

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
  local function empty_content_in_range(t, opening_index, closing_index)
    for i = opening_index, closing_index do
      t[i].content = ''
    end
  end

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
  local function join_attributes(reference_attributes, own_attributes)
    local merged_attributes = {}
    for _, attribute in ipairs(reference_attributes or {}) do
      table.insert(merged_attributes, attribute)
    end
    for _, attribute in ipairs(own_attributes or {}) do
      table.insert(merged_attributes, attribute)
    end
    if next(merged_attributes) == nil then
      merged_attributes = nil
    end
    return merged_attributes
  end

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
  local render_link_or_image =
    function(t, opening_index, closing_index, content_end_index,
             reference)
      process_emphasis(t, opening_index, content_end_index)
      local mapped = collect_emphasis_content(t, opening_index + 1,
```

```
                                                content_end_index - 1)

    local rendered = {}
    if (t[opening_index].element == "link") then
      rendered = writer.link(mapped, reference.url,
                             reference.title, reference.attributes)
    end

    if (t[opening_index].element == "image") then
      rendered = writer.image(mapped, reference.url, reference.title,
                              reference.attributes)
    end

    if (t[opening_index].element == "note") then
      if (t[opening_index].link_type == "note_inline") then
        rendered = writer.note(mapped)
      end
      if (t[opening_index].link_type == "raw_note") then
        rendered = writer.note(reference)
      end
    end

    t[opening_index].rendered = rendered
    delete_parsed_content_in_range(t, opening_index + 1,
                                   closing_index)
    empty_content_in_range(t, opening_index, closing_index)
    disable_previous_link_openers(t, opening_index)
    disable_range(t, opening_index, closing_index)
  end

```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```
local resolve_inline_following_content =
  function(t, closing_index, match_reference, match_link_attributes)
    local content = ""
    for i = closing_index + 1, #t do
      content = content .. t[i].content
    end

    local matching_content = parsers.succeed

    if match_reference then
      matching_content = matching_content
                       * parsers.inline_direct_ref_inside
    end
```

```
9566
9567        if match_link_attributes then
9568          matching_content = matching_content
9569                           * Cg(Ct(parsers.attributes^-1), "attributes")
9570        end
9571
9572        local matched = lpeg.match(Ct( matching_content
9573                                     * Cg(Cp(), "end_position")), content)
9574
9575        local matched_count = matched.end_position - 1
9576        for i = closing_index + 1, #t do
9577          local value = t[i]
9578
9579          local chars_left = matched_count
9580          matched_count = matched_count - #value.content
9581
9582          if matched_count <= 0 then
9583            value.content = value.content:sub(chars_left + 1)
9584            break
9585          end
9586
9587          value.content = ''
9588          value.is_active = false
9589        end
9590
9591        local attributes = matched.attributes
9592        if attributes == nil or next(attributes) == nil then
9593          attributes = nil
9594        end
9595
9596        return {
9597          url = matched.url or "",
9598          title = matched.title or "",
9599          attributes = attributes
9600        }
9601      end
9602
```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```
9603    local function resolve_inline_link(t, opening_index, closing_index)
9604      local inline_content
9605        = resolve_inline_following_content(t, closing_index, true,
9606                                           t.match_link_attributes)
9607      render_link_or_image(t, opening_index, closing_index,
9608                           closing_index, inline_content)
```

```
  end

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
  local resolve_note_inline_link =
    function(t, opening_index, closing_index)
      local inline_content
        = resolve_inline_following_content(t, closing_index,
                                           false, false)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, inline_content)
    end

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
  local function resolve_shortcut_link(t, opening_index, closing_index)
    local content
      = collect_link_content(t, opening_index + 1, closing_index - 1)
    local r = self.lookup_reference(content)

    if r then
      local inline_content
        = resolve_inline_following_content(t, closing_index, false,
                                           t.match_link_attributes)
      r.attributes
        = join_attributes(r.attributes, inline_content.attributes)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, r)
    end
  end

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```
  local function resolve_raw_note_link(t, opening_index, closing_index)
    local content
      = collect_link_content(t, opening_index + 1, closing_index - 1)
    local r = self.lookup_note_reference(content)

    if r then
      local parsed_ref = self.parser_functions.parse_blocks_nested(r)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, parsed_ref)
    end
  end
```

Resolve a full link `[a][b]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```
local function resolve_full_link(t, opening_index, closing_index)
  local next_link_closing_index
    = find_next_link_closing_index(t, closing_index + 4)
  local next_link_content
    = collect_link_content(t, closing_index + 3,
                           next_link_closing_index - 1)
  local r = self.lookup_reference(next_link_content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, next_link_closing_index,
                                         false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index,
                         closing_index, r)
  end
end

```

Resolve a collapsed link `[a][]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
local function resolve_collapsed_link(t, opening_index, closing_index)
  local next_link_closing_index
    = find_next_link_closing_index(t, closing_index + 4)
  local content
    = collect_link_content(t, opening_index + 1, closing_index - 1)
  local r = self.lookup_reference(content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, closing_index, false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index,
                         closing_index, r)
  end
end

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```
local function process_links_and_emphasis(t)
  for _,value in ipairs(t) do
    value.is_active = true
  end

  for i,value in ipairs(t) do
    if not value.is_closing
        or value.type ~= "delimiter"
        or not ( value.element == "link"
              or value.element == "image"
              or value.element == "note")
        or value.removed then
      goto continue
    end

    local opener_position = find_link_opener(t, 1, i - 1)
    if (opener_position == nil) then
      goto continue
    end

    local opening_delimiter = t[opener_position]
    opening_delimiter.removed = true

    local link_type = opening_delimiter.link_type

    if (link_type == "inline") then
      resolve_inline_link(t, opener_position, i)
    end
    if (link_type == "shortcut") then
      resolve_shortcut_link(t, opener_position, i)
    end
    if (link_type == "full") then
      resolve_full_link(t, opener_position, i)
    end
    if (link_type == "collapsed") then
      resolve_collapsed_link(t, opener_position, i)
    end
    if (link_type == "note_inline") then
      resolve_note_inline_link(t, opener_position, i)
    end
    if (link_type == "raw_note") then
      resolve_raw_note_link(t, opener_position, i)
```

```
9728        end
9729
9730        ::continue::
9731      end
9732
9733      t[#t].content = t[#t].content:gsub("%s*$","")
9734
9735      process_emphasis(t, 1, #t)
9736      local final_result = collect_emphasis_content(t, 1, #t)
9737      return final_result
9738    end
9739
9740    function self.defer_link_and_emphasis_processing(delimiter_table)
9741      return writer.defer_call(function()
9742        return process_links_and_emphasis(delimiter_table)
9743      end)
9744    end
9745
```

3.1.6.8 Inline Elements (local)

```
9746    parsers.Str      = ( parsers.normalchar
9747                       * (parsers.normalchar + parsers.at)^0)
9748                     / writer.string
9749
9750    parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
9751                     / writer.string
9752
9753    parsers.Ellipsis = P("...") / writer.ellipsis
9754
9755    parsers.Smart    = parsers.Ellipsis
9756
9757    parsers.Code     = parsers.inticks / writer.code
9758
9759    if options.blankBeforeBlockquote then
9760      parsers.bqstart = parsers.fail
9761    else
9762      parsers.bqstart = parsers.blockquote_start
9763    end
9764
9765    if options.blankBeforeHeading then
9766      parsers.headerstart = parsers.fail
9767    else
9768      parsers.headerstart = parsers.atx_heading
9769    end
9770
9771    if options.blankBeforeList then
```

```
9772    parsers.interrupting_bullets = parsers.fail
9773    parsers.interrupting_enumerators = parsers.fail
9774  else
9775    parsers.interrupting_bullets
9776      = parsers.bullet(parsers.dash, true)
9777      + parsers.bullet(parsers.asterisk, true)
9778      + parsers.bullet(parsers.plus, true)
9779
9780    parsers.interrupting_enumerators
9781      = parsers.enumerator(parsers.period, true)
9782      + parsers.enumerator(parsers.rparent, true)
9783  end
9784
9785  if options.html then
9786    parsers.html_interrupting
9787      = parsers.check_trail
9788      * ( parsers.html_incomplete_open_tag
9789        + parsers.html_incomplete_close_tag
9790        + parsers.html_incomplete_open_special_tag
9791        + parsers.html_comment_start
9792        + parsers.html_cdatasection_start
9793        + parsers.html_declaration_start
9794        + parsers.html_instruction_start
9795        - parsers.html_close_special_tag
9796        - parsers.html_empty_special_tag)
9797  else
9798    parsers.html_interrupting = parsers.fail
9799  end
9800
9801  parsers.ListStarter = parsers.starter
9802
9803  parsers.EndlineExceptions
9804                    = parsers.blankline -- paragraph break
9805                    + parsers.eof       -- end of document
9806                    + parsers.bqstart
9807                    + parsers.thematic_break_lines
9808                    + parsers.interrupting_bullets
9809                    + parsers.interrupting_enumerators
9810                    + parsers.headerstart
9811                    + parsers.html_interrupting
9812
9813  parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9814
9815  parsers.endline = parsers.newline
9816                  * (parsers.check_minimal_indent
9817                    * -V("EndlineExceptions")
9818                    + parsers.check_optional_indent
```

```
                    * -V("EndlineExceptions")
                    * -V("ListStarter")) / function(_) return end
                  * parsers.spacechar^0

parsers.Endline = parsers.endline
                / writer.soft_line_break

parsers.EndlineNoSub = parsers.endline

parsers.NoSoftLineBreakEndline
                      = parsers.newline
                      * (parsers.check_minimal_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        + parsers.check_optional_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        * -V("ListStarter"))
                      * parsers.spacechar^0
                      / writer.space

parsers.EndlineBreak = parsers.backslash * parsers.endline
                                         / writer.hard_line_break

parsers.OptionalIndent
                    = parsers.spacechar^1 / writer.space

parsers.Space       = parsers.spacechar^2 * parsers.endline
                                         / writer.hard_line_break
                    + parsers.spacechar^1
                    * parsers.endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.endline
                                         / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs
                    + parsers.spacechar^1

parsers.NoSoftLineBreakSpace
                    = parsers.spacechar^2 * parsers.endline
                                         / writer.hard_line_break
                    + parsers.spacechar^1
                    * parsers.endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.endline
                                         / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs
                    + parsers.spacechar^1
```

```
9866
9867   parsers.NonbreakingEndline
9868                     = parsers.endline
9869                     / writer.nbsp
9870
9871   parsers.NonbreakingSpace
9872                   = parsers.spacechar^2 * parsers.endline
9873                                         / writer.nbsp
9874                   + parsers.spacechar^1
9875                   * parsers.endline^-1 * parsers.eof / ""
9876                   + parsers.spacechar^1 * parsers.endline
9877                                         * parsers.optionalspace
9878                                         / writer.nbsp
9879                   + parsers.spacechar^1 * parsers.optionalspace
9880                                         / writer.nbsp
9881
```

The `reader->auto_link_url` method produces an autolink to a URL or a relative reference in the output format, where `url` is the link destination and `attributes` are the optional attributes.

```
9882 function self.auto_link_url(url, attributes)
9883   return writer.link(writer.escape(url),
9884                      url, nil, attributes)
9885 end
```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```
9886 function self.auto_link_email(email, attributes)
9887   return writer.link(writer.escape(email),
9888                      "mailto:"..email,
9889                      nil, attributes)
9890 end
9891
9892   parsers.AutoLinkUrl = parsers.auto_link_url
9893                       / self.auto_link_url
9894
9895   parsers.AutoLinkEmail
9896                       = parsers.auto_link_email
9897                       / self.auto_link_email
9898
9899   parsers.AutoLinkRelativeReference
9900                       = parsers.auto_link_relative_reference
9901                       / self.auto_link_url
9902
9903   parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9904                       / self.defer_link_and_emphasis_processing
```

```
9905
9906  parsers.EscapedChar = parsers.backslash
9907                      * C(parsers.escapable) / writer.string
9908
9909  parsers.InlineHtml = Cs(parsers.html_inline_comment)
9910                     / writer.inline_html_comment
9911                     + Cs(parsers.html_any_empty_inline_tag
9912                         + parsers.html_inline_instruction
9913                         + parsers.html_inline_cdatasection
9914                         + parsers.html_inline_declaration
9915                         + parsers.html_any_open_inline_tag
9916                         + parsers.html_any_close_tag)
9917                       / writer.inline_html_tag
9918
9919  parsers.HtmlEntity = parsers.html_entities / writer.string
```

3.1.6.9 Block Elements (local)

```
9920  parsers.DisplayHtml = Cs(parsers.check_trail
9921                           * ( parsers.html_comment
9922                             + parsers.html_special_block
9923                             + parsers.html_block
9924                             + parsers.html_any_block
9925                             + parsers.html_instruction
9926                             + parsers.html_cdatasection
9927                             + parsers.html_declaration))
9928                         / writer.block_html_element
9929
9930  parsers.indented_non_blank_line = parsers.indentedline
9931                                  - parsers.blankline
9932
9933  parsers.Verbatim
9934    = Cs( parsers.check_code_trail
9935        * (parsers.line - parsers.blankline)
9936        * (( parsers.check_minimal_blank_indent_and_full_code_trail
9937           * parsers.blankline)^0
9938          * ( (parsers.check_minimal_indent / "")
9939            * parsers.check_code_trail
9940            * (parsers.line - parsers.blankline))^1)^0)
9941    / self.expandtabs / writer.verbatim
9942
9943  parsers.Blockquote   = parsers.blockquote_body
9944                       / writer.blockquote
9945
9946  parsers.ThematicBreak = parsers.thematic_break_lines
9947                        / writer.thematic_break
9948
```

```
parsers.Reference    = parsers.define_reference_parser
                     / self.register_link

parsers.Paragraph    = parsers.freeze_trail
                     * (Ct((parsers.Inline)^1)
                     * (parsers.newline + parsers.eof)
                     * parsers.unfreeze_trail
                     / writer.paragraph)

parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
                     / writer.plain
```

3.1.6.10 Lists (local)

```
if options.taskLists then
  parsers.tickbox = ( parsers.ticked_box
                    + parsers.halfticked_box
                    + parsers.unticked_box
                    ) / writer.tickbox
else
   parsers.tickbox = parsers.fail
end

parsers.list_blank = parsers.conditionally_indented_blankline

parsers.ref_or_block_list_separated
  = parsers.sep_group_no_output(parsers.list_blank)
  * parsers.minimally_indented_ref
  + parsers.block_sep_group(parsers.list_blank)
  * parsers.minimally_indented_block

parsers.ref_or_block_non_separated
  = parsers.minimally_indented_ref
  + (parsers.succeed / writer.interblocksep)
  * parsers.minimally_indented_block
  - parsers.minimally_indented_blankline

parsers.tight_list_loop_body_pair =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_non_separated,
    parsers.minimally_indented_par_or_plain_no_blank,
    (parsers.succeed / writer.interblocksep),
    (parsers.succeed / writer.paragraphsep))

parsers.loose_list_loop_body_pair =
  parsers.create_loop_body_pair(
```

```
      parsers.ref_or_block_list_separated,
      parsers.minimally_indented_par_or_plain,
      parsers.block_sep_group(parsers.list_blank),
      parsers.par_sep_group(parsers.list_blank))

  parsers.tight_list_content_loop
    = V("Block")
    * parsers.tight_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_non_separated
    * parsers.tight_list_loop_body_pair.block^0
    +  (V("Paragraph") + V("Plain"))
    * parsers.tight_list_loop_body_pair.par^0

  parsers.loose_list_content_loop
    = V("Block")
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_list_separated
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.loose_list_loop_body_pair.par^0

  parsers.list_item_tightness_condition
    = -#( parsers.list_blank^0
        * parsers.minimally_indented_ref_or_block_or_par)
    * remove_indent("li")
    + remove_indent("li")
    * parsers.fail

  parsers.indented_content_tight
    = Ct( (parsers.blankline / "")
        * #parsers.list_blank
        * remove_indent("li")
        + ( (V("Reference") + (parsers.blankline / ""))
          * parsers.check_minimal_indent
          * parsers.tight_list_content_loop
          + (V("Reference") + (parsers.blankline / ""))
          + (parsers.tickbox^-1 / writer.escape)
          * parsers.tight_list_content_loop
          )
        * parsers.list_item_tightness_condition)

  parsers.indented_content_loose
    = Ct( (parsers.blankline / "")
        * #parsers.list_blank
        + ( (V("Reference") + (parsers.blankline / "")))
```

```
10040           * parsers.check_minimal_indent
10041           * parsers.loose_list_content_loop
10042           + (V("Reference") + (parsers.blankline / ""))
10043           + (parsers.tickbox^-1 / writer.escape)
10044           * parsers.loose_list_content_loop))
10045
10046   parsers.TightListItem = function(starter)
10047     return  -parsers.ThematicBreak
10048             * parsers.add_indent(starter, "li")
10049             * parsers.indented_content_tight
10050   end
10051
10052   parsers.LooseListItem = function(starter)
10053     return  -parsers.ThematicBreak
10054             * parsers.add_indent(starter, "li")
10055             * parsers.indented_content_loose
10056             * remove_indent("li")
10057   end
10058
10059   parsers.BulletListOfType = function(bullet_type)
10060     local bullet = parsers.bullet(bullet_type)
10061     return  ( Ct( parsers.TightListItem(bullet)
10062                 * ( (parsers.check_minimal_indent / "")
10063                   * parsers.TightListItem(bullet)
10064                   )^0
10065               )
10066               * Cc(true)
10067               * -#( (parsers.list_blank^0 / "")
10068                   * parsers.check_minimal_indent
10069                   * (bullet - parsers.ThematicBreak)
10070               )
10071               + Ct( parsers.LooseListItem(bullet)
10072                   * ( (parsers.list_blank^0 / "")
10073                     * (parsers.check_minimal_indent / "")
10074                     * parsers.LooseListItem(bullet)
10075                     )^0
10076               )
10077               * Cc(false)
10078             ) / writer.bulletlist
10079   end
10080
10081   parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10082                      + parsers.BulletListOfType(parsers.asterisk)
10083                      + parsers.BulletListOfType(parsers.plus)
10084
10085   local function ordered_list(items,tight,starter)
10086     local startnum = starter[2][1]
```

```
10087     if options.startNumber then
10088       startnum = tonumber(startnum) or 1  -- fallback for '#'
10089       if startnum ~= nil then
10090         startnum = math.floor(startnum)
10091       end
10092     else
10093       startnum = nil
10094     end
10095     return writer.orderedlist(items,tight,startnum)
10096   end
10097
10098   parsers.OrderedListOfType = function(delimiter_type)
10099     local enumerator = parsers.enumerator(delimiter_type)
10100     return  Cg(enumerator, "listtype")
10101           * (Ct( parsers.TightListItem(Cb("listtype"))
10102                * ( (parsers.check_minimal_indent / "")
10103                  * parsers.TightListItem(enumerator))^0)
10104           * Cc(true)
10105           * -#((parsers.list_blank^0 / "")
10106               * parsers.check_minimal_indent * enumerator)
10107           + Ct( parsers.LooseListItem(Cb("listtype"))
10108               * ((parsers.list_blank^0 / "")
10109                 * (parsers.check_minimal_indent / "")
10110                 * parsers.LooseListItem(enumerator))^0)
10111           * Cc(false)
10112           ) * Ct(Cb("listtype")) / ordered_list
10113   end
10114
10115   parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10116                       + parsers.OrderedListOfType(parsers.rparent)
```

3.1.6.11 Blank (local)

```
10117   parsers.Blank        = parsers.blankline / ""
10118                        + V("Reference")
```

3.1.6.12 Headings (local)

```
10119   function parsers.parse_heading_text(s)
10120     local inlines = self.parser_functions.parse_inlines(s)
10121     local flatten_inlines = self.writer.flatten_inlines
10122     self.writer.flatten_inlines = true
10123     local flat_text = self.parser_functions.parse_inlines(s)
10124     flat_text = util.rope_to_string(flat_text)
10125     self.writer.flatten_inlines = flatten_inlines
10126     return {flat_text, inlines}
10127   end
10128
```

```
10129  -- parse atx header
10130  parsers.AtxHeading = parsers.check_trail_no_rem
10131                     * Cg(parsers.heading_start, "level")
10132                     * ((C( parsers.optionalspace
10133                          * parsers.hash^0
10134                          * parsers.optionalspace
10135                          * parsers.newline)
10136                        + parsers.spacechar^1
10137                        * C(parsers.line))
10138                       / strip_atx_end
10139                       / parsers.parse_heading_text)
10140                     * Cb("level")
10141                     / writer.heading
10142
10143  parsers.heading_line  = parsers.linechar^1
10144                        - parsers.thematic_break_lines
10145
10146  parsers.heading_text = parsers.heading_line
10147                       * ( (V("Endline") / "\n")
10148                         * ( parsers.heading_line
10149                           - parsers.heading_level))^0
10150                       * parsers.newline^-1
10151
10152  parsers.SetextHeading = parsers.freeze_trail
10153                        * parsers.check_trail_no_rem
10154                        * #( parsers.heading_text
10155                           * parsers.check_minimal_indent
10156                           * parsers.check_trail
10157                           * parsers.heading_level)
10158                        * Cs(parsers.heading_text)
10159                        / parsers.parse_heading_text
10160                        * parsers.check_minimal_indent_and_trail
10161                        * parsers.heading_level
10162                        * parsers.newline
10163                        * parsers.unfreeze_trail
10164                        / writer.heading
10165
10166  parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
```

3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain TEX output.

```
10167  function self.finalize_grammar(extensions)
```

Create a local writable copy of the global read-only `walkable_syntax` hash table. This table can be used by user-defined syntax extensions to insert new

PEG patterns into existing rules of the PEG grammar of markdown using the `reader->insert_pattern` method. Furthermore, built-in syntax extensions can use this table to override existing rules using the `reader->update_rule` method.

```
10168     local walkable_syntax = (function(global_walkable_syntax)
10169       local local_walkable_syntax = {}
10170       for lhs, rule in pairs(global_walkable_syntax) do
10171         local_walkable_syntax[lhs] = util.table_copy(rule)
10172       end
10173       return local_walkable_syntax
10174     end)(walkable_syntax)
```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax[`*left-hand side terminal symbol*`]` before, instead of, or after a right-hand-side terminal symbol.

```
10175     local current_extension_name = nil
10176     self.insert_pattern = function(selector, pattern, pattern_name)
10177       assert(pattern_name == nil or type(pattern_name) == "string")
10178       local _, _, lhs, pos, rhs
10179         = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
10180       assert(lhs ~= nil,
10181         [[Expected selector in form ]]
10182         .. [["LHS (before|after|instead of) RHS", not "]]
10183         .. selector .. [["]])
10184       assert(walkable_syntax[lhs] ~= nil,
10185         [[Rule ]] .. lhs
10186         .. [[ -> ... does not exist in markdown grammar]])
10187       assert(pos == "before" or pos == "after" or pos == "instead of",
10188         [[Expected positional specifier "before", "after", ]]
10189         .. [[or "instead of", not "]]
10190         .. pos .. [["]])
10191       local rule = walkable_syntax[lhs]
10192       local index = nil
10193       for current_index, current_rhs in ipairs(rule) do
10194         if type(current_rhs) == "string" and current_rhs == rhs then
10195           index = current_index
10196           if pos == "after" then
10197             index = index + 1
10198           end
10199           break
10200         end
10201       end
10202       assert(index ~= nil,
10203         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
10204           .. [[ does not exist in markdown grammar]])
10205       local accountable_pattern
10206       if current_extension_name then
10207         accountable_pattern
```

```
          = {pattern, current_extension_name, pattern_name}
      else
        assert(type(pattern) == "string",
          [[reader->insert_pattern() was called outside ]]
          .. [[an extension with ]]
          .. [[a PEG pattern instead of a rule name]])
        accountable_pattern = pattern
      end
      if pos == "instead of" then
        rule[index] = accountable_pattern
      else
        table.insert(rule, index, accountable_pattern)
      end
    end
```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```
    local syntax =
      { "Blocks",

        Blocks = V("InitializeState")
               * V("ExpectedJekyllData")
               * V("Blank")^0
```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```
               * ( V("Block")
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 + ( V("Paragraph") + V("Plain") )
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 * V("Block")
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
                     )
                   )
                 + ( V("Paragraph") + V("Plain") )
                 * ( V("Blank")^0 * parsers.eof
                   + V("Blank")^0 / writer.paragraphsep
```

```
                )
              )^0,

    ExpectedJekyllData = parsers.succeed,

    Blank              = parsers.Blank,
    Reference          = parsers.Reference,

    Blockquote         = parsers.Blockquote,
    Verbatim           = parsers.Verbatim,
    ThematicBreak      = parsers.ThematicBreak,
    BulletList         = parsers.BulletList,
    OrderedList        = parsers.OrderedList,
    DisplayHtml        = parsers.DisplayHtml,
    Heading            = parsers.Heading,
    Paragraph          = parsers.Paragraph,
    Plain              = parsers.Plain,

    ListStarter        = parsers.ListStarter,
    EndlineExceptions  = parsers.EndlineExceptions,
    NoSoftLineBreakEndlineExceptions
                       = parsers.NoSoftLineBreakEndlineExceptions,

    Str                = parsers.Str,
    Space              = parsers.Space,
    NoSoftLineBreakSpace
                       = parsers.NoSoftLineBreakSpace,
    OptionalIndent     = parsers.OptionalIndent,
    Endline            = parsers.Endline,
    EndlineNoSub       = parsers.EndlineNoSub,
    NoSoftLineBreakEndline
                       = parsers.NoSoftLineBreakEndline,
    EndlineBreak       = parsers.EndlineBreak,
    LinkAndEmph        = parsers.LinkAndEmph,
    Code               = parsers.Code,
    AutoLinkUrl        = parsers.AutoLinkUrl,
    AutoLinkEmail      = parsers.AutoLinkEmail,
    AutoLinkRelativeReference
                       = parsers.AutoLinkRelativeReference,
    InlineHtml         = parsers.InlineHtml,
    HtmlEntity         = parsers.HtmlEntity,
    EscapedChar        = parsers.EscapedChar,
    Smart              = parsers.Smart,
    Symbol             = parsers.Symbol,
    SpecialChar        = parsers.fail,
    InitializeState    = parsers.succeed,
  }
```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a function that accepts the current PEG pattern in `walkable_syntax[`left-hand side terminal symbol`]` if defined or `nil` otherwise and returns a PEG pattern that will (re)define `walkable_syntax[`left-hand side terminal symbol`]`.

```
    self.update_rule = function(rule_name, get_pattern)
      assert(current_extension_name ~= nil)
      assert(syntax[rule_name] ~= nil,
        [[Rule ]] .. rule_name
        .. [[ -> ... does not exist in markdown grammar]])
      local previous_pattern
      local extension_name
      if walkable_syntax[rule_name] then
        local previous_accountable_pattern
          = walkable_syntax[rule_name][1]
        previous_pattern = previous_accountable_pattern[1]
        extension_name
          = previous_accountable_pattern[2]
          .. ", " .. current_extension_name
      else
        previous_pattern = nil
        extension_name = current_extension_name
      end
      local pattern
```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define `walkable_syntax[`left-hand side terminal symbol`]` unless it has been previously defined.

```
function(previous_pattern)
  assert(previous_pattern == nil)
  return pattern
end
```

```
      if type(get_pattern) == "function" then
        pattern = get_pattern(previous_pattern)
      else
        assert(previous_pattern == nil,
               [[Rule ]] .. rule_name ..
               [[ has already been updated by ]] .. extension_name)
        pattern = get_pattern
      end
      local accountable_pattern = { pattern, extension_name, rule_name }
      walkable_syntax[rule_name] = { accountable_pattern }
    end
```

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

```
    local special_characters = {}
    self.add_special_character = function(c)
      table.insert(special_characters, c)
      syntax.SpecialChar = S(table.concat(special_characters, ""))
    end

    self.add_special_character("*")
    self.add_special_character("[")
    self.add_special_character("]")
    self.add_special_character("<")
    self.add_special_character("!")
    self.add_special_character("\\")
```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```
    self.initialize_named_group = function(name, value)
      local pattern = Ct("")
      if value ~= nil then
        pattern = pattern / value
      end
      syntax.InitializeState = syntax.InitializeState
                             * Cg(pattern, name)
    end
```

Add a named group for indentation.

```
    self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
    for _, extension in ipairs(extensions) do
      current_extension_name = extension.name
      extension.extend_writer(writer)
      extension.extend_reader(self)
    end
    current_extension_name = nil
```

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

```
    if options.debugExtensions then
      local sorted_lhs = {}
      for lhs, _ in pairs(walkable_syntax) do
        table.insert(sorted_lhs, lhs)
      end
      table.sort(sorted_lhs)

      local output_lines = {"{"}
```

```
    for lhs_index, lhs in ipairs(sorted_lhs) do
      local encoded_lhs = util.encode_json_string(lhs)
      table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
      local rule = walkable_syntax[lhs]
      for rhs_index, rhs in ipairs(rule) do
        local human_readable_rhs
        if type(rhs) == "string" then
          human_readable_rhs = rhs
        else
          local pattern_name
          if rhs[3] then
            pattern_name = rhs[3]
          else
            pattern_name = "Anonymous Pattern"
          end
          local extension_name = rhs[2]
          human_readable_rhs = pattern_name .. [[ (]]
                               .. extension_name .. [[)]]
        end
        local encoded_rhs
          = util.encode_json_string(human_readable_rhs)
        local output_line = [[        ]] .. encoded_rhs
        if rhs_index < #rule then
          output_line = output_line .. ","
        end
        table.insert(output_lines, output_line)
      end
      local output_line = "    ]"
      if lhs_index < #sorted_lhs then
        output_line = output_line .. ","
      end
      table.insert(output_lines, output_line)
    end
    table.insert(output_lines, "}")

    local output = table.concat(output_lines, "\n")
    local output_filename = options.debugExtensionsFileName
    local output_file = assert(io.open(output_filename, "w"),
      [[Could not open file "]] .. output_filename
      .. [[" for writing]])
    assert(output_file:write(output))
    assert(output_file:close())
  end
```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

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

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```
        if type(rhs) == "string" then
          pattern = V(rhs)
        else
          pattern = rhs[1]
          if type(pattern) == "string" then
            pattern = V(pattern)
          end
        end
        syntax[lhs] = syntax[lhs] + pattern
      end
    end
```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```
    if options.underscores then
      self.add_special_character("_")
    end

    if not options.codeSpans then
      syntax.Code = parsers.fail
    else
      self.add_special_character("`")
    end

    if not options.html then
      syntax.DisplayHtml = parsers.fail
      syntax.InlineHtml = parsers.fail
      syntax.HtmlEntity  = parsers.fail
    else
      self.add_special_character("&")
    end

    if options.preserveTabs then
      options.stripIndent = false
    end

    if not options.smartEllipses then
```

```
      syntax.Smart = parsers.fail
    else
      self.add_special_character(".")
    end

    if not options.relativeReferences then
      syntax.AutoLinkRelativeReference = parsers.fail
    end

    if options.contentLevel == "inline" then
      syntax[1] = "Inlines"
      syntax.Inlines = V("InitializeState")
                       * parsers.Inline^0
                       * ( parsers.spacing^0
                         * parsers.eof / "")
      syntax.Space = parsers.Space + parsers.blankline / writer.space
    end

    local blocks_nested_t = util.table_copy(syntax)
    blocks_nested_t.ExpectedJekyllData = parsers.succeed
    parsers.blocks_nested = Ct(blocks_nested_t)

    parsers.blocks = Ct(syntax)

    local inlines_t = util.table_copy(syntax)
    inlines_t[1] = "Inlines"
    inlines_t.Inlines = V("InitializeState")
                       * parsers.Inline^0
                       * ( parsers.spacing^0
                         * parsers.eof / "")
    parsers.inlines = Ct(inlines_t)

    local inlines_no_inline_note_t = util.table_copy(inlines_t)
    inlines_no_inline_note_t.InlineNote = parsers.fail
    parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)

    local inlines_no_html_t = util.table_copy(inlines_t)
    inlines_no_html_t.DisplayHtml = parsers.fail
    inlines_no_html_t.InlineHtml = parsers.fail
    inlines_no_html_t.HtmlEntity = parsers.fail
    parsers.inlines_no_html = Ct(inlines_no_html_t)

    local inlines_nbsp_t = util.table_copy(inlines_t)
    inlines_nbsp_t.Endline = parsers.NonbreakingEndline
    inlines_nbsp_t.Space = parsers.NonbreakingSpace
    parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

```
10489     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10490     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10491     inlines_no_link_or_emphasis_t.EndlineExceptions
10492       = parsers.EndlineExceptions - parsers.eof
10493     parsers.inlines_no_link_or_emphasis
10494       = Ct(inlines_no_link_or_emphasis_t)
```

Return a function that converts markdown string `input` into a plain TEX output and returns it..

```
10495     return function(input)
```

Unicode-normalize the input.

```
10496       if options.unicodeNormalization then
10497         local form = options.unicodeNormalizationForm
10498         if form == "nfc" then
10499           input = uni_algos.normalize.NFC(input)
10500         elseif form == "nfd" then
10501           input = uni_algos.normalize.NFD(input)
10502         elseif form == "nfkc" then
10503           input = uni_algos.normalize.NFKC(input)
10504         elseif form == "nfkd" then
10505           input = uni_algos.normalize.NFKD(input)
10506         else
10507           return writer.error(
10508             format("Unknown normalization form %s.", form))
10509         end
10510       end
```

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.

```
10511       input = input:gsub("\r\n?", "\n")
10512       if input:sub(-1) ~= "\n" then
10513         input = input .. "\n"
10514       end
```

Clear the table of references.

```
10515       references = {}
10516       local document = self.parser_functions.parse_blocks(input)
10517       local output = util.rope_to_string(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
10518       local undosep_start, undosep_end
10519       local potential_secend_start, secend_start
10520       local potential_sep_start, sep_start
10521       while true do
10522         -- find a `writer->undosep`
10523         undosep_start, undosep_end
```

```
10524           = output:find(writer.undosep_text, 1, true)
10525         if undosep_start == nil then break end
10526         -- skip any preceding section ends
10527         secend_start = undosep_start
10528         while true do
10529           potential_secend_start = secend_start - #writer.secend_text
10530           if potential_secend_start < 1
10531              or output:sub(potential_secend_start,
10532                            secend_start - 1) ~= writer.secend_text
10533              then
10534             break
10535           end
10536           secend_start = potential_secend_start
10537         end
10538         -- find an immediately preceding
10539         -- block element / paragraph separator
10540         sep_start = secend_start
10541         potential_sep_start = sep_start - #writer.interblocksep_text
10542         if potential_sep_start >= 1
10543            and output:sub(potential_sep_start,
10544                           sep_start - 1) == writer.interblocksep_text
10545            then
10546           sep_start = potential_sep_start
10547         else
10548           potential_sep_start = sep_start - #writer.paragraphsep_text
10549           if potential_sep_start >= 1
10550              and output:sub(potential_sep_start,
10551                             sep_start - 1) == writer.paragraphsep_text
10552              then
10553             sep_start = potential_sep_start
10554           end
10555         end
10556         -- remove `writer->undosep` and immediately preceding
10557         -- block element / paragraph separator
10558         output = output:sub(1, sep_start - 1)
10559               .. output:sub(secend_start, undosep_start - 1)
10560               .. output:sub(undosep_end + 1)
10561       end
10562       return output
10563     end
10564   end
10565   return self
10566 end
```

3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax

extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```
M.extensions = {}
```

3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
M.extensions.bracketed_spans = function()
  return {
    name = "built-in bracketed_spans syntax extension",
    extend_writer = function(self)
```

Define `writer->span` as a function that will transform an input bracketed span `s` with attributes `attr` to the output format.

```
      function self.span(s, attr)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererBracketedSpanAttributeContextBegin",
                self.attributes(attr),
                s,
                "\\markdownRendererBracketedSpanAttributeContextEnd{}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local span_label  = parsers.lbracket
                          * (Cs((parsers.alphanumeric^1
                              + parsers.inticks
                              + parsers.autolink
                              + V("InlineHtml")
                              + ( parsers.backslash * parsers.backslash)
                              + ( parsers.backslash
                                * (parsers.lbracket + parsers.rbracket)
                                + V("Space") + V("Endline")
                                + (parsers.any
                                  - ( parsers.newline
                                    + parsers.lbracket
                                    + parsers.rbracket
                                    + parsers.blankline^2))))^1)
                          / self.parser_functions.parse_inlines)
                          * parsers.rbracket

      local Span = span_label
                   * Ct(parsers.attributes)
```

```
                  / writer.span

      self.insert_pattern("Inline before LinkAndEmph",
                          Span, "Span")
    end
  }
end
```

3.1.7.2 Citations

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

```
M.extensions.citations = function(citation_nbsps)
  return {
    name = "built-in citations syntax extension",
    extend_writer = function(self)
```

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.

```
      function self.citations(text_cites, cites)
        local buffer = {}
        if self.flatten_inlines then
          for _,cite in ipairs(cites) do
            if cite.prenote then
              table.insert(buffer, {cite.prenote, " "})
            end
            table.insert(buffer, cite.name)
            if cite.postnote then
              table.insert(buffer, {" ", cite.postnote})
            end
          end
        else
```

```
        table.insert(buffer,
                     {"\\markdownRenderer",
                      text_cites and "TextCite" or "Cite",
                      "{", #cites, "}"})
        for _,cite in ipairs(cites) do
          table.insert(buffer,
                       {cite.suppress_author and "-" or "+", "{",
                        cite.prenote or "", "}{",
                        cite.postnote or "", "}{", cite.name, "}"})
        end
      end
      return buffer
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local citation_chars
                = parsers.alphanumeric
                + S("#$%&-+<>~/_")

    local citation_name
                = Cs(parsers.dash^-1) * parsers.at
                * Cs(citation_chars
                    * ((( citation_chars
                        + parsers.internal_punctuation
                        - parsers.comma - parsers.semicolon)
                       * -#(( parsers.internal_punctuation
                            - parsers.comma
                            - parsers.semicolon)^0
                           * -( citation_chars
                              + parsers.internal_punctuation
                              - parsers.comma
                              - parsers.semicolon)))^0
                      * citation_chars)^-1)

    local citation_body_prenote
                = Cs((parsers.alphanumeric^1
                     + parsers.bracketed
                     + parsers.inticks
                     + parsers.autolink
                     + V("InlineHtml")
                     + V("Space") + V("EndlineNoSub")
                     + (parsers.anyescaped
                       - ( parsers.newline
                         + parsers.rbracket
                         + parsers.blankline^2))
```

```
                      - ( parsers.spnl
                        * parsers.dash^-1
                        * parsers.at))^1)

    local citation_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + parsers.autolink
                       + V("InlineHtml")
                       + V("Space") + V("EndlineNoSub")
                       + (parsers.anyescaped
                         - ( parsers.newline
                           + parsers.rbracket
                           + parsers.semicolon
                           + parsers.blankline^2))
                       - (parsers.spnl * parsers.rbracket))^1)

    local citation_body_chunk
                  = ( citation_body_prenote
                    * parsers.spnlc_sep
                    + Cc("")
                    * parsers.spnlc
                  )
                  * citation_name
                  * ( parsers.internal_punctuation
                    - parsers.semicolon)^-1
                  * ( parsers.spnlc / function(_) return end
                    * citation_body_postnote
                    + Cc("")
                    * parsers.spnlc
                  )

    local citation_body
                  = citation_body_chunk
                  * ( parsers.semicolon
                    * parsers.spnlc
                    * citation_body_chunk
                  )^0

    local citation_headless_body_postnote
                  = Cs((parsers.alphanumeric^1
                       + parsers.bracketed
                       + parsers.inticks
                       + parsers.autolink
                       + V("InlineHtml")
                       + V("Space") + V("Endline")
```

```
                        + (parsers.anyescaped
                          - ( parsers.newline
                            + parsers.rbracket
                            + parsers.at
                            + parsers.semicolon + parsers.blankline^2))
                        - (parsers.spnl * parsers.rbracket))^0)

    local citation_headless_body
                  = citation_headless_body_postnote
                  * ( parsers.semicolon
                    * parsers.spnlc
                    * citation_body_chunk
                  )^0

    local citations
                  = function(text_cites, raw_cites)
        local function normalize(str)
            if str == "" then
                str = nil
            else
                str = (citation_nbsps and
                  self.parser_functions.parse_inlines_nbsp or
                  self.parser_functions.parse_inlines)(str)
            end
            return str
        end

        local cites = {}
        for i = 1,#raw_cites,4 do
            cites[#cites+1] = {
                prenote = normalize(raw_cites[i]),
                suppress_author = raw_cites[i+1] == "-",
                name = writer.identifier(raw_cites[i+2]),
                postnote = normalize(raw_cites[i+3]),
            }
        end
        return writer.citations(text_cites, cites)
    end

    local TextCitations
                  = Ct((parsers.spnlc
                  * Cc("")
                  * citation_name
                  * ((parsers.spnlc
                      * parsers.lbracket
                      * citation_headless_body
                      * parsers.rbracket) + Cc("")))^1)
```

```
                    / function(raw_cites)
                        return citations(true, raw_cites)
                      end

      local ParenthesizedCitations
                    = Ct((parsers.spnlc
                    * parsers.lbracket
                    * citation_body
                    * parsers.rbracket)^1)
                    / function(raw_cites)
                        return citations(false, raw_cites)
                      end

      local Citations = TextCitations + ParenthesizedCitations

      self.insert_pattern("Inline before LinkAndEmph",
                          Citations, "Citations")

      self.add_special_character("@")
      self.add_special_character("-")
    end
  }
end
```

3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
M.extensions.content_blocks = function(language_map)
```

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

```
  local languages_json = (function()
    local base, prev, curr
    for _, pathname in ipairs{kpse.lookup(language_map,
                                          {all=true})} do
      local file = io.open(pathname, "r")
      if not file then goto continue end
      local input = assert(file:read("*a"))
      assert(file:close())
      local json = input:gsub('("[^\n]-"):','[%1]=')
      curr = load("_ENV = {}; return "..json)()
      if type(curr) == "table" then
        if base == nil then
```

```
10803           base = curr
10804         else
10805           setmetatable(prev, { __index = curr })
10806         end
10807         prev = curr
10808       end
10809       ::continue::
10810     end
10811     return base or {}
10812   end)()
10813
10814   return {
10815     name = "built-in content_blocks syntax extension",
10816     extend_writer = function(self)
```

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

```
10817       function self.contentblock(src,suf,type,tit)
10818         if not self.is_writing then return "" end
10819         src = src.."."..suf
10820         suf = suf:lower()
10821         if type == "onlineimage" then
10822           return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
10823                   "{",self.string(src),"}",
10824                   "{",self.uri(src),"}",
10825                   "{",self.string(tit or ""),"}"}
10826         elseif languages_json[suf] then
10827           return {"\\markdownRendererContentBlockCode{",suf,"}",
10828                   "{",self.string(languages_json[suf]),"}",
10829                   "{",self.string(src),"}",
10830                   "{",self.uri(src),"}",
10831                   "{",self.string(tit or ""),"}"}
10832         else
10833           return {"\\markdownRendererContentBlock{",suf,"}",
10834                   "{",self.string(src),"}",
10835                   "{",self.uri(src),"}",
10836                   "{",self.string(tit or ""),"}"}
10837         end
10838       end
10839     end, extend_reader = function(self)
10840       local parsers = self.parsers
10841       local writer = self.writer
10842
10843       local contentblock_tail
10844                   = parsers.optionaltitle
```

```
                * (parsers.newline + parsers.eof)

    -- case insensitive online image suffix:
    local onlineimagesuffix
                = (function(...)
                    local parser = nil
                    for _, suffix in ipairs({...}) do
                      local pattern=nil
                      for i=1,#suffix do
                        local char=suffix:sub(i,i)
                        char = S(char:lower()..char:upper())
                        if pattern == nil then
                          pattern = char
                        else
                          pattern = pattern * char
                        end
                      end
                      if parser == nil then
                        parser = pattern
                      else
                        parser = parser + pattern
                      end
                    end
                    return parser
                  end)("png", "jpg", "jpeg", "gif", "tif", "tiff")

    -- online image url for iA Writer content blocks with
    -- mandatory suffix, allowing nested brackets:
    local onlineimageurl
                = (parsers.less
                  * Cs((parsers.anyescaped
                       - parsers.more
                       - parsers.spacing
                       - #(parsers.period
                          * onlineimagesuffix
                          * parsers.more
                          * contentblock_tail))^0)
                  * parsers.period
                  * Cs(onlineimagesuffix)
                  * parsers.more
                  + (Cs((parsers.inparens
                        + (parsers.anyescaped
                          - parsers.spacing
                          - parsers.rparent
                          - #(parsers.period
                             * onlineimagesuffix
                             * contentblock_tail)))^0)
```

```
                        * parsers.period
                        * Cs(onlineimagesuffix))
                      ) * Cc("onlineimage")

      -- filename for iA Writer content blocks with mandatory suffix:
      local localfilepath
                    = parsers.slash
                    * Cs((parsers.anyescaped
                         - parsers.tab
                         - parsers.newline
                         - #(parsers.period
                            * parsers.alphanumeric^1
                            * contentblock_tail))^1)
                    * parsers.period
                    * Cs(parsers.alphanumeric^1)
                    * Cc("localfile")

      local ContentBlock
                    = parsers.check_trail_no_rem
                    * (localfilepath + onlineimageurl)
                    * contentblock_tail
                    / writer.contentblock

      self.insert_pattern("Block before Blockquote",
                          ContentBlock, "ContentBlock")
    end
  }
end
```

3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```
M.extensions.definition_lists = function(tight_lists)
  return {
    name = "built-in definition_lists syntax extension",
    extend_writer = function(self)
```

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

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1]
```

```
            = {"\\markdownRendererDlDefinitionBegin ",def,
               "\\markdownRendererDlDefinitionEnd "}
      end
      retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
      return retVal
    end

    function self.definitionlist(items,tight)
      if not self.is_writing then return "" end
      local buffer = {}
      for _,item in ipairs(items) do
        buffer[#buffer + 1] = dlitem(item.term, item.definitions)
      end
      if tight and tight_lists then
        return {"\\markdownRendererDlBeginTight\n", buffer,
          "\n\\markdownRendererDlEndTight"}
      else
        return {"\\markdownRendererDlBegin\n", buffer,
          "\n\\markdownRendererDlEnd"}
      end
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local defstartchar = S("~:")

    local defstart
      = parsers.check_trail_length(0) * defstartchar
      * #parsers.spacing
      * (parsers.tab + parsers.space^-3)
      + parsers.check_trail_length(1)
      * defstartchar * #parsers.spacing
      * (parsers.tab + parsers.space^-2)
      + parsers.check_trail_length(2)
      * defstartchar * #parsers.spacing
      * (parsers.tab + parsers.space^-1)
      + parsers.check_trail_length(3)
      * defstartchar * #parsers.spacing

    local indented_line
      = (parsers.check_minimal_indent / "")
      * parsers.check_code_trail * parsers.line

    local blank
      = parsers.check_minimal_blank_indent_and_any_trail
      * parsers.optionalspace * parsers.newline
```

```
10975
10976       local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10977
10978       local indented_blocks = function(bl)
10979         return Cs( bl
10980               * (blank^1 * (parsers.check_minimal_indent / "")
10981                 * parsers.check_code_trail * -parsers.blankline * bl)^0
10982               * (blank^1 + parsers.eof))
10983       end
10984
10985       local function definition_list_item(term, defs, _)
10986         return { term = self.parser_functions.parse_inlines(term),
10987                  definitions = defs }
10988       end
10989
10990       local DefinitionListItemLoose
10991         = C(parsers.line) * blank^0
10992         * Ct((parsers.check_minimal_indent * (defstart
10993               * indented_blocks(dlchunk)
10994               / self.parser_functions.parse_blocks_nested))^1)
10995         * Cc(false) / definition_list_item
10996
10997       local DefinitionListItemTight
10998         = C(parsers.line)
10999         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11000               / self.parser_functions.parse_blocks_nested))^1)
11001         * Cc(true) / definition_list_item
11002
11003       local DefinitionList
11004         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
11005           + Ct(DefinitionListItemTight^1)
11006           * (blank^0
11007             * -DefinitionListItemLoose * Cc(true))
11008           ) / writer.definitionlist
11009
11010       self.insert_pattern("Block after Heading",
11011                           DefinitionList, "DefinitionList")
11012     end
11013   }
11014 end
```

3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```
11015 M.extensions.fancy_lists = function()
11016   return {
```

```
11017     name = "built-in fancy_lists syntax extension",
11018     extend_writer = function(self)
11019       local options = self.options
11020
```

Define `writer->fancylist` as a function that will transform an input ordered list to the output format, where:

- `items` is an array of the list items,
- `tight` specifies, whether the list is tight or not,
- `startnum` is the number of the first list item,
- `numstyle` is the style of the list item labels from among the following:
  - `Decimal` – decimal arabic numbers,
  - `LowerRoman` – lower roman numbers,
  - `UpperRoman` – upper roman numbers,
  - `LowerAlpha` – lower ASCII alphabetic characters, and
  - `UpperAlpha` – upper ASCII alphabetic characters, and
- `numdelim` is the style of delimiters between list item labels and texts from among the following:
  - `Default` – default style,
  - `OneParen` – parentheses, and
  - `Period` – periods.

```
11021       function self.fancylist(items,tight,startnum,numstyle,numdelim)
11022         if not self.is_writing then return "" end
11023         local buffer = {}
11024         local num = startnum
11025         for _,item in ipairs(items) do
11026           if item ~= "" then
11027             buffer[#buffer + 1] = self.fancyitem(item,num)
11028           end
11029           if num ~= nil and item ~= "" then
11030             num = num + 1
11031           end
11032         end
11033         local contents = util.intersperse(buffer,"\n")
11034         if tight and options.tightLists then
11035           return {"\\markdownRendererFancyOlBeginTight{",
11036                   numstyle,"}{",numdelim,"}",contents,
11037                   "\n\\markdownRendererFancyOlEndTight "}
11038         else
11039           return {"\\markdownRendererFancyOlBegin{",
11040                   numstyle,"}{",numdelim,"}",contents,
11041                   "\n\\markdownRendererFancyOlEnd "}
```

```
11042         end
11043       end
```

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

```
11044       function self.fancyitem(s,num)
11045         if num ~= nil then
11046           return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
11047                   "\\markdownRendererFancyOlItemEnd "}
11048         else
11049           return {"\\markdownRendererFancyOlItem ",s,
11050                   "\\markdownRendererFancyOlItemEnd "}
11051         end
11052       end
11053     end, extend_reader = function(self)
11054       local parsers = self.parsers
11055       local options = self.options
11056       local writer = self.writer
11057
11058       local function combine_markers_and_delims(markers, delims)
11059         local markers_table = {}
11060         for _,marker in ipairs(markers) do
11061           local start_marker
11062           local continuation_marker
11063           if type(marker) == "table" then
11064             start_marker = marker[1]
11065             continuation_marker = marker[2]
11066           else
11067             start_marker = marker
11068             continuation_marker = marker
11069           end
11070           for _,delim in ipairs(delims) do
11071             table.insert(markers_table,
11072                          {start_marker, continuation_marker, delim})
11073           end
11074         end
11075         return markers_table
11076       end
11077
11078       local function join_table_with_func(func, markers_table)
11079         local pattern = func(table.unpack(markers_table[1]))
11080         for i = 2, #markers_table do
11081           pattern = pattern + func(table.unpack(markers_table[i]))
11082         end
11083         return pattern
11084       end
```

```
11085
11086       local lowercase_letter_marker = R("az")
11087       local uppercase_letter_marker = R("AZ")
11088
11089       local roman_marker = function(chars)
11090         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11091         local l, x, v, i
11092           = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11093         return  m^-3
11094                 * (c*m + c*d + d^-1 * c^-3)
11095                 * (x*c + x*l + l^-1 * x^-3)
11096                 * (i*x + i*v + v^-1 * i^-3)
11097       end
11098
11099       local lowercase_roman_marker
11100         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11101       local uppercase_roman_marker
11102         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11103
11104       local lowercase_opening_roman_marker  = P("i")
11105       local uppercase_opening_roman_marker  = P("I")
11106
11107       local digit_marker = parsers.dig * parsers.dig^-8
11108
11109       local markers = {
11110         {lowercase_opening_roman_marker, lowercase_roman_marker},
11111         {uppercase_opening_roman_marker, uppercase_roman_marker},
11112         lowercase_letter_marker,
11113         uppercase_letter_marker,
11114         lowercase_roman_marker,
11115         uppercase_roman_marker,
11116         digit_marker
11117       }
11118
11119       local delims = {
11120         parsers.period,
11121         parsers.rparent
11122       }
11123
11124       local markers_table = combine_markers_and_delims(markers, delims)
11125
11126       local function enumerator(start_marker, _,
11127                                 delimiter_type, interrupting)
11128         local delimiter_range
11129         local allowed_end
11130         if interrupting then
11131           delimiter_range = P("1")
```

```
11132           allowed_end = C(parsers.spacechar^1) * #parsers.linechar
11133         else
11134           delimiter_range = start_marker
11135           allowed_end = C(parsers.spacechar^1)
11136                       + #(parsers.newline + parsers.eof)
11137         end
11138
11139         return parsers.check_trail
11140                 * Ct(C(delimiter_range) * C(delimiter_type))
11141                 * allowed_end
11142       end
11143
11144       local starter = join_table_with_func(enumerator, markers_table)
11145
11146       local TightListItem = function(starter)
11147         return  parsers.add_indent(starter, "li")
11148                 * parsers.indented_content_tight
11149       end
11150
11151       local LooseListItem = function(starter)
11152         return  parsers.add_indent(starter, "li")
11153                 * parsers.indented_content_loose
11154                 * remove_indent("li")
11155       end
11156
11157       local function roman2number(roman)
11158         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11159                          ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11160         local numeral = 0
11161
11162         local i = 1
11163         local len = string.len(roman)
11164         while i < len do
11165           local z1, z2 = romans[ string.sub(roman, i, i) ],
11166                          romans[ string.sub(roman, i+1, i+1) ]
11167           if z1 < z2 then
11168               numeral = numeral + (z2 - z1)
11169               i = i + 2
11170           else
11171               numeral = numeral + z1
11172               i = i + 1
11173           end
11174         end
11175         if i <= len then
11176           numeral = numeral + romans[ string.sub(roman,i,i) ]
11177         end
11178         return numeral
```

```
11179       end
11180
11181       local function sniffstyle(numstr, delimend)
11182         local numdelim
11183         if delimend == ")" then
11184           numdelim = "OneParen"
11185         elseif delimend == "." then
11186           numdelim = "Period"
11187         else
11188           numdelim = "Default"
11189         end
11190
11191         local num
11192         num = numstr:match("^([I])$")
11193         if num then
11194           return roman2number(num), "UpperRoman", numdelim
11195         end
11196         num = numstr:match("^([i])$")
11197         if num then
11198           return roman2number(string.upper(num)), "LowerRoman", numdelim
11199         end
11200         num = numstr:match("^([A-Z])$")
11201         if num then
11202           return string.byte(num) - string.byte("A") + 1,
11203                  "UpperAlpha", numdelim
11204         end
11205         num = numstr:match("^([a-z])$")
11206         if num then
11207           return string.byte(num) - string.byte("a") + 1,
11208                  "LowerAlpha", numdelim
11209         end
11210         num = numstr:match("^([IVXLCDM]+)")
11211         if num then
11212           return roman2number(num), "UpperRoman", numdelim
11213         end
11214         num = numstr:match("^([ivxlcdm]+)")
11215         if num then
11216           return roman2number(string.upper(num)), "LowerRoman", numdelim
11217         end
11218         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11219       end
11220
11221       local function fancylist(items,tight,start)
11222         local startnum, numstyle, numdelim
11223           = sniffstyle(start[2][1], start[2][2])
11224         return writer.fancylist(items,tight,
11225                                 options.startNumber and startnum or 1,
```

```
                                numstyle or "Decimal",
                                numdelim or "Default")
      end

      local FancyListOfType
        = function(start_marker, continuation_marker, delimiter_type)
          local enumerator_start
            = enumerator(start_marker, continuation_marker,
                         delimiter_type)
          local enumerator_cont
            = enumerator(continuation_marker, continuation_marker,
                         delimiter_type)
          return Cg(enumerator_start, "listtype")
               * (Ct( TightListItem(Cb("listtype"))
                   * ((parsers.check_minimal_indent / "")
                   * TightListItem(enumerator_cont))^0)
               * Cc(true)
               * -#((parsers.conditionally_indented_blankline^0 / "")
                   * parsers.check_minimal_indent * enumerator_cont)
               + Ct( LooseListItem(Cb("listtype"))
                   * ((parsers.conditionally_indented_blankline^0 / "")
                     * (parsers.check_minimal_indent / "")
                     * LooseListItem(enumerator_cont))^0)
               * Cc(false)
               ) * Ct(Cb("listtype")) / fancylist
        end

      local FancyList
        = join_table_with_func(FancyListOfType, markers_table)

      local ListStarter = starter

      self.update_rule("OrderedList", FancyList)
      self.update_rule("ListStarter", ListStarter)
    end
  }
end
```

3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the

syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```
M.extensions.fenced_code = function(blank_before_code_fence,
                                    allow_attributes,
                                    allow_raw_blocks)
  return {
    name = "built-in fenced_code syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.fencedCode(s, i, attr)
        if not self.is_writing then return "" end
        s = s:gsub("\n$", "")
        local buf = {}
        if attr ~= nil then
          table.insert(buf,
            {"\\markdownRendererFencedCodeAttributeContextBegin",
             self.attributes(attr)})
        end
        local name = util.cache_verbatim(options.cacheDir, s)
        table.insert(buf,
          {"\\markdownRendererInputFencedCode{",
           name,"}{",self.string(i),"}{",self.infostring(i),"}"})
        if attr ~= nil then
          table.insert(buf,
            "\\markdownRendererFencedCodeAttributeContextEnd{}")
        end
        return buf
      end

```

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

```
      if allow_raw_blocks then
        function self.rawBlock(s, attr)
          if not self.is_writing then return "" end
          s = s:gsub("\n$", "")
          local name = util.cache_verbatim(options.cacheDir, s)
          return {"\\markdownRendererInputRawBlock{",
                  name,"}{", self.string(attr),"}"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
```

```
11302       local writer = self.writer
11303
11304       local function captures_geq_length(_,i,a,b)
11305         return #a >= #b and i
11306       end
11307
11308       local function strip_enclosing_whitespaces(str)
11309         return str:gsub("^%s*(.-)%s*$", "%1")
11310       end
11311
11312       local tilde_infostring = Cs(Cs((V("HtmlEntity")
11313                                     + parsers.anyescaped
11314                                     - parsers.newline)^0)
11315                                / strip_enclosing_whitespaces)
11316
11317       local backtick_infostring
11318         = Cs( Cs((V("HtmlEntity")
11319             + ( -#(parsers.backslash * parsers.backtick)
11320               * parsers.anyescaped)
11321               - parsers.newline
11322               - parsers.backtick)^0)
11323         / strip_enclosing_whitespaces)
11324
11325       local fenceindent
11326
11327       local function has_trail(indent_table)
11328         return indent_table ~= nil and
11329           indent_table.trail ~= nil and
11330           next(indent_table.trail) ~= nil
11331       end
11332
11333       local function has_indents(indent_table)
11334         return indent_table ~= nil and
11335           indent_table.indents ~= nil and
11336           next(indent_table.indents) ~= nil
11337       end
11338
11339       local function get_last_indent_name(indent_table)
11340         if has_indents(indent_table) then
11341           return indent_table.indents[#indent_table.indents].name
11342         end
11343       end
11344
11345       local count_fenced_start_indent =
11346         function(_, _, indent_table, trail)
11347           local last_indent_name = get_last_indent_name(indent_table)
11348           fenceindent = 0
```

```
11349           if last_indent_name ~= "li" then
11350             fenceindent = #trail
11351           end
11352           return true
11353         end
11354
11355       local fencehead = function(char, infostring)
11356         return Cmt( Cb("indent_info")
11357                   * parsers.check_trail, count_fenced_start_indent)
11358              * Cg(char^3, "fencelength")
11359              * parsers.optionalspace
11360              * infostring
11361              * (parsers.newline + parsers.eof)
11362       end
11363
11364       local fencetail = function(char)
11365         return parsers.check_trail_no_rem
11366              * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11367              * parsers.optionalspace * (parsers.newline + parsers.eof)
11368              + parsers.eof
11369       end
11370
11371       local process_fenced_line =
11372         function(s, i, -- luacheck: ignore s i
11373                  indent_table, line_content, is_blank)
11374           local remainder = ""
11375           if has_trail(indent_table) then
11376             remainder = indent_table.trail.internal_remainder
11377           end
11378
11379           if is_blank
11380              and get_last_indent_name(indent_table) == "li" then
11381             remainder = ""
11382           end
11383
11384           local str = remainder .. line_content
11385           local index = 1
11386           local remaining = fenceindent
11387
11388           while true do
11389             local c = str:sub(index, index)
11390             if c == " " and remaining > 0 then
11391               remaining = remaining - 1
11392               index = index + 1
11393             elseif c == "\t" and remaining > 3 then
11394               remaining = remaining - 4
11395               index = index + 1
```

```
        else
          break
        end
      end

      return true, str:sub(index)
    end

  local fencedline = function(char)
    return Cmt( Cb("indent_info")
              * C(parsers.line - fencetail(char))
              * Cc(false), process_fenced_line)
  end

  local blankfencedline
    = Cmt( Cb("indent_info")
         * C(parsers.blankline)
         * Cc(true), process_fenced_line)

  local TildeFencedCode
    = fencehead(parsers.tilde, tilde_infostring)
    * Cs(( (parsers.check_minimal_blank_indent / "")
         * blankfencedline
         + ( parsers.check_minimal_indent / "")
           * fencedline(parsers.tilde))^0)
    * ( (parsers.check_minimal_indent / "")
      * fencetail(parsers.tilde) + parsers.succeed)

  local BacktickFencedCode
         = fencehead(parsers.backtick, backtick_infostring)
         * Cs(( (parsers.check_minimal_blank_indent / "")
              * blankfencedline
              + (parsers.check_minimal_indent / "")
              * fencedline(parsers.backtick))^0)
         * ( (parsers.check_minimal_indent / "")
           * fencetail(parsers.backtick) + parsers.succeed)

  local infostring_with_attributes
                    = Ct(C((parsers.linechar
                          - ( parsers.optionalspace
                            * parsers.attributes))^0)
                        * parsers.optionalspace
                        * Ct(parsers.attributes))

  local FencedCode
    = ((TildeFencedCode + BacktickFencedCode)
    / function(infostring, code)
```

```
11443           local expanded_code = self.expandtabs(code)
11444
11445           if allow_raw_blocks then
11446             local raw_attr = lpeg.match(parsers.raw_attribute,
11447                                         infostring)
11448             if raw_attr then
11449               return writer.rawBlock(expanded_code, raw_attr)
11450             end
11451           end
11452
11453           local attr = nil
11454           if allow_attributes then
11455             local match = lpeg.match(infostring_with_attributes,
11456                                      infostring)
11457             if match then
11458               infostring, attr = table.unpack(match)
11459             end
11460           end
11461           return writer.fencedCode(expanded_code, infostring, attr)
11462         end)
11463
11464       self.insert_pattern("Block after Verbatim",
11465                           FencedCode, "FencedCode")
11466
11467       local fencestart
11468       if blank_before_code_fence then
11469         fencestart = parsers.fail
11470       else
11471         fencestart = fencehead(parsers.backtick, backtick_infostring)
11472                    + fencehead(parsers.tilde, tilde_infostring)
11473       end
11474
11475       self.update_rule("EndlineExceptions", function(previous_pattern)
11476         if previous_pattern == nil then
11477           previous_pattern = parsers.EndlineExceptions
11478         end
11479         return previous_pattern + fencestart
11480       end)
11481
11482       self.add_special_character("`")
11483       self.add_special_character("~")
11484     end
11485   }
11486 end
```

3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```
M.extensions.fenced_divs = function(blank_before_div_fence)
  return {
    name = "built-in fenced_divs syntax extension",
    extend_writer = function(self)
```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```
      function self.div_begin(attributes)
        local start_output
          = {"\\markdownRendererFencedDivAttributeContextBegin\n",
             self.attributes(attributes)}
        local end_output
          = {"\\markdownRendererFencedDivAttributeContextEnd{}"}
        return self.push_attributes(
          "div", attributes, start_output, end_output)
      end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
      function self.div_end()
        return self.pop_attributes("div")
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
      local fenced_div_infostring
                              = C((parsers.linechar
                                 - ( parsers.spacechar^1
                                   * parsers.colon^1))^1)

      local fenced_div_begin = parsers.nonindentspace
                              * parsers.colon^3
                              * parsers.optionalspace
                              * fenced_div_infostring
                              * ( parsers.spacechar^1
                                * parsers.colon^1)^0
                              * parsers.optionalspace
                              * (parsers.newline + parsers.eof)

      local fenced_div_end = parsers.nonindentspace
                            * parsers.colon^3
```

```
                          * parsers.optionalspace
                          * (parsers.newline + parsers.eof)
```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```
    self.initialize_named_group("fenced_div_level", "0")
    self.initialize_named_group("fenced_div_num_opening_indents")

    local function increment_div_level()
      local push_indent_table =
        function(s, i, indent_table, -- luacheck: ignore s i
                 fenced_div_num_opening_indents, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level) + 1
          local num_opening_indents = 0
          if indent_table.indents ~= nil then
            num_opening_indents = #indent_table.indents
          end
          fenced_div_num_opening_indents[fenced_div_level]
            = num_opening_indents
          return true, fenced_div_num_opening_indents
        end

      local increment_level =
        function(s, i, fenced_div_level) -- luacheck: ignore s i
          fenced_div_level = tonumber(fenced_div_level) + 1
          return true, tostring(fenced_div_level)
        end

      return Cg( Cmt( Cb("indent_info")
                    * Cb("fenced_div_num_opening_indents")
                    * Cb("fenced_div_level"), push_indent_table)
               , "fenced_div_num_opening_indents")
           * Cg( Cmt( Cb("fenced_div_level"), increment_level)
               , "fenced_div_level")
    end

    local function decrement_div_level()
      local pop_indent_table =
        function(s, i, -- luacheck: ignore s i
                 fenced_div_indent_table, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level)
          fenced_div_indent_table[fenced_div_level] = nil
          return true, tostring(fenced_div_level - 1)
```

```
11562           end
11563
11564         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11565                       * Cb("fenced_div_level"), pop_indent_table)
11566                  , "fenced_div_level")
11567       end
11568
11569
11570       local non_fenced_div_block
11571         = parsers.check_minimal_indent * V("Block")
11572         - parsers.check_minimal_indent_and_trail * fenced_div_end
11573
11574       local non_fenced_div_paragraph
11575         = parsers.check_minimal_indent * V("Paragraph")
11576         - parsers.check_minimal_indent_and_trail * fenced_div_end
11577
11578       local blank = parsers.minimally_indented_blank
11579
11580       local block_separated = parsers.block_sep_group(blank)
11581                             * non_fenced_div_block
11582
11583       local loop_body_pair
11584         = parsers.create_loop_body_pair(block_separated,
11585                                         non_fenced_div_paragraph,
11586                                         parsers.block_sep_group(blank),
11587                                         parsers.par_sep_group(blank))
11588
11589       local content_loop  = ( non_fenced_div_block
11590                             * loop_body_pair.block^0
11591                             + non_fenced_div_paragraph
11592                             * block_separated
11593                             * loop_body_pair.block^0
11594                             + non_fenced_div_paragraph
11595                             * loop_body_pair.par^0)
11596                           * blank^0
11597
11598       local FencedDiv = fenced_div_begin
11599                       / function (infostring)
11600                           local attr
11601                             = lpeg.match(Ct(parsers.attributes),
11602                                          infostring)
11603                           if attr == nil then
11604                             attr = {"." .. infostring}
11605                           end
11606                           return attr
11607                         end
11608                       / writer.div_begin
```

```
                    * increment_div_level()
                    * parsers.skipblanklines
                    * Ct(content_loop)
                    * parsers.minimally_indented_blank^0
                    * parsers.check_minimal_indent_and_trail
                    * fenced_div_end
                    * decrement_div_level()
                    * (Cc("") / writer.div_end)

    self.insert_pattern("Block after Verbatim",
                        FencedDiv, "FencedDiv")

    self.add_special_character(":")

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```
    local function is_inside_div()
      local check_div_level =
        function(s, i, fenced_div_level) -- luacheck: ignore s i
          fenced_div_level = tonumber(fenced_div_level)
          return fenced_div_level > 0
        end

      return Cmt(Cb("fenced_div_level"), check_div_level)
    end

    local function check_indent()
      local compare_indent =
        function(s, i, indent_table, -- luacheck: ignore s i
                 fenced_div_num_opening_indents, fenced_div_level)
          fenced_div_level = tonumber(fenced_div_level)
          local num_current_indents
            = ( indent_table.current_line_indents ~= nil and
                #indent_table.current_line_indents) or 0
          local num_opening_indents
            = fenced_div_num_opening_indents[fenced_div_level]
          return num_current_indents == num_opening_indents
        end

      return Cmt( Cb("indent_info")
                * Cb("fenced_div_num_opening_indents")
                * Cb("fenced_div_level"), compare_indent)
    end

    local fencestart = is_inside_div()
```

```
                        * fenced_div_end
                        * check_indent()

      if not blank_before_div_fence then
        self.update_rule("EndlineExceptions", function(previous_pattern)
          if previous_pattern == nil then
            previous_pattern = parsers.EndlineExceptions
          end
          return previous_pattern + fencestart
        end)
      end
    end
  }
end
```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```
M.extensions.header_attributes = function()
  return {
    name = "built-in header_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local function strip_atx_end(s)
        return s:gsub("%s+#*%s*$","")
      end

      local AtxHeading = Cg(parsers.heading_start, "level")
                       * parsers.optionalspace
                       * (C(((parsers.linechar
                              - (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline))
                             * (parsers.linechar
                               - parsers.lbrace)^0)^1)
                             / strip_atx_end
                             / parsers.parse_heading_text)
                       * Cg(Ct(parsers.newline
                              + (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline)), "attributes")
                       * Cb("level")
                       * Cb("attributes")
```

```
                      / writer.heading

      local function strip_trailing_spaces(s)
        return s:gsub("%s*$","")
      end

      local heading_line  = (parsers.linechar
                            - (parsers.attributes
                              * parsers.optionalspace
                              * parsers.newline))^1
                          - parsers.thematic_break_lines

      local heading_text
        = heading_line
        * ( (V("Endline") / "\n")
          * (heading_line - parsers.heading_level))^0
        * parsers.newline^-1

      local SetextHeading
        = parsers.freeze_trail * parsers.check_trail_no_rem
        * #(heading_text
           * (parsers.attributes
             * parsers.optionalspace
             * parsers.newline)^-1
           * parsers.check_minimal_indent
           * parsers.check_trail
           * parsers.heading_level)
        * Cs(heading_text) / strip_trailing_spaces
        / parsers.parse_heading_text
        * Cg(Ct((parsers.attributes
              * parsers.optionalspace
              * parsers.newline)^-1), "attributes")
        * parsers.check_minimal_indent_and_trail * parsers.heading_level
        * Cb("attributes")
        * parsers.newline
        * parsers.unfreeze_trail
        / writer.heading

      local Heading = AtxHeading + SetextHeading
      self.update_rule("Heading", Heading)
    end
  }
end
```

3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```
M.extensions.inline_code_attributes = function()
  return {
    name = "built-in inline_code_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local writer = self.writer

      local CodeWithAttributes = parsers.inticks
                               * Ct(parsers.attributes)
                               / writer.code

      self.insert_pattern("Inline before Code",
                          CodeWithAttributes,
                          "CodeWithAttributes")
    end
  }
end
```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
M.extensions.line_blocks = function()
  return {
    name = "built-in line_blocks syntax extension",
    extend_writer = function(self)
```

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```
      function self.lineblock(lines)
        if not self.is_writing then return "" end
        local buffer = {}
        for i = 1, #lines - 1 do
          buffer[#buffer + 1] = { lines[i], self.hard_line_break }
        end
        buffer[#buffer + 1] = lines[#lines]

        return {"\\markdownRendererLineBlockBegin\n"
                  ,buffer,
                  "\n\\markdownRendererLineBlockEnd "}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

```

```
11774       local LineBlock
11775         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
11776                  * ((parsers.space)/entities.char_entity("nbsp"))^0
11777                  * parsers.linechar^0 * (parsers.newline/""))
11778                  * (-parsers.pipe
11779                    * (parsers.space^1/" ")
11780                    * parsers.linechar^1
11781                    * (parsers.newline/"")
11782                    )^0
11783                  * (parsers.blankline/"")^0)
11784               / self.parser_functions.parse_inlines)^1)
11785         / writer.lineblock
11786
11787       self.insert_pattern("Block after Blockquote",
11788                           LineBlock, "LineBlock")
11789     end
11790   }
11791 end
```

3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```
11792 M.extensions.mark = function()
11793   return {
11794     name = "built-in mark syntax extension",
11795     extend_writer = function(self)
```

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

```
11796       function self.mark(s)
11797         if self.flatten_inlines then return s end
11798         return {"\\markdownRendererMark{", s, "}"}
11799       end
11800     end, extend_reader = function(self)
11801       local parsers = self.parsers
11802       local writer = self.writer
11803
11804       local doubleequals = P("==")
11805
11806       local Mark
11807         = parsers.between(V("Inline"), doubleequals, doubleequals)
11808         / function (inlines) return writer.mark(inlines) end
11809
11810       self.add_special_character("=")
11811       self.insert_pattern("Inline before LinkAndEmph",
11812                           Mark, "Mark")
11813     end
11814   }
```

```
end
```

3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
M.extensions.link_attributes = function()
  return {
    name = "built-in link_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local parsers = self.parsers
      local options = self.options

```

The following patterns define link reference definitions with attributes.

```
      local define_reference_parser
        = (parsers.check_trail / "")
        * parsers.link_label
        * parsers.colon
        * parsers.spnlc * parsers.url
        * ( parsers.spnlc_sep * parsers.title
          * (parsers.spnlc * Ct(parsers.attributes))
          * parsers.only_blank
          + parsers.spnlc_sep * parsers.title * parsers.only_blank
          + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
          * parsers.only_blank
          + Cc("") * parsers.only_blank)

      local ReferenceWithAttributes = define_reference_parser
                                    / self.register_link

      self.update_rule("Reference", ReferenceWithAttributes)

```

The following patterns define direct and indirect links with attributes.

```

      local LinkWithAttributesAndEmph
        = Ct(parsers.link_and_emph_table * Cg(Cc(true),
             "match_link_attributes"))
        / self.defer_link_and_emphasis_processing

      self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)

```

The following patterns define autolinks with attributes.

```
      local AutoLinkUrlWithAttributes
                      = parsers.auto_link_url
```

```
                      * Ct(parsers.attributes)
                      / self.auto_link_url

      self.insert_pattern("Inline before AutoLinkUrl",
                          AutoLinkUrlWithAttributes,
                          "AutoLinkUrlWithAttributes")

      local AutoLinkEmailWithAttributes
                      = parsers.auto_link_email
                      * Ct(parsers.attributes)
                      / self.auto_link_email

      self.insert_pattern("Inline before AutoLinkEmail",
                          AutoLinkEmailWithAttributes,
                          "AutoLinkEmailWithAttributes")

      if options.relativeReferences then

        local AutoLinkRelativeReferenceWithAttributes
                        = parsers.auto_link_relative_reference
                        * Ct(parsers.attributes)
                        / self.auto_link_url

        self.insert_pattern(
          "Inline before AutoLinkRelativeReference",
          AutoLinkRelativeReferenceWithAttributes,
          "AutoLinkRelativeReferenceWithAttributes")

      end

    end
  }
end
```

3.1.7.13 Notes

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

```
M.extensions.notes = function(notes, inline_notes)
  assert(notes or inline_notes)
  return {
    name = "built-in notes syntax extension",
    extend_writer = function(self)
```

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

```
11890       function self.note(s)
11891         if self.flatten_inlines then return "" end
11892         return {"\\markdownRendererNote{",s,"}"}
11893       end
11894     end, extend_reader = function(self)
11895       local parsers = self.parsers
11896       local writer = self.writer
11897
11898       local rawnotes = parsers.rawnotes
11899
11900       if inline_notes then
11901         local InlineNote
11902           = parsers.circumflex
11903           * ( parsers.link_label
11904             / self.parser_functions.parse_inlines_no_inline_note)
11905           / writer.note
11906
11907         self.insert_pattern("Inline after LinkAndEmph",
11908                             InlineNote, "InlineNote")
11909       end
11910       if notes then
11911         local function strip_first_char(s)
11912           return s:sub(2)
11913         end
11914
11915         local RawNoteRef
11916                       = #(parsers.lbracket * parsers.circumflex)
11917                       * parsers.link_label / strip_first_char
11918
11919         -- like indirect_link
11920         local function lookup_note(ref)
11921           return writer.defer_call(function()
11922             local found = rawnotes[self.normalize_tag(ref)]
11923             if found then
11924               return writer.note(
11925                 self.parser_functions.parse_blocks_nested(found))
11926             else
11927               return {"[",
11928                 self.parser_functions.parse_inlines("^" .. ref), "]"}
11929             end
11930           end)
11931         end
11932
11933         local function register_note(ref,rawnote)
11934           local normalized_tag = self.normalize_tag(ref)
```

```
11935           if rawnotes[normalized_tag] == nil then
11936             rawnotes[normalized_tag] = rawnote
11937           end
11938           return ""
11939         end
11940
11941         local NoteRef = RawNoteRef / lookup_note
11942
11943         local optionally_indented_line
11944           = parsers.check_optional_indent_and_any_trail * parsers.line
11945
11946         local blank
11947           = parsers.check_optional_blank_indent_and_any_trail
11948           * parsers.optionalspace * parsers.newline
11949
11950         local chunk
11951           = Cs(parsers.line
11952           * (optionally_indented_line - blank)^0)
11953
11954         local indented_blocks = function(bl)
11955           return Cs( bl
11956                 * ( blank^1 * (parsers.check_optional_indent / "")
11957                   * parsers.check_code_trail
11958                   * -parsers.blankline * bl)^0)
11959         end
11960
11961         local NoteBlock
11962                     = parsers.check_trail_no_rem
11963                     * RawNoteRef * parsers.colon
11964                     * parsers.spnlc * indented_blocks(chunk)
11965                     / register_note
11966
11967         self.update_rule("Reference", function(previous_pattern)
11968           if previous_pattern == nil then
11969             previous_pattern = parsers.Reference
11970           end
11971           return NoteBlock + previous_pattern
11972         end)
11973
11974         self.insert_pattern("Inline before LinkAndEmph",
11975                             NoteRef, "NoteRef")
11976       end
11977
11978       self.add_special_character("^")
11979     end
11980   }
11981 end
```

3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```
11982 M.extensions.pipe_tables = function(table_captions, table_attributes)
11983
11984   local function make_pipe_table_rectangular(rows)
11985     local num_columns = #rows[2]
11986     local rectangular_rows = {}
11987     for i = 1, #rows do
11988       local row = rows[i]
11989       local rectangular_row = {}
11990       for j = 1, num_columns do
11991         rectangular_row[j] = row[j] or ""
11992       end
11993       table.insert(rectangular_rows, rectangular_row)
11994     end
11995     return rectangular_rows
11996   end
11997
11998   local function pipe_table_row(allow_empty_first_column
11999                                 , nonempty_column
12000                                 , column_separator
12001                                 , column)
12002     local row_beginning
12003     if allow_empty_first_column then
12004       row_beginning = -- empty first column
12005                       #(parsers.spacechar^4
12006                         * column_separator)
12007                     * parsers.optionalspace
12008                     * column
12009                     * parsers.optionalspace
12010                     -- non-empty first column
12011                     + parsers.nonindentspace
12012                     * nonempty_column^-1
12013                     * parsers.optionalspace
12014     else
12015       row_beginning = parsers.nonindentspace
12016                     * nonempty_column^-1
12017                     * parsers.optionalspace
12018     end
12019
12020     return Ct(row_beginning
```

```
            * (-- single column with no leading pipes
               #(column_separator
                 * parsers.optionalspace
                 * parsers.newline)
              * column_separator
              * parsers.optionalspace
              -- single column with leading pipes or
              -- more than a single column
              + (column_separator
                * parsers.optionalspace
                * column
                * parsers.optionalspace)^1
              * (column_separator
                * parsers.optionalspace)^-1))
end

return {
  name = "built-in pipe_tables syntax extension",
  extend_writer = function(self)
```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```
    function self.table(rows, caption, attributes)
      if not self.is_writing then return "" end
      local buffer = {}
      if attributes ~= nil then
        table.insert(buffer,
                     "\\markdownRendererTableAttributeContextBegin\n")
        table.insert(buffer, self.attributes(attributes))
      end
      table.insert(buffer,
                   {"\\markdownRendererTable{",
                    caption or "", "}{", #rows - 1, "}{",
                    #rows[1], "}"})
      local temp = rows[2] -- put alignments on the first row
      rows[2] = rows[1]
      rows[1] = temp
      for i, row in ipairs(rows) do
        table.insert(buffer, "{")
        for _, column in ipairs(row) do
          if i > 1 then -- do not use braces for alignments
            table.insert(buffer, "{")
          end
          table.insert(buffer, column)
          if i > 1 then
            table.insert(buffer, "}")
          end
```

```
      end
      table.insert(buffer, "}")
    end
    if attributes ~= nil then
      table.insert(buffer,
                   "\\markdownRendererTableAttributeContextEnd{}")
    end
    return buffer
  end
end, extend_reader = function(self)
  local parsers = self.parsers
  local writer = self.writer

  local table_hline_separator = parsers.pipe + parsers.plus

  local table_hline_column = (parsers.dash
                              - #(parsers.dash
                                 * (parsers.spacechar
                                   + table_hline_separator
                                   + parsers.newline)))^1
                           * (parsers.colon * Cc("r")
                             + parsers.dash * Cc("d"))
                           + parsers.colon
                           * (parsers.dash
                             - #(parsers.dash
                                * (parsers.spacechar
                                  + table_hline_separator
                                  + parsers.newline)))^1
                           * (parsers.colon * Cc("c")
                             + parsers.dash * Cc("l"))

  local table_hline = pipe_table_row(false
                                     , table_hline_column
                                     , table_hline_separator
                                     , table_hline_column)

  local table_caption_beginning
    = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
      * parsers.optionalspace * parsers.newline)^0
    * parsers.check_minimal_indent_and_trail
    * (P("Table")^-1 * parsers.colon)
    * parsers.optionalspace

  local function strip_trailing_spaces(s)
    return s:gsub("%s*$","")
  end
```

```
12112       local table_row
12113         = pipe_table_row(true
12114                         , (C((parsers.linechar - parsers.pipe)^1)
12115                           / strip_trailing_spaces
12116                           / self.parser_functions.parse_inlines)
12117                         , parsers.pipe
12118                         , (C((parsers.linechar - parsers.pipe)^0)
12119                           / strip_trailing_spaces
12120                           / self.parser_functions.parse_inlines))
12121
12122       local table_caption
12123       if table_captions then
12124         table_caption = #table_caption_beginning
12125                       * table_caption_beginning
12126         if table_attributes then
12127           table_caption = table_caption
12128                         * (C(((( parsers.linechar
12129                                - (parsers.attributes
12130                                  * parsers.optionalspace
12131                                  * parsers.newline
12132                                  * -#( parsers.optionalspace
12133                                      * parsers.linechar)))
12134                               + ( parsers.newline
12135                                 * #( parsers.optionalspace
12136                                    * parsers.linechar)
12137                                 * C(parsers.optionalspace)
12138                                 / writer.space))
12139                              * (parsers.linechar
12140                                - parsers.lbrace)^0)^1)
12141                             / self.parser_functions.parse_inlines)
12142                         * (parsers.newline
12143                           + ( Ct(parsers.attributes)
12144                             * parsers.optionalspace
12145                             * parsers.newline))
12146         else
12147           table_caption = table_caption
12148                         * C(( parsers.linechar
12149                             + ( parsers.newline
12150                               * #( parsers.optionalspace
12151                                  * parsers.linechar)
12152                               * C(parsers.optionalspace)
12153                               / writer.space))^1)
12154                         / self.parser_functions.parse_inlines
12155                         * parsers.newline
12156         end
12157       else
12158         table_caption = parsers.fail
```

```
      end

      local PipeTable
        = Ct( table_row * parsers.newline
            * (parsers.check_minimal_indent_and_trail / {})
          * table_hline * parsers.newline
          * ( (parsers.check_minimal_indent / {})
            * table_row * parsers.newline)^0)
        / make_pipe_table_rectangular
        * table_caption^-1
        / writer.table

      self.insert_pattern("Block after Blockquote",
                          PipeTable, "PipeTable")
    end
  }
end
```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
M.extensions.raw_inline = function()
  return {
    name = "built-in raw_inline syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.rawInline(s, attr)
        if not self.is_writing then return "" end
        if self.flatten_inlines then return s end
        local name = util.cache_verbatim(options.cacheDir, s)
        return {"\\markdownRendererInputRawInline{",
                name,"}{", self.string(attr),"}"}
      end
    end, extend_reader = function(self)
      local writer = self.writer

      local RawInline = parsers.inticks
                      * parsers.raw_attribute
                      / writer.rawInline

      self.insert_pattern("Inline before Code",
                          RawInline, "RawInline")
    end
```

```
12199   }
12200 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
12201 M.extensions.strike_through = function()
12202   return {
12203     name = "built-in strike_through syntax extension",
12204     extend_writer = function(self)
```

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

```
12205       function self.strike_through(s)
12206         if self.flatten_inlines then return s end
12207         return {"\\markdownRendererStrikeThrough{",s,"}"}
12208       end
12209     end, extend_reader = function(self)
12210       local parsers = self.parsers
12211       local writer = self.writer
12212
12213       local StrikeThrough = (
12214         parsers.between(parsers.Inline, parsers.doubletildes,
12215                         parsers.doubletildes)
12216       ) / writer.strike_through
12217
12218       self.insert_pattern("Inline after LinkAndEmph",
12219                           StrikeThrough, "StrikeThrough")
12220
12221       self.add_special_character("~")
12222     end
12223   }
12224 end
```

3.1.7.17 Subscripts

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

```
12225 M.extensions.subscripts = function()
12226   return {
12227     name = "built-in subscripts syntax extension",
12228     extend_writer = function(self)
```

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

```
12229       function self.subscript(s)
12230         if self.flatten_inlines then return s end
```

```
12231         return {"\\markdownRendererSubscript{",s,"}"}
12232       end
12233     end, extend_reader = function(self)
12234       local parsers = self.parsers
12235       local writer = self.writer
12236
12237       local Subscript = (
12238         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12239       ) / writer.subscript
12240
12241       self.insert_pattern("Inline after LinkAndEmph",
12242                           Subscript, "Subscript")
12243
12244       self.add_special_character("~")
12245     end
12246   }
12247 end
```

3.1.7.18 Superscripts

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

```
12248 M.extensions.superscripts = function()
12249   return {
12250     name = "built-in superscripts syntax extension",
12251     extend_writer = function(self)
```

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

```
12252       function self.superscript(s)
12253         if self.flatten_inlines then return s end
12254         return {"\\markdownRendererSuperscript{",s,"}"}
12255       end
12256     end, extend_reader = function(self)
12257       local parsers = self.parsers
12258       local writer = self.writer
12259
12260       local Superscript = (
12261         parsers.between(parsers.Str, parsers.circumflex,
12262                         parsers.circumflex)
12263       ) / writer.superscript
12264
12265       self.insert_pattern("Inline after LinkAndEmph",
12266                           Superscript, "Superscript")
12267
12268       self.add_special_character("^")
12269     end
12270   }
```

```
12271 end
```

3.1.7.19 TEX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12272 M.extensions.tex_math = function(tex_math_dollars,
12273                                   tex_math_single_backslash,
12274                                   tex_math_double_backslash)
12275   return {
12276     name = "built-in tex_math syntax extension",
12277     extend_writer = function(self)
```

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

```
12278       function self.display_math(s)
12279         if self.flatten_inlines then return s end
12280         return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
12281       end
```

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

```
12282       function self.inline_math(s)
12283         if self.flatten_inlines then return s end
12284         return {"\\markdownRendererInlineMath{",self.math(s),"}"}
12285       end
12286     end, extend_reader = function(self)
12287       local parsers = self.parsers
12288       local writer = self.writer
12289
12290       local function between(p, starter, ender)
12291         return (starter * Cs(p * (p - ender)^0) * ender)
12292       end
12293
12294       local function strip_preceding_whitespaces(str)
12295         return str:gsub("^%s*(.-)$", "%1")
12296       end
12297
12298       local allowed_before_closing
12299         = B( parsers.backslash * parsers.any
12300            + parsers.any * (parsers.any - parsers.backslash))
12301
12302       local allowed_before_closing_no_space
12303         = B( parsers.backslash * parsers.any
12304            + parsers.any * (parsers.nonspacechar - parsers.backslash))
12305
```

The following patterns implement the Pandoc dollar math syntax extension.

```
12306       local dollar_math_content
12307         = (parsers.newline * (parsers.check_optional_indent / "")
12308         + parsers.backslash^-1
12309         * parsers.linechar)
12310         - parsers.blankline^2
12311         - parsers.dollar
12312
12313       local inline_math_opening_dollars = parsers.dollar
12314                                         * #(parsers.nonspacechar)
12315
12316       local inline_math_closing_dollars
12317         = allowed_before_closing_no_space
12318         * parsers.dollar
12319         * -#(parsers.digit)
12320
12321       local inline_math_dollars = between(Cs( dollar_math_content),
12322                                           inline_math_opening_dollars,
12323                                           inline_math_closing_dollars)
12324
12325       local display_math_opening_dollars  = parsers.dollar
12326                                           * parsers.dollar
12327
12328       local display_math_closing_dollars  = parsers.dollar
12329                                           * parsers.dollar
12330
12331       local display_math_dollars = between(Cs( dollar_math_content),
12332                                            display_math_opening_dollars,
12333                                            display_math_closing_dollars)
```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```
12334       local backslash_math_content
12335         = (parsers.newline * (parsers.check_optional_indent / "")
12336         + parsers.linechar)
12337         - parsers.blankline^2
```

The following patterns implement the Pandoc double backslash math syntax extension.

```
12338       local inline_math_opening_double  = parsers.backslash
12339                                         * parsers.backslash
12340                                         * parsers.lparent
12341
12342       local inline_math_closing_double  = allowed_before_closing
12343                                         * parsers.spacechar^0
12344                                         * parsers.backslash
12345                                         * parsers.backslash
12346                                         * parsers.rparent
12347
```

```
12348       local inline_math_double  = between(Cs( backslash_math_content),
12349                                           inline_math_opening_double,
12350                                           inline_math_closing_double)
12351                                 / strip_preceding_whitespaces
12352
12353       local display_math_opening_double = parsers.backslash
12354                                         * parsers.backslash
12355                                         * parsers.lbracket
12356
12357       local display_math_closing_double = allowed_before_closing
12358                                         * parsers.spacechar^0
12359                                         * parsers.backslash
12360                                         * parsers.backslash
12361                                         * parsers.rbracket
12362
12363       local display_math_double = between(Cs( backslash_math_content),
12364                                           display_math_opening_double,
12365                                           display_math_closing_double)
12366                                 / strip_preceding_whitespaces
```

The following patterns implement the Pandoc single backslash math syntax extension.

```
12367       local inline_math_opening_single  = parsers.backslash
12368                                         * parsers.lparent
12369
12370       local inline_math_closing_single  = allowed_before_closing
12371                                         * parsers.spacechar^0
12372                                         * parsers.backslash
12373                                         * parsers.rparent
12374
12375       local inline_math_single  = between(Cs( backslash_math_content),
12376                                           inline_math_opening_single,
12377                                           inline_math_closing_single)
12378                                 / strip_preceding_whitespaces
12379
12380       local display_math_opening_single = parsers.backslash
12381                                         * parsers.lbracket
12382
12383       local display_math_closing_single = allowed_before_closing
12384                                         * parsers.spacechar^0
12385                                         * parsers.backslash
12386                                         * parsers.rbracket
12387
12388       local display_math_single = between(Cs( backslash_math_content),
12389                                           display_math_opening_single,
12390                                           display_math_closing_single)
12391                                 / strip_preceding_whitespaces
12392
12393       local display_math = parsers.fail
```

```
12394
12395       local inline_math = parsers.fail
12396
12397       if tex_math_dollars then
12398         display_math = display_math + display_math_dollars
12399         inline_math = inline_math + inline_math_dollars
12400       end
12401
12402       if tex_math_double_backslash then
12403         display_math = display_math + display_math_double
12404         inline_math = inline_math + inline_math_double
12405       end
12406
12407       if tex_math_single_backslash then
12408         display_math = display_math + display_math_single
12409         inline_math = inline_math + inline_math_single
12410       end
12411
12412       local TexMath = display_math / writer.display_math
12413                     + inline_math / writer.inline_math
12414
12415       self.insert_pattern("Inline after LinkAndEmph",
12416                           TexMath, "TexMath")
12417
12418       if tex_math_dollars then
12419         self.add_special_character("$")
12420       end
12421
12422       if tex_math_single_backslash or tex_math_double_backslash then
12423         self.add_special_character("\\")
12424         self.add_special_character("[")
12425         self.add_special_character("]")
12426         self.add_special_character(")")
12427         self.add_special_character("(")
12428       end
12429     end
12430   }
12431 end
```

3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. When both `expect_jekyll_data` and

`ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```
M.extensions.jekyll_data = function(expect_jekyll_data,
                                    ensure_jekyll_data)
  return {
    name = "built-in jekyll_data syntax extension",
    extend_writer = function(self)
```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```
      function self.jekyllData(d, t, p)
        if not self.is_writing then return "" end

        local buf = {}

        local keys = {}
        for k, _ in pairs(d) do
          table.insert(keys, k)
        end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
        table.sort(keys, function(first, second)
          if type(first) ~= type(second) then
            return type(first) < type(second)
          else
            return first < second
          end
        end)

        if not p then
          table.insert(buf, "\\markdownRendererJekyllDataBegin")
        end

        local is_sequence = false
        if #d > 0 and #d == #keys then
          for i=1, #d do
            if d[i] == nil then
              goto not_a_sequence
            end
          end
          is_sequence = true
        end
```

```
        ::not_a_sequence::

        if is_sequence then
          table.insert(buf,
            "\\markdownRendererJekyllDataSequenceBegin{")
          table.insert(buf, self.identifier(p or "null"))
          table.insert(buf, "}{")
          table.insert(buf, #keys)
          table.insert(buf, "}")
        else
          table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
          table.insert(buf, self.identifier(p or "null"))
          table.insert(buf, "}{")
          table.insert(buf, #keys)
          table.insert(buf, "}")
        end

        for _, k in ipairs(keys) do
          local v = d[k]
          local typ = type(v)
          k = tostring(k or "null")
          if typ == "table" and next(v) ~= nil then
            table.insert(
              buf,
              self.jekyllData(v, t, k)
            )
          else
            k = self.identifier(k)
            v = tostring(v)
            if typ == "boolean" then
              table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              table.insert(buf, v)
              table.insert(buf, "}")
            elseif typ == "number" then
              table.insert(buf, "\\markdownRendererJekyllDataNumber{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              table.insert(buf, v)
              table.insert(buf, "}")
            elseif typ == "string" then
              table.insert(buf,
                "\\markdownRendererJekyllDataProgrammaticString{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              table.insert(buf, self.identifier(v))
```

```
12514             table.insert(buf, "}")
12515             table.insert(buf,
12516               "\\markdownRendererJekyllDataTypographicString{")
12517             table.insert(buf, k)
12518             table.insert(buf, "}{")
12519             table.insert(buf, t(v))
12520             table.insert(buf, "}")
12521           elseif typ == "table" then
12522             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12523             table.insert(buf, k)
12524             table.insert(buf, "}")
12525           else
12526             local error = self.error(format(
12527               "Unexpected type %s for value of "
12528               .. "YAML key %s.", typ, k))
12529             table.insert(buf, error)
12530           end
12531         end
12532       end
12533
12534       if is_sequence then
12535         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12536       else
12537         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12538       end
12539
12540       if not p then
12541         table.insert(buf, "\\markdownRendererJekyllDataEnd")
12542       end
12543
12544       return buf
12545     end
12546   end, extend_reader = function(self)
12547     local parsers = self.parsers
12548     local writer = self.writer
12549
12550     local JekyllData
12551       = Cmt( C((parsers.line - P("---") - P("..."))^0)
12552            , function(s, i, text) -- luacheck: ignore s i
12553                local data
12554                local ran_ok, _ = pcall(function()
12555                  local tinyyaml = require("tinyyaml")
12556                  data = tinyyaml.parse(text, {timestamps=false})
12557                end)
12558                if ran_ok and data ~= nil then
12559                  return true, writer.jekyllData(data, function(s)
12560                    return self.parser_functions.parse_blocks_nested(s)
```

```
12561                   end, nil)
12562                 else
12563                   return false
12564                 end
12565               end
12566             )
12567
12568       local UnexpectedJekyllData
12569         = P("---")
12570         * parsers.blankline / 0
12571         -- if followed by blank, it's thematic break
12572         * #(-parsers.blankline)
12573         * JekyllData
12574         * (P("---") + P("..."))
12575
12576       local ExpectedJekyllData
12577         = ( P("---")
12578           * parsers.blankline / 0
12579           -- if followed by blank, it's thematic break
12580           * #(-parsers.blankline)
12581           )^-1
12582         * JekyllData
12583         * (P("---") + P("..."))^-1
12584
12585       if ensure_jekyll_data then
12586         ExpectedJekyllData = ExpectedJekyllData
12587                            * parsers.eof
12588       else
12589         ExpectedJekyllData = ( ExpectedJekyllData
12590                              * (V("Blank")^0 / writer.interblocksep)
12591                              )^-1
12592       end
12593
12594       self.insert_pattern("Block before Blockquote",
12595                           UnexpectedJekyllData, "UnexpectedJekyllData")
12596       if expect_jekyll_data then
12597         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12598       end
12599     end
12600   }
12601 end
```

3.1.8 Conversion from Markdown to Plain TEX

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its own function `new` unless option `eagerCache` or `finalizeCache` has been enabled

and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```
function M.new(options)
```

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

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

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```
  local parser_convert = nil
  return function(input)
    local function convert(input)
      if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
        local parser = require("markdown-parser")
        if metadata.version ~= parser.metadata.version then
          warn("markdown.lua " .. metadata.version .. " used with " ..
               "markdown-parser.lua " .. parser.metadata.version .. ".")
        end
        parser_convert = parser.new(options)
      end
      return parser_convert(input)
    end
```

If we cache markdown documents, produce the cache file and transform its filename to plain TEX output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```
    local output
    if options.eagerCache or options.finalizeCache then
      local salt = util.salt(options)
      local name = util.cache(options.cacheDir, input, salt, convert,
                              ".md.tex")
      output = [[\input{]] .. name .. [[}\relax]]
```

Otherwise, return the result of the conversion directly.

```
    else
      output = convert(input)
    end
```

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

```
12628     if options.finalizeCache then
12629       local file, mode
12630       if options.frozenCacheCounter > 0 then
12631         mode = "a"
12632       else
12633         mode = "w"
12634       end
12635       file = assert(io.open(options.frozenCacheFileName, mode),
12636         [[Could not open file "]] .. options.frozenCacheFileName
12637         .. [[" for writing]])
12638       assert(file:write(
12639         [[\expandafter\global\expandafter\def\csname ]]
12640         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12641         .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
12642       assert(file:close())
12643     end
12644     return output
12645   end
12646 end
```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain TEX output. See Section 2.1.1.

```
12647 function M.new(options)
```

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

```
12648   options = options or {}
12649   setmetatable(options, { __index = function (_, key)
12650     return defaultOptions[key] end })
```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
12651   if options.singletonCache and singletonCache.convert then
12652     for k, v in pairs(defaultOptions) do
12653       if type(v) == "table" then
12654         for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12655           if singletonCache.options[k][i] ~= options[k][i] then
12656             goto miss
12657           end
12658         end
```

The `cacheDir` option is disregarded.

```
12659       elseif k ~= "cacheDir"
12660          and singletonCache.options[k] ~= options[k] then
12661         goto miss
12662       end
12663     end
12664     return singletonCache.convert
12665   end
12666   ::miss::
```

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

```
12667  local extensions = {}
12668
12669  if options.bracketedSpans then
12670    local bracketed_spans_extension = M.extensions.bracketed_spans()
12671    table.insert(extensions, bracketed_spans_extension)
12672  end
12673
12674  if options.contentBlocks then
12675    local content_blocks_extension = M.extensions.content_blocks(
12676      options.contentBlocksLanguageMap)
12677    table.insert(extensions, content_blocks_extension)
12678  end
12679
12680  if options.definitionLists then
12681    local definition_lists_extension = M.extensions.definition_lists(
12682      options.tightLists)
12683    table.insert(extensions, definition_lists_extension)
12684  end
12685
12686  if options.fencedCode then
12687    local fenced_code_extension = M.extensions.fenced_code(
12688      options.blankBeforeCodeFence,
12689      options.fencedCodeAttributes,
12690      options.rawAttribute)
12691    table.insert(extensions, fenced_code_extension)
12692  end
12693
12694  if options.fencedDivs then
12695    local fenced_div_extension = M.extensions.fenced_divs(
12696      options.blankBeforeDivFence)
12697    table.insert(extensions, fenced_div_extension)
12698  end
12699
12700  if options.headerAttributes then
12701    local header_attributes_extension = M.extensions.header_attributes()
12702    table.insert(extensions, header_attributes_extension)
12703  end
12704
12705  if options.inlineCodeAttributes then
12706    local inline_code_attributes_extension =
12707      M.extensions.inline_code_attributes()
12708    table.insert(extensions, inline_code_attributes_extension)
12709  end
12710
12711  if options.jekyllData then
12712    local jekyll_data_extension = M.extensions.jekyll_data(
```

```
12713       options.expectJekyllData, options.ensureJekyllData)
12714     table.insert(extensions, jekyll_data_extension)
12715   end
12716
12717   if options.linkAttributes then
12718     local link_attributes_extension =
12719       M.extensions.link_attributes()
12720     table.insert(extensions, link_attributes_extension)
12721   end
12722
12723   if options.lineBlocks then
12724     local line_block_extension = M.extensions.line_blocks()
12725     table.insert(extensions, line_block_extension)
12726   end
12727
12728   if options.mark then
12729     local mark_extension = M.extensions.mark()
12730     table.insert(extensions, mark_extension)
12731   end
12732
12733   if options.pipeTables then
12734     local pipe_tables_extension = M.extensions.pipe_tables(
12735       options.tableCaptions, options.tableAttributes)
12736     table.insert(extensions, pipe_tables_extension)
12737   end
12738
12739   if options.rawAttribute then
12740     local raw_inline_extension = M.extensions.raw_inline()
12741     table.insert(extensions, raw_inline_extension)
12742   end
12743
12744   if options.strikeThrough then
12745     local strike_through_extension = M.extensions.strike_through()
12746     table.insert(extensions, strike_through_extension)
12747   end
12748
12749   if options.subscripts then
12750     local subscript_extension = M.extensions.subscripts()
12751     table.insert(extensions, subscript_extension)
12752   end
12753
12754   if options.superscripts then
12755     local superscript_extension = M.extensions.superscripts()
12756     table.insert(extensions, superscript_extension)
12757   end
12758
12759   if options.texMathDollars or
```

```
12760       options.texMathSingleBackslash or
12761       options.texMathDoubleBackslash then
12762     local tex_math_extension = M.extensions.tex_math(
12763       options.texMathDollars,
12764       options.texMathSingleBackslash,
12765       options.texMathDoubleBackslash)
12766     table.insert(extensions, tex_math_extension)
12767   end
12768
12769   if options.notes or options.inlineNotes then
12770     local notes_extension = M.extensions.notes(
12771       options.notes, options.inlineNotes)
12772     table.insert(extensions, notes_extension)
12773   end
12774
12775   if options.citations then
12776     local citations_extension
12777       = M.extensions.citations(options.citationNbsps)
12778     table.insert(extensions, citations_extension)
12779   end
12780
12781   if options.fancyLists then
12782     local fancy_lists_extension = M.extensions.fancy_lists()
12783     table.insert(extensions, fancy_lists_extension)
12784   end
```

Apply user-defined syntax extensions based on `options.extensions`.

```
12785   for _, user_extension_filename in ipairs(options.extensions) do
12786     local user_extension = (function(filename)
```

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

```
12787       local pathname = assert(kpse.find_file(filename),
12788         [[Could not locate user-defined syntax extension "]]
12789         .. filename)
12790       local input_file = assert(io.open(pathname, "r"),
12791         [[Could not open user-defined syntax extension "]]
12792         .. pathname .. [[" for reading]])
12793       local input = assert(input_file:read("*a"))
12794       assert(input_file:close())
12795       local user_extension, err = load([[
12796         local sandbox = {}
12797         setmetatable(sandbox, {__index = _G})
12798         _ENV = sandbox
12799       ]] .. input)()
12800       assert(user_extension,
12801         [[Failed to compile user-defined syntax extension "]]
12802         .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
12803       assert(user_extension.api_version ~= nil,
12804         [[User-defined syntax extension "]] .. pathname
12805         .. [[" does not specify mandatory field "api_version"]])
12806       assert(type(user_extension.api_version) == "number",
12807         [[User-defined syntax extension "]] .. pathname
12808         .. [[" specifies field "api_version" of type "]]
12809         .. type(user_extension.api_version)
12810         .. [[" but "number" was expected]])
12811       assert(user_extension.api_version > 0
12812            and user_extension.api_version
12813             <= metadata.user_extension_api_version,
12814         [[User-defined syntax extension "]] .. pathname
12815         .. [[" uses syntax extension API version "]]
12816         .. user_extension.api_version .. [[ but markdown.lua ]]
12817         .. metadata.version .. [[ uses API version ]]
12818         .. metadata.user_extension_api_version
12819         .. [[, which is incompatible]])
12820
12821       assert(user_extension.grammar_version ~= nil,
12822         [[User-defined syntax extension "]] .. pathname
12823         .. [[" does not specify mandatory field "grammar_version"]])
12824       assert(type(user_extension.grammar_version) == "number",
12825         [[User-defined syntax extension "]] .. pathname
12826         .. [[" specifies field "grammar_version" of type "]]
12827         .. type(user_extension.grammar_version)
12828         .. [[" but "number" was expected]])
12829       assert(user_extension.grammar_version == metadata.grammar_version,
12830         [[User-defined syntax extension "]] .. pathname
12831         .. [[" uses grammar version "]]
12832         .. user_extension.grammar_version
12833         .. [[ but markdown.lua ]] .. metadata.version
12834         .. [[ uses grammar version ]] .. metadata.grammar_version
12835         .. [[, which is incompatible]])
12836
12837       assert(user_extension.finalize_grammar ~= nil,
12838         [[User-defined syntax extension "]] .. pathname
12839         .. [[" does not specify mandatory "finalize_grammar" field]])
12840       assert(type(user_extension.finalize_grammar) == "function",
12841         [[User-defined syntax extension "]] .. pathname
12842         .. [[" specifies field "finalize_grammar" of type "]]
12843         .. type(user_extension.finalize_grammar)
12844         .. [[" but "function" was expected]])
```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```
12845       local extension = {
```

```
        name = [[user-defined "]] .. pathname .. [[" syntax extension]],
        extend_reader = user_extension.finalize_grammar,
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

Produce a conversion function from markdown to plain TEX.

```
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  local convert = reader.finalize_grammar(extensions)
```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```
  collectgarbage("collect")
```

Update the singleton cache.

```
  if options.singletonCache then
    local singletonCacheOptions = {}
    for k, v in pairs(options) do
      singletonCacheOptions[k] = v
    end
    setmetatable(singletonCacheOptions,
      { __index = function (_, key)
        return defaultOptions[key] end })
    singletonCache.options = singletonCacheOptions
    singletonCache.convert = convert
  end
```

Return the conversion function from markdown to plain TEX.

```
  return convert
end

return M
```

3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

local input
if input_filename then
  local input_file = assert(io.open(input_filename, "r"),
    [[Could not open file "]] .. input_filename .. [[" for reading]])
  input = assert(input_file:read("*a"))
  assert(input_file:close())
else
```

```
  input = assert(io.read("*a"))
end

```

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

```
local lfs = require("lfs")
if options.cacheDir and not lfs.isdir(options.cacheDir) then
  assert(lfs.mkdir(options["cacheDir"]))
end
```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it.

```
local kpse
(function()
  local should_initialize = package.loaded.kpse == nil
                         or tex.initialize ~= nil
  kpse = require("kpse")
  if should_initialize then
    kpse.set_program_name("luatex")
  end
end)()
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.

```
if metadata.version ~= md.metadata.version then
  warn("markdown-cli.lua " .. metadata.version .. " used with " ..
       "markdown.lua " .. md.metadata.version .. ".")
end
local convert = md.new(options)
local output = convert(input)

if output_filename then
  local output_file = assert(io.open(output_filename, "w"),
    [[Could not open file "]] .. output_filename .. [[" for writing]])
  assert(output_file:write(output))
  assert(output_file:close())
else
  assert(io.write(output))
end
```

Remove the `options.cacheDir` directory if it is empty.

```
if options.cacheDir then
  lfs.rmdir(options.cacheDir)
end
```

3.2 Plain TEX Implementation

The plain TEX implementation provides macros for the interfacing between TEX 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 TEX exposed by the plain TEX interface (see Section 2.2).

3.2.1 Logging Facilities

```
\ExplSyntaxOn
\cs_if_free:NT
  \markdownInfo
  {
    \cs_new:Npn
      \markdownInfo #1
      {
        \msg_info:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownWarning
  {
    \cs_new:Npn
      \markdownWarning #1
      {
        \msg_warning:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownError
  {
    \cs_new:Npn
      \markdownError #1 #2
      {
        \msg_error:nnee
          { markdown }
          { generic-message-with-help-text }
          { #1 }
          { #2 }
      }
  }
```

```
\msg_new:nnn
  { markdown }
  { generic-message }
  { #1 }
\msg_new:nnnn
  { markdown }
  { generic-message-with-help-text }
  { #1 }
  { #2 }
\cs_generate_variant:Nn
  \msg_info:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_warning:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnee }
\ExplSyntaxOff
```

3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain TEX themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
\prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
\cs_new:Nn
  \@@_plain_tex_load_theme:nnn
  {
    \prop_get:NnNTF
      \g_@@_plain_tex_loaded_themes_linenos_prop
      { #1 }
      \l_tmpa_tl
      {
        \prop_get:NnN
          \g_@@_plain_tex_loaded_themes_versions_prop
          { #1 }
          \l_tmpb_tl
        \str_if_eq:nVTF
          { #2 }
          \l_tmpb_tl
          {
            \msg_warning:nnnVn
              { markdown }
              { repeatedly-loaded-plain-tex-theme }
```

```
            { #1 }
            \l_tmpa_tl
            { #2 }
        }
        {
          \msg_error:nnnnVV
            { markdown }
            { different-versions-of-plain-tex-theme }
            { #1 }
            { #2 }
            \l_tmpb_tl
            \l_tmpa_tl
        }
    }
    {
      \prop_gput:Nnx
        \g_@@_plain_tex_loaded_themes_linenos_prop
        { #1 }
        { \tex_the:D \tex_inputlineno:D }  % noqa: W200
      \prop_gput:Nnn
        \g_@@_plain_tex_loaded_themes_versions_prop
        { #1 }
        { #2 }
```

Load built-in plain TEX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```
      \prop_if_in:NnTF
        \g_@@_plain_tex_built_in_themes_prop
        { #1 }
        {
          \msg_info:nnnn
            { markdown }
            { loading-built-in-plain-tex-theme }
            { #1 }
            { #2 }
          \prop_item:Nn
            \g_@@_plain_tex_built_in_themes_prop
            { #1 }
        }
        {
          \msg_info:nnnn
            { markdown }
            { loading-plain-tex-theme }
            { #1 }
            { #2 }
          \file_input:n
            { markdown theme #3 }
```

```
          }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-plain-tex-theme }
  { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { loading-built-in-plain-tex-theme }
  { Loading~version~#2~of~built-in~plain~TeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-plain-tex-theme }
  {
    Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\msg_new:nnn
  { markdown }
  { different-versions-of-plain-tex-theme }
  {
    Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\cs_generate_variant:Nn
  \prop_gput:Nnn
  { Nnx }
\cs_gset_eq:NN
  \@@_load_theme:nnn
  \@@_plain_tex_load_theme:nnn
\cs_generate_variant:Nn
  \@@_load_theme:nnn
  { VeV }
\cs_generate_variant:Nn
  \msg_error:nnnnnn
  { nnnnVV }
\cs_generate_variant:Nn
  \msg_warning:nnnnn
  { nnnVn }
```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as LaTeX and ConTeXt.

```
\cs_new:Npn
  \markdownLoadPlainTeXTheme
  {
```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_reverse:N
      \l_tmpa_tl
    \tl_set:Ne
      \l_tmpb_tl
      {
        \tl_tail:V
          \l_tmpa_tl
      }
    \tl_reverse:N
      \l_tmpb_tl
```

Next, we munge the theme name.

```
    \str_set:NV
      \l_tmpa_str
      \l_tmpb_tl
    \str_replace_all:Nnn
      \l_tmpa_str
      { / }
      { _ }
```

Finally, we load the plain TEX theme.

```
    \@@_plain_tex_load_theme:VeV
      \l_tmpb_tl
      { \markdownThemeVersion }
      \l_tmpa_str
  }
\cs_generate_variant:Nn
  \tl_set:Nn
  { Ne }
\cs_generate_variant:Nn
  \@@_plain_tex_load_theme:nnn
  { VeV }
```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```
\prop_gput:Nnn
  \g_@@_plain_tex_built_in_themes_prop
  { witiko / dot }
  {
    \str_if_eq:enF
      { \markdownThemeVersion }
      { silent }
      {
```

```
      \markdownWarning
        {
          The~theme~name~"witiko/dot"~has~been~soft-deprecated.
          \iow_newline:
          Consider~changing~the~name~to~"witiko/diagrams@v1".
        }
    }
```

We enable the `fencedCode` Lua option.

```
  \markdownSetup { fencedCode }
```

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

```
  \cs_set_eq:NN
    \@@_dot_previous_definition:nnn
    \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot …`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain TEX option is disabled and the code block has not been previously typeset:

```
  \regex_const:Nn
    \c_@@_dot_infostring_regex
    { ^dot(\s+(.+))? }
  \seq_new:N
    \l_@@_dot_matches_seq
  \markdownSetup {
    rendererPrototypes = {
      inputFencedCode = {
        \regex_extract_once:NnNTF
          \c_@@_dot_infostring_regex
          { #2 }
          \l_@@_dot_matches_seq
          {
            \@@_if_option:nF
              { frozenCache }
              {
                \sys_shell_now:n
                  {
                    if~!~test~-e~#1.pdf.source~
                    ||~!~diff~#1~#1.pdf.source;
                    then~
                      dot~-Tpdf~-o~#1.pdf~#1;
                      cp~#1~#1.pdf.source;
                    fi
                  }
              }
```

We include the typeset image using the image token renderer:

```
            \exp_args:NNne
              \exp_last_unbraced:No
              \markdownRendererImage
                {
                  { Graphviz~image }
                  { #1.pdf }
                  { #1.pdf }
                }
                {
                  \seq_item:Nn
                    \l_@@_dot_matches_seq
                    { 3 }
                }
          }
```

If the infostring does not start with `dot …`, we use the previous definition of the fenced code token renderer prototype:

```
          {
            \@@_dot_previous_definition:nnn
              { #1 }
              { #2 }
              { #3 }
          }
        },
      },
    }
  }
```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```
\prop_gput:Nnn
  \g_@@_plain_tex_built_in_themes_prop
  { witiko / diagrams }
  {
    \str_case:enF
      { \markdownThemeVersion }
      {
        { latest }
          {
            \markdownWarning
              {
                Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
                theme~"witiko/diagrams".~This~will~keep~your~documents~
                from~suddenly~breaking~when~we~have~released~future~
                versions~of~the~theme~with~backwards-incompatible~
                syntax~and~behavior.
              }
            \markdownSetup
```

```
              {
                import = witiko/diagrams/v2,
              }
          }
        { v2 }
          {
            \markdownSetup
              {
                import = witiko/diagrams/v2,
              }
          }
        { v1 }
          {
            \markdownSetup
              {
                import = witiko/dot@silent,
              }
          }
      }
      {
        \msg_error:nnnen
          { markdown }
          { unknown-theme-version }
          { witiko/diagrams }
          { \markdownThemeVersion }
          { v1 }
      }
  }
\cs_generate_variant:Nn
  \msg_error:nnnnn
  { nnnen }
\msg_new:nnnn
  { markdown }
  { unknown-theme-version }
  { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
  { Known~versions~are:~#3 }
```

Next, we implement the theme witiko/diagrams/v2.

```
\prop_gput:Nnn
  \g_@@_plain_tex_built_in_themes_prop
  { witiko / diagrams / v2 }
  {
```

We enable the fencedCode and fencedCodeAttributes Lua option.

```
    \@@_setup:n
      {
        fencedCode = true,
        fencedCodeAttributes = true,
```

```
    }
```

Store the previous fenced code token renderer prototype.

```
    \cs_set_eq:NN
      \@@_diagrams_previous_fenced_code:nnn
      \markdownRendererInputFencedCodePrototype
```

Store the caption of the diagram.

```
    \tl_new:N
      \l_@@_diagrams_caption_tl
    \@@_setup:n
      {
        rendererPrototypes = {
```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```
          fencedCodeAttributeContextBegin = {
            \group_begin:
            \markdownRendererImageAttributeContextBegin
            \cs_set_eq:NN
              \@@_diagrams_previous_key_value:nn
              \markdownRendererAttributeKeyValuePrototype
            \@@_setup:n
              {
                rendererPrototypes = {
                  attributeKeyValue = {
                    \str_if_eq:nnTF
                      { ##1 }
                      { caption }
                      {
                        \tl_set:Nn
                          \l_@@_diagrams_caption_tl
                          { ##2 }
                      }
                      {
                        \@@_diagrams_previous_key_value:nn
                          { ##1 }
                          { ##2 }
                      }
                  },
                },
              }
          },
          fencedCodeAttributeContextEnd = {
            \markdownRendererImageAttributeContextEnd
            \group_end:
          },
        },
      }
```

```
\cs_new:Nn
  \@@_diagrams_render_diagram:nnnn
  {
    \@@_if_option:nF
      { frozenCache }
      {
        \sys_shell_now:n
          {
            if~!~test~-e~#2.source~
            ||~!~diff~#1~#2.source;
            then~
              (#3);
              cp~#1~#2.source;
            fi
          }
        \exp_args:NNnV
          \exp_last_unbraced:No
          \markdownRendererImage
            {
              { #4 }
              { #2 }
              { #2 }
            }
            \l_@@_diagrams_caption_tl
      }
  }
```

Use the prop `\g_markdown_diagrams_infostrings_prop` to determine how the code with a given infostring should be processed and routed to the token renderer prototype(s) for images.

```
\prop_new:N
  \g_markdown_diagrams_infostrings_prop
```

If we know a processing function for a given infostring, use it.

```
\@@_setup:n
  {
    rendererPrototypes = {
      inputFencedCode = {
        \prop_get:NnNTF
          \g_markdown_diagrams_infostrings_prop
          { #2 }
          \l_tmpa_tl
          {
            \cs_set:NV
              \@@_diagrams_infostrings_current:n
              \l_tmpa_tl
            \@@_diagrams_infostrings_current:n
              { #1 }
```

```
            }
```

Otherwise, use the previous fenced code token renderer prototype.

```
            {
              \@@_diagrams_previous_fenced_code:nnn
                { #1 }
                { #2 }
                { #3 }
            }
        },
      },
    }
  \cs_generate_variant:Nn
    \cs_set:Nn
    { NV }
```

Typeset fenced code with infostring `dot` using the command `dot` from the package Graphviz.

```
  \cs_set:Nn
    \@@_diagrams_infostrings_current:n
    {
      \@@_diagrams_render_diagram:nnnn
        { #1 }
        { #1.pdf }
        { dot~-Tpdf~-o~#1.pdf~#1 }
        { Graphviz~image }
    }
  \@@_tl_set_from_cs:NNn
    \l_tmpa_tl
    \@@_diagrams_infostrings_current:n
    { 1 }
  \prop_gput:NnV
    \g_markdown_diagrams_infostrings_prop
    { dot }
    \l_tmpa_tl
```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```
  \cs_set:Nn
    \@@_diagrams_infostrings_current:n
    {
      \@@_diagrams_render_diagram:nnnn
        { #1 }
        { #1.pdf }
        { mmdc~--pdfFit~-i~#1~-o~#1.pdf }
        { Mermaid~image }
    }
  \@@_tl_set_from_cs:NNn
```

```
      \l_tmpa_tl
      \@@_diagrams_infostrings_current:n
      { 1 }
    \prop_gput:NnV
      \g_markdown_diagrams_infostrings_prop
      { mermaid }
      \l_tmpa_tl
```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```
    \regex_const:Nn
      \c_@@_diagrams_filename_suffix_regex
      { \.[^.]*$ }
    \cs_set:Nn
      \@@_diagrams_infostrings_current:n
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \regex_replace_once:NnN
          \c_@@_diagrams_filename_suffix_regex
          { .pdf }
          \l_tmpa_tl
        \@@_diagrams_render_diagram:nVnn
          { #1 }
          \l_tmpa_tl
          { plantuml~-tpdf~#1 }
          { PlantUML~image }
      }
    \cs_generate_variant:Nn
      \@@_diagrams_render_diagram:nnnn
      { nVnn }
    \@@_tl_set_from_cs:NNn
      \l_tmpa_tl
      \@@_diagrams_infostrings_current:n
      { 1 }
    \prop_gput:NnV
      \g_markdown_diagrams_infostrings_prop
      { plantuml }
      \l_tmpa_tl
  }
```

We locally change the category code of percent signs, so that we can use them in the shell code:

```
\group_begin:
\char_set_catcode_other:N \%
```

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

```
\prop_gput:Nnn
  \g_@@_plain_tex_built_in_themes_prop
  { witiko / graphicx / http }
  {
    \cs_set_eq:NN
      \@@_graphicx_http_previous_definition:nnnn
      \markdownRendererImagePrototype
```

We define variables and functions to enumerate the images for caching and to store the pathname of the file containing the pathname of the downloaded image file.

```
    \int_new:N
      \g_@@_graphicx_http_image_number_int
    \int_gset:Nn
      \g_@@_graphicx_http_image_number_int
      { 0 }
    \cs_new:Nn
      \@@_graphicx_http_filename:
      {
        \markdownOptionCacheDir
        / witiko_graphicx_http .
        \int_use:N
          \g_@@_graphicx_http_image_number_int
      }
```

We define a function that will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The function produces a shell command that tries to downloads the online image to the pathname.

```
    \cs_new:Nn
      \@@_graphicx_http_download:nn
      {
        wget~-O~#2~#1~
        ||~curl~--location~-o~#2~#1~
        ||~rm~-f~#2
      }
```

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

```
    \str_new:N
      \l_@@_graphicx_http_filename_str
    \ior_new:N
      \g_@@_graphicx_http_filename_ior
    \markdownSetup {
      rendererPrototypes = {
        image = {
```

```
      \@@_if_option:nF
        { frozenCache }
        {
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TEX option is disabled:

```
          \sys_shell_now:e
            {
              mkdir~-p~" \markdownOptionCacheDir ";
              if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
              then~
```

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

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

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

```
                if~!~[~-e~"$OUTPUT"~];
                then~
                  \@@_graphicx_http_download:nn
                    { '#3' }
                    { "$OUTPUT" } ;
                  printf~'%s'~"$OUTPUT"~
                    >~" \@@_graphicx_http_filename: ";
                fi;
```

If the image does not have the http or https protocols or the image has already been downloaded, the URL will be stored as-is:

```
              else~
                printf~'%s'~'#3'~
                  >~" \@@_graphicx_http_filename: ";
              fi
            }
        }
```

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

```
      \ior_open:Ne
        \g_@@_graphicx_http_filename_ior
        { \@@_graphicx_http_filename: }
      \ior_str_get:NN
        \g_@@_graphicx_http_filename_ior
        \l_@@_graphicx_http_filename_str
      \ior_close:N
```

```
            \g_@@_graphicx_http_filename_ior
          \@@_graphicx_http_previous_definition:nnVn
            { #1 }
            { #2 }
            \l_@@_graphicx_http_filename_str
            { #4 }
          \int_gincr:N
            \g_@@_graphicx_http_image_number_int
        }
      }
    }
    \cs_generate_variant:Nn
      \ior_open:Nn
      { Ne }
    \cs_generate_variant:Nn
      \@@_graphicx_http_previous_definition:nnnn
      { nnVn }
  }
\group_end:
```

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

```
\prop_gput:Nnn
  \g_@@_plain_tex_built_in_themes_prop
  { witiko / tilde }
  {
    \markdownSetup {
      rendererPrototypes = {
        tilde = {~},
      },
    }
  }
```

The themes witiko/example/foo and witiko/example/bar are supposed to be used in code examples. They don't do anything.

```
\clist_map_inline:nn
  { foo, bar }
  {
    \prop_gput:Nnn
      \g_@@_plain_tex_built_in_themes_prop
      { witiko / example / #1 }
      {
        \markdownWarning
          {
            The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
            examples.~Using~it~in~actual~code~has~no~effect,~except~
            this~warning~message,~and~is~usually~a~mistake.
          }
```

```
      }
  }
\ExplSyntaxOff
```

The `witiko/markdown/defaults` plain TEX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererParagraphSeparatorPrototype{%
  \markdownRendererInterblockSeparator}%
\def\markdownRendererHardLineBreakPrototype{\hfil\break}%
\def\markdownRendererSoftLineBreakPrototype{ }%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`\{}%
\def\markdownRendererRightBracePrototype{\char`\}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\&}%
\def\markdownRendererUnderscorePrototype{\char`_}%
\def\markdownRendererHashPrototype{\char`\#}%
\def\markdownRendererCircumflexPrototype{\char`^}%
\def\markdownRendererBackslashPrototype{\char`\\}%
\def\markdownRendererTildePrototype{\char`~}%
\def\markdownRendererPipePrototype{|}%
\def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
\def\markdownRendererLinkPrototype#1#2#3#4{#2}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \markdownInput{#3}}%
\def\markdownRendererContentBlockOnlineImagePrototype{%
  \markdownRendererImage}%
\def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
  \markdownRendererInputFencedCode{#3}{#2}{#2}}%
\def\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererFancyOlBeginPrototype#1#2{%
  \markdownRendererOlBegin}%
```

```
\def\markdownRendererFancyOlBeginTightPrototype#1#2{%
  \markdownRendererOlBeginTight}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
\def\markdownRendererFancyOlItemWithNumberPrototype{%
  \markdownRendererOlItemWithNumber}%
\def\markdownRendererFancyOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
\def\markdownRendererFancyOlEndTightPrototype{%
  \markdownRendererOlEndTight}%
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{#1}%
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{\par}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{{\it#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
\def\markdownRendererLineBlockEndPrototype{\endgroup}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input#1\relax{}}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2#3{%
  \markdownRendererInputVerbatim{#1}}%
\def\markdownRendererHeadingOnePrototype#1{#1}%
\def\markdownRendererHeadingTwoPrototype#1{#1}%
\def\markdownRendererHeadingThreePrototype#1{#1}%
\def\markdownRendererHeadingFourPrototype#1{#1}%
\def\markdownRendererHeadingFivePrototype#1{#1}%
\def\markdownRendererHeadingSixPrototype#1{#1}%
\def\markdownRendererThematicBreakPrototype{}%
\def\markdownRendererNotePrototype#1{#1}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
\def\markdownRendererTickedBoxPrototype{[X]}%
\def\markdownRendererHalfTickedBoxPrototype{[/]}%
\def\markdownRendererUntickedBoxPrototype{[ ]}%
\def\markdownRendererStrikeThroughPrototype#1{#1}%
\def\markdownRendererSuperscriptPrototype#1{#1}%
```

```
\def\markdownRendererSubscriptPrototype#1{#1}%
\def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
\def\markdownRendererInlineMathPrototype#1{$#1$}%
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextBeginPrototype
  {
    \group_begin:
    \color_group_begin:
  }
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextEndPrototype
  {
    \color_group_end:
    \group_end:
  }
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset:Npn
  \markdownRendererReplacementCharacterPrototype
  { \codepoint_str_generate:n { fffd } }
\ExplSyntaxOff
\def\markdownRendererSectionBeginPrototype{}%
\def\markdownRendererSectionEndPrototype{}%
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererWarningPrototype
  #1#2#3#4
  {
    \tl_set:Nn
      \l_tmpa_tl
```

```
      { #2 }
    \tl_if_empty:nF
      { #4 }
      {
        \tl_put_right:Nn
          \l_tmpa_tl
          { \iow_newline: #4 }
      }
    \exp_args:NV
      \markdownWarning
      \l_tmpa_tl
  }
\ExplSyntaxOff
\def\markdownRendererErrorPrototype#1#2#3#4{%
  \markdownError{#2}{#4}}%
```

3.2.3.1 Raw Attributes

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

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_plain_tex_default_input_raw_inline:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline:nn
      { #1 }
```

```
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key–value `markdown/jekyllData`. See also Section 2.2.6.1.

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

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth p.

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

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

```
\ExplSyntaxOn
\seq_new:N   \g_@@_jekyll_data_datatypes_seq
\tl_const:Nn \c_@@_jekyll_data_sequence_tl   { sequence }
\tl_const:Nn \c_@@_jekyll_data_mapping_tl    { mapping  }
\tl_const:Nn \c_@@_jekyll_data_scalar_tl     { scalar   }
```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\@@_jekyll_data_push_address_segment:n` macro.

```
\seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
\cs_new:Nn \@@_jekyll_data_push_address_segment:n
  {
    \seq_if_empty:NF
      \g_@@_jekyll_data_datatypes_seq
      {
        \seq_get_right:NN
          \g_@@_jekyll_data_datatypes_seq
          \l_tmpa_tl
```

If we are currently in a sequence, we will put an asterisk (*) instead of a key into \g_@@_jekyll_data_wildcard_absolute_address_seq to make it represent a *wildcard*. Keeping a wildcard instead of a precise address makes it easy for the users to react to *any* item of a sequence regardless of how many there are, which can often be useful.

```
      \str_if_eq:NNTF
        \l_tmpa_tl
        \c_@@_jekyll_data_sequence_tl
        {
          \seq_put_right:Nn
            \g_@@_jekyll_data_wildcard_absolute_address_seq
            { *  }
        }
        {
          \seq_put_right:Nn
            \g_@@_jekyll_data_wildcard_absolute_address_seq
            { #1 }
        }
    }
}
```

Out of \g_@@_jekyll_data_wildcard_absolute_address_seq, we will construct the following two token lists:

\g_@@_jekyll_data_wildcard_absolute_address_tl An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the name key in the following YAML document would correspond to the /*/person/name absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

\g_@@_jekyll_data_wildcard_relative_address_tl A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the name key in the following YAML document would correspond to the name relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct \g_@@_jekyll_data_wildcard_absolute_address_tl using the \@@_jekyll_data_concatenate_address:NN macro and we will construct both token lists using the \@@_jekyll_data_update_address_tls: macro.

```
\tl_new:N  \g_@@_jekyll_data_wildcard_absolute_address_tl
\tl_new:N  \g_@@_jekyll_data_wildcard_relative_address_tl
\cs_new:Nn \@@_jekyll_data_concatenate_address:NN
  {
    \seq_pop_left:NN #1 \l_tmpa_tl
    \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
    \seq_put_left:NV #1 \l_tmpa_tl
  }
\cs_new:Nn \@@_jekyll_data_update_address_tls:
  {
    \@@_jekyll_data_concatenate_address:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_absolute_address_tl
    \seq_get_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_relative_address_tl
  }
```

To make sure that the stacks and token lists stay in sync, we will use the `\@@_jekyll_data_push:nN` and `\@@_jekyll_data_pop:` macros.

```
\cs_new:Nn \@@_jekyll_data_push:nN
  {
    \@@_jekyll_data_push_address_segment:n
      { #1 }
    \seq_put_right:NV
     \g_@@_jekyll_data_datatypes_seq
     #2
    \@@_jekyll_data_update_address_tls:
  }
\cs_new:Nn \@@_jekyll_data_pop:
  {
    \seq_pop_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \l_tmpa_tl
    \seq_pop_right:NN
      \g_@@_jekyll_data_datatypes_seq
      \l_tmpa_tl
    \@@_jekyll_data_update_address_tls:
  }
```

To set a single key–value, we will use the `\@@_jekyll_data_set_keyval_known:nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\@@_jekyll_data_set_keyvals_known:nn` macro.

```
\cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
  {
    \keys_set_known:nn
      { markdown/jekyllData }
      { { #1 } = { #2 } }
```

```
  }
\cs_generate_variant:Nn
  \@@_jekyll_data_set_keyval_known:nn
  { Vn }
\cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
  {
    \@@_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_scalar_tl
    \@@_jekyll_data_set_keyval_known:Vn
      \g_@@_jekyll_data_wildcard_absolute_address_tl
      { #2 }
    \@@_jekyll_data_set_keyval_known:Vn
      \g_@@_jekyll_data_wildcard_relative_address_tl
      { #2 }
    \@@_jekyll_data_pop:
  }
```

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

```
\def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
  \@@_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_sequence_tl
}
\def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
  \@@_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_mapping_tl
}
\def\markdownRendererJekyllDataSequenceEndPrototype{
  \@@_jekyll_data_pop:
}
\def\markdownRendererJekyllDataMappingEndPrototype{
  \@@_jekyll_data_pop:
}
\def\markdownRendererJekyllDataBooleanPrototype#1#2{
  \@@_jekyll_data_set_keyvals_known:nn
    { #1 }
    { #2 }
}
\def\markdownRendererJekyllDataEmptyPrototype#1{}
\def\markdownRendererJekyllDataNumberPrototype#1#2{
  \@@_jekyll_data_set_keyvals_known:nn
    { #1 }
    { #2 }
}
```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```
\def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
\def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
  \@@_jekyll_data_set_keyvals_known:nn
    { #1 }
    { #2 }
}
\ExplSyntaxOff
```

3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key–values using the option `jekyllDataKeyValue`=⟨*module*⟩. See also Section 2.2.6.1.

```
\ExplSyntaxOn
\@@_with_various_cases:nn
  { jekyllDataKeyValue }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \@@_route_jekyll_data_to_key_values:n
            { ##1 }
        },
```

When no ⟨*module*⟩ has been provided, assume that the YAML metadata specify absolute paths to key–values.

```
        #1 .default:n = { },
      }
  }
\seq_new:N
  \l_@@_jekyll_data_current_position_seq
\tl_new:N
  \l_@@_jekyll_data_current_position_tl
\cs_new:Nn
  \@@_route_jekyll_data_to_key_values:n
  {
    \markdownSetup
      {
        renderers = {
          jekyllData(Sequence|Mapping)Begin = {
            \bool_lazy_and:nnTF
              {
                \seq_if_empty_p:N
                  \l_@@_jekyll_data_current_position_seq
```

```
      }
      {
        \str_if_eq_p:nn
          { ##1 }
          { null }
      }
      {
        \tl_if_empty:nF
          { #1 }
          {
            \seq_put_right:Nn
              \l_@@_jekyll_data_current_position_seq
              { #1 }
          }
      }
      {
        \seq_put_right:Nn
          \l_@@_jekyll_data_current_position_seq
          { ##1 }
      }
  },
  jekyllData(Sequence|Mapping)End = {
    \seq_pop_right:NN
      \l_@@_jekyll_data_current_position_seq
      \l_tmpa_tl
  },
```

For every YAML key `path.to.`⟨*key*⟩ with a value of type ⟨*non-string type*⟩, set the key ⟨*non-string type*⟩ of the key–value ⟨*module*⟩`/path/to/`⟨*key*⟩ if it is known and the key ⟨*key*⟩ of the key–value ⟨*module*⟩`/path/to` otherwise. ⟨*Non-string type*⟩ is one of `boolean`, `number`, and `empty`.

```
  jekyllDataBoolean = {
    \tl_set:Nx
      \l_@@_jekyll_data_current_position_tl
      {
        \seq_use:Nn
          \l_@@_jekyll_data_current_position_seq
          { / }
      }
    \keys_if_exist:VnTF
      \l_@@_jekyll_data_current_position_tl
      { ##1 / boolean }
      {
        \@@_keys_set:xn
          {
            \tl_use:N
              \l_@@_jekyll_data_current_position_tl
```

```
                / ##1 / boolean
              }
              { ##2 }
          }
          {
            \@@_keys_set:xn
              {
                \tl_use:N
                  \l_@@_jekyll_data_current_position_tl
                / ##1
              }
              { ##2 }
          }
      },
      jekyllDataNumber = {
        \tl_set:Nx
          \l_@@_jekyll_data_current_position_tl
          {
            \seq_use:Nn
              \l_@@_jekyll_data_current_position_seq
              { / }
          }
        \keys_if_exist:VnTF
          \l_@@_jekyll_data_current_position_tl
          { ##1 / number }
          {
            \@@_keys_set:xn
              {
                \tl_use:N
                  \l_@@_jekyll_data_current_position_tl
                / ##1 / number
              }
              { ##2 }
          }
          {
            \@@_keys_set:xn
              {
                \tl_use:N
                  \l_@@_jekyll_data_current_position_tl
                / ##1
              }
              { ##2 }
          }
      },
```

For the ⟨*non-string type*⟩ of `empty`, no value is passed to the key–value. Therefore,

a default value should always be defined for nullable keys using the key property `.default:n`.

```
jekyllDataEmpty = {
  \tl_set:Nx
    \l_@@_jekyll_data_current_position_tl
    {
      \seq_use:Nn
        \l_@@_jekyll_data_current_position_seq
        { / }
    }
  \keys_if_exist:VnTF
    \l_@@_jekyll_data_current_position_tl
    { ##1 / empty }
    {
        \keys_set:xn
        {
          \tl_use:N
            \l_@@_jekyll_data_current_position_tl
          / ##1
        }
        { empty }
    }
    {
      \keys_set:Vn
        \l_@@_jekyll_data_current_position_tl
        { ##1 }
    }
},
```

For every YAML key `path.to.`⟨*key*⟩ with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key–value ⟨*module*⟩`/path/to/`⟨*key*⟩ if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key ⟨*key*⟩ of the key–value ⟨*module*⟩`/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key ⟨*key*⟩ if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set ⟨*key*⟩ normally, i.e. regardless of whether it is known or unknown.

```
jekyllDataTypographicString = {
  \tl_set:Nx
    \l_@@_jekyll_data_current_position_tl
    {
      \seq_use:Nn
        \l_@@_jekyll_data_current_position_seq
        { / }
    }
```

```
      \keys_if_exist:VnTF
        \l_@@_jekyll_data_current_position_tl
        { ##1 / typographicString }
        {
          \@@_keys_set:xn
            {
              \tl_use:N
                \l_@@_jekyll_data_current_position_tl
              / ##1 / typographicString
            }
            { ##2 }
        }
        {
          \keys_if_exist:VnTF
            \l_@@_jekyll_data_current_position_tl
            { ##1 / programmaticString }
            {
              \@@_keys_set_known:xn
                {
                  \tl_use:N
                    \l_@@_jekyll_data_current_position_tl
                  / ##1
                }
                { ##2 }
            }
            {
              \@@_keys_set:xn
                {
                  \tl_use:N
                    \l_@@_jekyll_data_current_position_tl
                  / ##1
                }
                { ##2 }
            }
        }
    },
    jekyllDataProgrammaticString = {
      \tl_set:Nx
        \l_@@_jekyll_data_current_position_tl
        {
          \seq_use:Nn
            \l_@@_jekyll_data_current_position_seq
            { / }
        }
      \keys_if_exist:VnT
        \l_@@_jekyll_data_current_position_tl
        { ##1 / programmaticString }
```

```
              {
                \@@_keys_set:xn
                  {
                    \tl_use:N
                      \l_@@_jekyll_data_current_position_tl
                    / ##1 / programmaticString
                  }
                  { ##2 }
              }
          },
        },
      }
  }
\cs_new:Nn
  \@@_keys_set:nn
  {
    \keys_set:nn
      { }
      { { #1 } = { #2 } }
  }
\cs_new:Nn
  \@@_keys_set_known:nn
  {
    \keys_set_known:nn
      { }
      { { #1 } = { #2 } }
  }
\cs_generate_variant:Nn
  \@@_keys_set:nn
  { xn }
\cs_generate_variant:Nn
  \@@_keys_set_known:nn
  { xn }
\cs_generate_variant:Nn
  \keys_set:nn
  { xn, Vn }
\prg_generate_conditional_variant:Nnn
  \keys_if_exist:nn
  { Vn }
  { T, TF }
\ExplSyntaxOff
```

If plain TEX is the top layer, we load the `witiko/markdown/defaults` plain TEX theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
\str_if_eq:VVT
```

```
  \c_@@_top_layer_tl
  \c_@@_option_layer_plain_tex_tl
  {
    \use:c
      { ExplSyntaxOff }
    \@@_if_option:nF
      { noDefaults }
      {
        \@@_if_option:nTF
          { experimental }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults@experimental }
          }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults }
          }
      }
    \use:c
      { ExplSyntaxOn }
  }
\ExplSyntaxOff
```

3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expands to a Lua table that contains the plain TEX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```
\ExplSyntaxOn
\tl_new:N \g_@@_formatted_lua_options_tl
\cs_new:Nn \@@_format_lua_options:
  {
    \tl_gclear:N
      \g_@@_formatted_lua_options_tl
    \seq_map_function:NN
      \g_@@_lua_options_seq
      \@@_format_lua_option:n
  }
\cs_new:Nn \@@_format_lua_option:n
  {
    \@@_typecheck_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
```

```
    \bool_case_true:nF
      {
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_boolean_tl ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_number_tl ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_counter_tl
        }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { #1~=~ \l_tmpa_tl ,~ }
          }
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_clist_tl
        }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { #1~=~\c_left_brace_str }
            \clist_map_inline:Vn
              \l_tmpa_tl
              {
                \@@_lua_escape:xN
                  { ##1 }
                  \l_tmpb_tl
                \tl_gput_right:Nn
                  \g_@@_formatted_lua_options_tl
                  { " }
                \tl_gput_right:NV
                  \g_@@_formatted_lua_options_tl
                  \l_tmpb_tl
                \tl_gput_right:Nn
                  \g_@@_formatted_lua_options_tl
```

```
                  { " ,~ }
              }
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { \c_right_brace_str ,~ }
          }
      }
      {
        \@@_get_option_value:nN
          { #1 }
          \l_tmpa_tl
        \@@_lua_escape:xN
          { \l_tmpa_tl }
          \l_tmpb_tl
        \tl_gput_right:Nn
          \g_@@_formatted_lua_options_tl
          { #1~=~ " }
        \tl_gput_right:NV
          \g_@@_formatted_lua_options_tl
          \l_tmpb_tl
        \tl_gput_right:Nn
          \g_@@_formatted_lua_options_tl
          { " ,~ }
      }
  }
\cs_generate_variant:Nn
  \clist_map_inline:nn
  { Vn }
\let
  \markdownPrepareLuaOptions
  \@@_format_lua_options:
\def
  \markdownLuaOptions
    {
      {
        \g_@@_formatted_lua_options_tl
      }
    }
\sys_if_engine_luatex:TF
  {
    \cs_new:Nn
      \@@_lua_escape:nN
      {
        \tl_set:Nx
          #2
          {
            \lua_escape:n
```

```
              { #1 }
          }
      }
  }
  {
    \regex_const:Nn
      \c_@@_lua_escape_regex
      { [\\"'] }
    \cs_new:Nn
      \@@_lua_escape:nN
      {
        \tl_set:Nn
          #2
          { #1 }
        \regex_replace_all:NnN
          \c_@@_lua_escape_regex
          { \u { c_backslash_str } \0 }
          #2
      }
  }
\cs_generate_variant:Nn
  \@@_lua_escape:nN
  { xN }
```

After the `\markdownPrepareInputFilename` macro has been fully expanded, the `\markdownInputFilename` macro will expands to a Lua string that contains the input filename passed as the first argument.

```
\tl_new:N
  \markdownInputFilename
\cs_new:Npn
  \markdownPrepareInputFilename
  #1
  {
    \@@_lua_escape:xN
      { #1 }
      \markdownInputFilename
    \tl_gset:Nx
      \markdownInputFilename
      { " \markdownInputFilename " }
  }
```

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

```
\cs_new:Npn
  \markdownPrepare
  {
```

First, ensure that the `cacheDir` directory exists.

```
14236     local~lfs = require("lfs")
14237     local~options = \markdownLuaOptions
14238     if~not~lfs.isdir(options.cacheDir) then~
14239       assert(lfs.mkdir(options.cacheDir))
14240     end~
```

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

```
14241     local~md = require("markdown")
14242     local~convert = md.new(options)
14243   }
```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain TEX. It opens the input file, converts it, and prints the conversion result.

```
14244 \cs_new:Npn
14245   \markdownConvert
14246   {
14247     local~filename = \markdownInputFilename
14248     local~file = assert(io.open(filename, "r"),
14249       [[Could~not~open~file~"]] .. filename .. [["~for~reading]])
14250     local~input = assert(file:read("*a"))
14251     assert(file:close())
14252     print(convert(input))
14253   }
14254 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TEX.

```
14255 \def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
14256   if options.cacheDir then
14257     lfs.rmdir(options.cacheDir)
14258   end
14259 }%
```

3.2.5 Buffering Block-Level Markdown Input

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

```
14260 \csname newread\endcsname\markdownInputFileStream
14261 \csname newwrite\endcsname\markdownOutputFileStream
```

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

```
14262 \begingroup
```

```
  \catcode`\^^I=12%
  \gdef\markdownReadAndConvertTab{^^I}%
\endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the LaTeX 2_ε `\filecontents` macro to plain TeX.

```
\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 `stripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

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

```
      |markdownIfOption{frozenCache}{}{@
        |immediate|openout|markdownOutputFileStream@
          |markdownOptionInputTempFileName|relax@
        |markdownInfo{@
          Buffering block-level markdown input into the temporary @
          input file "|markdownOptionInputTempFileName" and scanning @
          for the closing token sequence "#1"}@
      }@
```

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

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

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stipping away leading percent signs (`%`) when `stripPercentSigns` 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.

```
      |def|markdownReadAndConvertStripPercentSign##1{@
        |markdownIfOption{stripPercentSigns}{@
          |if##1%@
            |expandafter|expandafter|expandafter@
```

```
            |markdownReadAndConvertProcessLine@
        |else@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
            |expandafter|expandafter|expandafter##1@
        |fi@
      }{@
        |expandafter@
          |markdownReadAndConvertProcessLine@
          |expandafter##1@
      }@
    }@
```

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.

```
    |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 `inputTempFileName` 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.

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

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

```
        |def^^M{@
          |markdownInfo{The ending token sequence was found}@
          |markdownIfOption{frozenCache}{}{@
            |immediate|closeout|markdownOutputFileStream@
          }@
          |endgroup@
          |markdownInput{@
            |markdownOptionOutputDir@
            /|markdownOptionInputTempFileName@
          }@
          #2}@
      |fi@
```

Repeat with the next line.

```
      ^^M}@
```

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

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

```
    |catcode`|^^M=13@
    |def^^M##1^^M{@
      |def^^M####1^^M{@
        |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
      ^^M}@
    ^^M}@
```

Reset the character categories back to the former state.

```
|endgroup
```

Use the lt3luabridge library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownLuaExecute
  #1
  {
    \int_compare:nNnT
      { \g_luabridge_method_int }
      =
      { \c_luabridge_method_shell_int }
      {
        \sys_if_shell_unrestricted:F
          {
            \sys_if_shell:TF
              {
                \msg_error:nn
                  { markdown }
                  { restricted-shell-access }
              }
              {
                \msg_error:nn
                  { markdown }
                  { disabled-shell-access }
              }
          }
      }
    \str_gset:NV
      \g_luabridge_output_dirname_str
      \markdownOptionOutputDir
```

```
    \luabridge_now:e
      { #1 }
  }
\cs_generate_variant:Nn
  \msg_new:nnnn
  { nnnV }
\tl_set:Nn
  \l_tmpa_tl
  {
    You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
    --enable-write18~flag,~or~write~shell_escape=t~in~the~
    texmf.cnf~file.
  }
\msg_new:nnnV
  { markdown }
  { restricted-shell-access }
  { Shell~escape~is~restricted }
  \l_tmpa_tl
\msg_new:nnnV
  { markdown }
  { disabled-shell-access }
  { Shell~escape~is~disabled }
  \l_tmpa_tl
\ExplSyntaxOff
```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```
\ExplSyntaxOn
\tl_new:N
  \g_@@_after_markinline_tl
\tl_gset:Nn
  \g_@@_after_markinline_tl
  { \unskip }
\cs_new:Npn
  \markinline
  {
```

Locally change the category of the special plain TEX characters to *other* in order to prevent unwanted interpretation of the input markdown text as TEX code.

```
    \group_begin:
    \cctab_select:N
      \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
    \@@_if_option:nF
```

```
      { frozenCache }
      {
        \immediate
          \openout
          \markdownOutputFileStream
          \markdownOptionInputTempFileName
          \relax
        \msg_info:nne
          { markdown }
          { buffering-markinline }
          { \markdownOptionInputTempFileName }
      }
```

Peek ahead and extract the inline markdown text.

```
    \peek_regex_replace_once:nnF
      { { (.*?) } }
      {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
        \c { @@_if_option:nF }
          \cB { frozenCache \cE }
          \cB {
            \c { immediate }
              \c { write }
              \c { markdownOutputFileStream }
              \cB { \1 \cE }
            \c { immediate }
              \c { closeout }
              \c { markdownOutputFileStream }
          \cE }
```

Reset the category codes and `\input` the result of the conversion.

```
        \c { group_end: }
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { contentLevel = inline \cE }
        \c { markdownInput }
          \cB {
            \c { markdownOptionOutputDir } /
            \c { markdownOptionInputTempFileName }
          \cE }
        \c { group_end: }
        \c { tl_use:N }
          \c { g_@@_after_markinline_tl }
      }
      {
        \msg_error:nn
```

```
          { markdown }
          { markinline-peek-failure }
        \group_end:
        \tl_use:N
          \g_@@_after_markinline_tl
      }
  }
\msg_new:nnn
  { markdown }
  { buffering-markinline }
  { Buffering~inline~markdown~input~into~
    the~temporary~input~file~"#1". }
\msg_new:nnnn
  { markdown }
  { markinline-peek-failure }
  { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
  { The~macro~should~be~followed~by~inline~
    markdown~text~in~curly~braces }
\ExplSyntaxOff
```

3.2.7 Typesetting Markdown

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

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownInput
  #1
  {
    \@@_if_option:nTF
      { frozenCache }
      {
        \markdownInputRaw
          { #1 }
      }
      {
```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On LATEX, this also includes the directories specified in `\input@path`.

```
        \tl_set:Nx
          \l_tmpa_tl
          { #1 }
        \file_get_full_name:VNTF
          \l_tmpa_tl
          \l_tmpb_tl
```

```
          {
            \exp_args:NV
              \markdownInputRaw
              \l_tmpb_tl
          }
          {
            \msg_error:nnV
              { markdown }
              { markdown-file-does-not-exist }
              \l_tmpa_tl
          }
      }
  }
\msg_new:nnn
  { markdown }
  { markdown-file-does-not-exist }
  {
    Markdown~file~#1~does~not~exist
  }
\ExplSyntaxOff
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code. Furthermore, use the ampersand symbol to specify parameters.

```
  \catcode`|=0%
  \catcode`\\=12%
  \catcode`|&=6%
  |gdef|markdownInputRaw#1{%
```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |begingroup
    |catcode`|%=12
```

Furthermore, also change the category code of the hash sign (#) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
    |catcode`|#=12
```

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

```
    |markdownIfOption{frozenCache}{%
      |ifnum|markdownOptionFrozenCacheCounter=0|relax
        |markdownInfo{Reading frozen cache from
          "|markdownOptionFrozenCacheFileName"}%
        |input|markdownOptionFrozenCacheFileName|relax
```

```
      |fi
      |markdownInfo{Including markdown document number
        "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
      |csname markdownFrozenCache%
        |the|markdownOptionFrozenCacheCounter|endcsname
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }{%
      |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 LaTeXMk to track changes to the markdown document.

```
      |openin|markdownInputFileStream&1
      |closein|markdownInputFileStream
      |markdownPrepareLuaOptions
      |markdownPrepareInputFilename{&1}%
      |markdownLuaExecute{%
        |markdownPrepare
        |markdownConvert
        |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
      |markdownIfOption{finalizeCache}{%
        |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
    }%
    |endgroup
  }%
|endgroup
```

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

```
\gdef\markdownEscape#1{%
  \catcode`\%=14\relax
  \catcode`\#=6\relax
  \input #1\relax
  \catcode`\%=12\relax
  \catcode`\#=12\relax
}%
```

3.3 LaTeX Implementation

The LaTeX implementation makes use of the fact that, apart from some subtle differences, LaTeX implements the majority of the plain TeX format [17, Section 9]. As a consequence, we can directly reuse the existing plain TeX implementation.

```
\def\markdownVersionSpace{ }%
```

```
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain TEX implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the LATEX interface (see Section 2.3.3).

```
\ExplSyntaxOn
\cs_gset_eq:NN
  \markinlinePlainTeX
  \markinline
\cs_gset:Npn
  \markinline
  {
    \peek_regex_replace_once:nn
      { ( \[ (.*?) \] ) ? }
      {
```

Apply the options locally.

```
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { \2 \cE }
        \c { tl_put_right:Nn }
          \c { g_@@_after_markinline_tl }
          \cB { \c { group_end: } \cE }
        \c { markinlinePlainTeX }
      }
  }
\ExplSyntaxOff
```

The `\markdownInputPlainTeX` macro is used to store the original plain TEX implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the LATEX interface (see Section 2.3.3).

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
\renewcommand\yamlInput[2][]{%
  \begingroup
    \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

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

```
\ExplSyntaxOn
\renewenvironment
  { markdown }
  {
```

In our implementation of the `markdown` LaTeX environment, we want to distinguish between the following two cases:

```
\begin{markdown} [smartEllipses]
% This is an optional argument ^
% ...
\end{markdown}
```

```
\begin{markdown}
  [smartEllipses]
% ^ This is link
\end{markdown}
```

Therefore, we cannot use the built-in LaTeX support for environments with optional arguments or packages such as xparse. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` LaTeX environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by TeX via the `\endlinechar` plain TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
    \group_begin:
    \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
    \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
    \peek_regex_replace_once:nnF
      { \ *\[\r*([^]]*)\][^\r]* }
      {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized

as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
        \c { group_end: }
        \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the \markdownSetup macro.

```
        \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the \markdownReadAndConvert macro process the rest of the LaTeX environment.

We also make provision for using the \markdown command as a part of a different LaTeX environment as follows:

```
\newenvironment{foo}%
               {code before \markdown[some, options]}%
               {\markdownEnd code after}
```

```
        \c { exp_args:NV }
          \c { markdownReadAndConvert@ }
          \c { @currenvir }
      }
      {
        \group_end:
        \exp_args:NV
          \markdownReadAndConvert@
          \@currenvir
      }
  }
  { \markdownEnd }
\renewenvironment
  { markdown* }
  [ 1 ]
  {
    \@@_if_option:nTF
      { experimental }
      {
        \msg_error:nnn
          { markdown }
          { latex-markdown-star-deprecated }
          { #1 }
      }
      {
        \msg_warning:nnn
          { markdown }
          { latex-markdown-star-deprecated }
          { #1 }
```

```
      }
    \@@_setup:n
      { #1 }
    \markdownReadAndConvert@
      { markdown* }
  }
  { \markdownEnd }
\renewenvironment
  { yaml }
  {
    \group_begin:
    \yamlSetup
      { jekyllData, expectJekyllData, ensureJekyllData }
    \markdown
  }
  { \yamlEnd }
\msg_new:nnn
  { markdown }
  { latex-markdown-star-deprecated }
  {
    The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
    be~removed~in~the~next~major~version~of~the~Markdown~package.
  }
\cs_generate_variant:Nn
  \@@_setup:n
  { V }
\ExplSyntaxOff
\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*.

```
  \catcode`\|=0\catcode`\<=1\catcode`\>=2%
  \catcode`\\=12|catcode`|{=12|catcode`|}=12%
  |gdef|markdownReadAndConvert@#1<%
    |markdownReadAndConvert<\end{#1}>%
                           <|end<#1>>>%
|endgroup
```

3.3.2 Themes

This section overrides the plain TEX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in LATEX themes provided with the Markdown package.

```
\ExplSyntaxOn
```

```
\prop_new:N \g_@@_latex_loaded_themes_linenos_prop
\prop_new:N \g_@@_latex_loaded_themes_versions_prop
\cs_gset:Nn
  \@@_load_theme:nnn
  {
```

If the Markdown package has not yet been loaded, determine whether either this is a built-in theme according to the prop `\g_@@_latex_built_in_themes_prop` or a file named `markdowntheme`⟨*munged theme name*⟩`.sty` exists and whether we are still in the preamble.

```
    \ifmarkdownLaTeXLoaded
      \ifx\@onlypreamble\@notprerr
```

If both conditions are true, end with an error, since we cannot load LaTeX themes after the preamble.

```
        \bool_if:nTF
          {
            \bool_lazy_or_p:nn
              {
                \prop_if_in_p:Nn
                  \g_@@_latex_built_in_themes_prop
                  { #1 }
              }
              {
                \file_if_exist_p:n
                  { markdown theme #3.sty }
              }
          }
          {
            \msg_error:nnn
              { markdown }
              { latex-theme-after-preamble }
              { #1 }
          }
```

Otherwise, try loading a plain TeX theme instead.

```
          {
            \@@_plain_tex_load_theme:nnn
              { #1 }
              { #2 }
              { #3 }
          }
      \else
```

If the Markdown package has already been loaded but we are still in the preamble, load a LaTeX theme if it exists or load a plain TeX theme otherwise.

```
        \bool_if:nTF
          {
```

```
      \bool_lazy_or_p:nn
        {
          \prop_if_in_p:Nn
            \g_@@_latex_built_in_themes_prop
            { #1 }
        }
        {
          \file_if_exist_p:n
            { markdown theme #3.sty }
        }
    }
    {
      \prop_get:NnNTF
        \g_@@_latex_loaded_themes_linenos_prop
        { #1 }
        \l_tmpa_tl
        {
          \prop_get:NnN
            \g_@@_latex_loaded_themes_versions_prop
            { #1 }
            \l_tmpb_tl
          \str_if_eq:nVTF
            { #2 }
            \l_tmpb_tl
            {
              \msg_warning:nnnVn
                { markdown }
                { repeatedly-loaded-latex-theme }
                { #1 }
                \l_tmpa_tl
                { #2 }
            }
            {
              \msg_error:nnnnVV
                { markdown }
                { different-versions-of-latex-theme }
                { #1 }
                { #2 }
                \l_tmpb_tl
                \l_tmpa_tl
            }
        }
        {
          \prop_gput:Nnx
            \g_@@_latex_loaded_themes_linenos_prop
            { #1 }
            { \tex_the:D \tex_inputlineno:D }  % noqa: W200
```

```
                \prop_gput:Nnn
                  \g_@@_latex_loaded_themes_versions_prop
                  { #1 }
                  { #2 }
```

Load built-in plain TEX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```
                \prop_if_in:NnTF
                  \g_@@_latex_built_in_themes_prop
                  { #1 }
                  {
                    \msg_info:nnnn
                      { markdown }
                      { loading-built-in-latex-theme }
                      { #1 }
                      { #2 }
                    \prop_item:Nn
                      \g_@@_latex_built_in_themes_prop
                      { #1 }
                  }
                  {
                    \msg_info:nnnn
                      { markdown }
                      { loading-latex-theme }
                      { #1 }
                      { #2 }
                    \RequirePackage
                      { markdown theme #3 }
                  }
              }
          }
          {
            \@@_plain_tex_load_theme:nnn
              { #1 }
              { #2 }
              { #3 }
          }
      \fi
    \else
```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```
      \msg_info:nnnn
        { markdown }
        { theme-loading-postponed }
        { #1 }
        { #2 }
      \AtEndOfPackage
```

```
        {
          \@@_set_theme:n
            { #1 @ #2 }
        }
    \fi
  }
\msg_new:nnn
  { markdown }
  { theme-loading-postponed }
  {
    Postponing~loading~version~#2~of~Markdown~theme~#1~until~
    Markdown~package~has~finished~loading
  }
\msg_new:nnn
  { markdown }
  { loading-built-in-latex-theme }
  { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { loading-latex-theme }
  { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-latex-theme }
  {
    Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\msg_new:nnn
  { markdown }
  { different-versions-of-latex-theme }
  {
    Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\cs_generate_variant:Nn
  \msg_new:nnnn
  { nnVV }
\tl_set:Nn
  \l_tmpa_tl
  { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
\tl_put_right:NV
  \l_tmpa_tl
  \c_backslash_str
\tl_put_right:Nn
  \l_tmpa_tl
  { begin { document } }
```

```
\tl_set:Nn
  \l_tmpb_tl
  { Load~Markdown~theme~#1~before~ }
\tl_put_right:NV
  \l_tmpb_tl
  \c_backslash_str
\tl_put_right:Nn
  \l_tmpb_tl
  { begin { document } }
\msg_new:nnVV
  { markdown }
  { latex-theme-after-preamble }
  \l_tmpa_tl
  \l_tmpb_tl
```

The `witiko/dot` and `witiko/graphicx/http` LaTeX themes load the package graphicx, see also Section 1.1.3. Then, they load the corresponding plain TeX themes.

```
\tl_set:Nn
  \l_tmpa_tl
  {
    \RequirePackage
      { graphicx }
    \markdownLoadPlainTeXTheme
  }
\prop_gput:NnV
  \g_@@_latex_built_in_themes_prop
  { witiko / dot }
  \l_tmpa_tl
\prop_gput:NnV
  \g_@@_latex_built_in_themes_prop
  { witiko / graphicx / http }
  \l_tmpa_tl
\ExplSyntaxOff
```

The `witiko/markdown/defaults` LaTeX theme also loads the corresponding plain TeX theme.

```
\markdownLoadPlainTeXTheme
```

Next, the LaTeX theme overrides some of the plain TeX definitions. See Section 3.3.4 for the actual definitions.

3.3.3 Options

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

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

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

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

3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package paralist or the package enumitem have already been loaded, use them. Otherwise, if the option `experimental` or any test phase has been enabled, use the package enumitem. Otherwise, use the package paralist.

```
\ExplSyntaxOn
\bool_new:N
  \g_@@_tight_or_fancy_lists_bool
\bool_gset_false:N
  \g_@@_tight_or_fancy_lists_bool
\@@_if_option:nTF
  { tightLists }
  {
    \bool_gset_true:N
      \g_@@_tight_or_fancy_lists_bool
  }
  {
    \@@_if_option:nT
      { fancyLists }
      {
        \bool_gset_true:N
          \g_@@_tight_or_fancy_lists_bool
      }
  }
\bool_new:N
  \g_@@_beamer_paralist_or_enumitem_bool
\bool_gset_true:N
  \g_@@_beamer_paralist_or_enumitem_bool
\@ifclassloaded
  { beamer }
  { }
  {
    \@ifpackageloaded
      { paralist }
      { }
      {
        \@ifpackageloaded
        { enumitem }
```

```
        { }
        {
          \bool_gset_false:N
            \g_@@_beamer_paralist_or_enumitem_bool
        }
      }
  }
\bool_if:nT
  {
    \g_@@_tight_or_fancy_lists_bool &&
    ! \g_@@_beamer_paralist_or_enumitem_bool
  }
  {
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          {
            \str_if_eq_p:en
              { \markdownThemeVersion }
              { experimental }
          }
          {
            \bool_lazy_and_p:nn
              {
                \prop_if_exist_p:N
                  \g__pdfmanagement_documentproperties_prop
              }
              {
                \bool_lazy_any_p:n
                  {
                    {
                      \prop_if_in_p:Nn
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-I }
                    }
                    {
                      \prop_if_in_p:Nn
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-II }
                    }
                    {
                      \prop_if_in_p:Nn
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-III }
                    }
                    {
                      \prop_if_in_p:Nn
```

```
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-IV }
                    }
                    {
                      \prop_if_in_p:Nn
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-V }
                    }
                    {
                      \prop_if_in_p:Nn
                        \g__pdfmanagement_documentproperties_prop
                        { document / testphase / phase-VI }
                    }
                }
            }
        }
      }
      {
        \RequirePackage
          { enumitem }
      }
      {
        \RequirePackage
          { paralist }
      }
  }
\ExplSyntaxOff
```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_latex_fancy_list_item_label_number:nn
  {
    \str_case:nn
      { #1 }
      {
        { Decimal } { #2 }
        { LowerRoman } { \int_to_roman:n { #2 } }
        { UpperRoman } { \int_to_Roman:n { #2 } }
        { LowerAlpha } { \int_to_alph:n { #2 } }
        { UpperAlpha } { \int_to_Alph:n { #2 } }
      }
  }
\cs_new:Nn
  \@@_latex_fancy_list_item_label_delimiter:n
  {
```

```
    \str_case:nn
      { #1 }
      {
        { Default } { . }
        { OneParen } { ) }
        { Period } { . }
      }
  }
\cs_new:Nn
  \@@_latex_fancy_list_item_label:nnn
  {
    \@@_latex_fancy_list_item_label_number:nn
      { #1 }
      { #3 }
    \@@_latex_fancy_list_item_label_delimiter:n
      { #2 }
  }
\cs_generate_variant:Nn
  \@@_latex_fancy_list_item_label:nnn
  { VVn }
\tl_new:N
  \l_@@_latex_fancy_list_item_label_number_style_tl
\tl_new:N
  \l_@@_latex_fancy_list_item_label_delimiter_style_tl
\@ifpackageloaded { enumitem } {
  \markdownSetup { rendererPrototypes = {
```

First, let's define the tight list item renderer prototypes.

```
    ulBeginTight = {
      \begin
        { itemize }
        [ noitemsep ]
    },
    ulEndTight = {
      \end
        { itemize }
    },
    olBeginTight = {
      \begin
        { enumerate }
        [ noitemsep ]
    },
    olEndTight = {
      \end
        { enumerate }
    },
    dlBeginTight = {
      \begin
```

```
      { description }
      [ noitemsep ]
  },
  dlEndTight = {
    \end
      { description }
  },
```

Second, let's define the fancy list item renderer prototypes.

```
  fancyOlBegin = {
    \group_begin:
    \tl_set:Nn
      \l_@@_latex_fancy_list_item_label_number_style_tl
      { #1 }
    \tl_set:Nn
      \l_@@_latex_fancy_list_item_label_delimiter_style_tl
      { #2 }
    \begin
      { enumerate }
  },
  fancyOlBeginTight = {
    \group_begin:
    \tl_set:Nn
      \l_@@_latex_fancy_list_item_label_number_style_tl
      { #1 }
    \tl_set:Nn
      \l_@@_latex_fancy_list_item_label_delimiter_style_tl
      { #2 }
    \begin
      { enumerate }
      [ noitemsep ]
  },
  fancyOlEnd(|Tight) = {
    \end { enumerate }
    \group_end:
  },
  fancyOlItemWithNumber = {
    \item
      [
        \@@_latex_fancy_list_item_label:VVn
          \l_@@_latex_fancy_list_item_label_number_style_tl
          \l_@@_latex_fancy_list_item_label_delimiter_style_tl
          { #1 }
      ]
  },
} }
```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
}
{ \@ifpackageloaded { paralist } {
  \markdownSetup { rendererPrototypes = {
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```
    ulBeginTight = {
      \group_begin:
      \pltopsep=\topsep
      \plpartopsep=\partopsep
      \begin { compactitem }
    },
    ulEndTight = {
      \end { compactitem }
      \group_end:
    },
    fancyOlBegin = {
      \group_begin:
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \begin { enumerate }
    },
    fancyOlEnd = {
      \end { enumerate }
      \group_end:
    },
```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```
    olBeginTight = {
      \group_begin:
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin { compactenum }
    },
    olEndTight = {
      \end { compactenum }
      \group_end:
    },
    fancyOlBeginTight = {
      \group_begin:
```

```
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin { compactenum }
    },
    fancyOlEndTight = {
      \end { compactenum }
      \group_end:
    },
    fancyOlItemWithNumber = {
      \item
        [
          \@@_latex_fancy_list_item_label:VVn
            \l_@@_latex_fancy_list_item_label_number_style_tl
            \l_@@_latex_fancy_list_item_label_delimiter_style_tl
            { #1 }
        ]
    },
```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```
    dlBeginTight = {
      \group_begin:
      \plpartopsep=\partopsep
      \pltopsep=\topsep
      \begin { compactdesc }
    },
    dlEndTight = {
      \end { compactdesc }
      \group_end:
    }
  } }
}
{
```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
  \markdownSetup
    {
      rendererPrototypes = {
        ulBeginTight = \markdownRendererUlBegin,
        ulEndTight = \markdownRendererUlEnd,
```

```
        fancyOlBegin = \markdownRendererOlBegin,
        fancyOlEnd = \markdownRendererOlEnd,
        olBeginTight = \markdownRendererOlBegin,
        olEndTight = \markdownRendererOlEnd,
        fancyOlBeginTight = \markdownRendererOlBegin,
        fancyOlEndTight = \markdownRendererOlEnd,
        dlBeginTight = \markdownRendererDlBegin,
        dlEndTight = \markdownRendererDlEnd,
      },
    }
} }
\ExplSyntaxOff
\RequirePackage{amsmath}
```

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

```
\@ifpackageloaded{unicode-math}{
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\mdlgwhtsquare$},
  }}
}{
  \RequirePackage{amssymb}
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\square$},
  }}
}
\RequirePackage{csvsimple}
\RequirePackage{fancyvrb}
\RequirePackage{graphicx}
\markdownSetup{rendererPrototypes={
  hardLineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
```

We can capitalize on the fact that the expansion of renderers is performed by TEX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,[34] we can reliably detect math mode inside the renderer.

[34]This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```
  codeSpan = {%
    \ifmmode
      \text{#1}%
    \else
      \texttt{#1}%
    \fi
  }}}
```

3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package csvsimple when the raw attribute is `csv`, display the content using the default templates of the package luaxml when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```
\ExplSyntaxOn
\markdownSetup{
  rendererPrototypes = {
    contentBlock = {
      \str_case:nnF
        { #1 }
        {
          { csv }
            {
              \begin { table }
                \begin { center }
                  \csvautotabular { #3 }
                \end{ center }
                \tl_if_empty:nF
                  { #4 }
                  { \caption { #4 } }
              \end { table }
            }
          { html }
            {
```

If we are using TEX4ht[35], we will pass HTML elements to the output HTML document unchanged.

```
              \cs_if_exist:NTF
                \HCode
                {
                  \if_mode_vertical:
                    \IgnorePar
```

[35] See https://tug.org/tex4ht/.

```
                \fi:
                \EndP
                \special
                  { t4ht* < #3 }
                \par
                \ShowPar
              }
              {
                \@@_luaxml_print_html:n
                  { #3 }
              }
            }
          { tex }
            {
              \markdownEscape
                { #3 }
            }
        }
        {
          \markdownInput
            { #3 }
        }
    },
  },
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  ulBegin = {\begin{itemize}},
  ulEnd = {\end{itemize}},
  olBegin = {\begin{enumerate}},
  olItem = {\item{}},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  dlBegin = {\begin{description}},
  dlItem = {\item[#1]},
  dlEnd = {\end{description}},
  emphasis = {\emph{#1}},
  tickedBox = {$\boxtimes$},
  halfTickedBox = {$\boxdot$}}}
```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```
\ExplSyntaxOn
\seq_new:N
  \g_@@_header_identifiers_seq
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin = {
```

```
        \markdownSetup
          {
            rendererPrototypes = {
              attributeIdentifier = {
                \seq_gput_right:Nn
                  \g_@@_header_identifiers_seq
                  { ##1 }
              },
            },
          }
      },
      headerAttributeContextEnd = {
        \seq_map_inline:Nn
          \g_@@_header_identifiers_seq
          { \label { ##1 } }
        \seq_gclear:N
          \g_@@_header_identifiers_seq
      },
    },
  }
```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```
\bool_new:N
  \l_@@_header_unnumbered_bool
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin += {
        \markdownSetup
          {
            rendererPrototypes = {
              attributeClassName = {
                \bool_if:nT
                  {
                    \str_if_eq_p:nn
                      { ##1 }
                      { unnumbered } &&
                    ! \l_@@_header_unnumbered_bool
                  }
                  {
                    \group_begin:
                    \bool_set_true:N
                      \l_@@_header_unnumbered_bool
                    \c@secnumdepth = 0
                    \markdownSetup
                      {
```

```
                        rendererPrototypes = {
                          sectionBegin = {
                            \group_begin:
                          },
                          sectionEnd = {
                            \group_end:
                          },
                        },
                      }
                    }
                  },
                },
              }
          },
      },
    }
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  superscript = {\textsuperscript{#1}},
  subscript = {\textsubscript{#1}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  note = {\footnote{#1}}}}
```

3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\RequirePackage{ltxcmds}
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInputFencedCodePrototype#1#2#3
  {
    \tl_if_empty:nTF
      { #2 }
      { \markdownRendererInputVerbatim{#1} }
```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written.

```
      {
        \regex_extract_once:nnN
          { \w* }
          { #2 }
          \l_tmpa_seq
        \seq_pop_left:NN
          \l_tmpa_seq
          \l_tmpa_tl
```

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

```
        \ltx@ifpackageloaded
          { minted }
          {
            \catcode`\%=14\relax
            \catcode`\#=6\relax
            \exp_args:NV
              \inputminted
              \l_tmpa_tl
              { #1 }
            \catcode`\%=12\relax
            \catcode`\#=12\relax
          }
          {
```

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

```
            \ltx@ifpackageloaded
              { listings }
              { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
              { \markdownRendererInputFencedCode { #1 } { } { } }
          }
      }
  }
\ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
\ExplSyntaxOn
\def\markdownLATEXStrongEmphasis#1{
  \str_if_in:NnTF
    \f@series
    { b }
    { \textnormal{#1} }
    { \textbf{#1} }
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={strongEmphasis={%
  \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
\@ifundefined{chapter}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}},
```

```
    headingFive = {\subparagraph{#1}}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
}%
```

3.3.4.4 Tickboxes

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

```
\markdownSetup{
  rendererPrototypes = {
    ulItem = {%
      \futurelet\markdownLaTeXCheckbox\markdownLaTeXUlItem
    },
  },
}
\def\markdownLaTeXUlItem{%
  \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
    \item[\markdownLaTeXCheckbox]%
    \expandafter\@gobble
  \else
    \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
      \item[\markdownLaTeXCheckbox]%
      \expandafter\expandafter\expandafter\@gobble
    \else
      \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
        \item[\markdownLaTeXCheckbox]%
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\@gobble
      \else
        \item{}%
      \fi
    \fi
  \fi
}
```

3.3.4.5 HTML elements

If the `html` option is enabled and we are using TEX4ht[36], we will pass HTML elements to the output HTML document unchanged.

[36] See https://tug.org/tex4ht/.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

3.3.4.6 Citations

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

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\long\def\@gobblethree#1#2#3{}%
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
        \relax
        \cite{#1#2#6}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobblethree
      \fi
    \else%  Before a postnote (#5), dump the accumulator
      \ifx\relax#1\relax\else
        \cite{#1}%
```

```
      \fi
      \cite[#5]{#6}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobblethree
    \fi
  \else%  Before a prenote (#4), dump the accumulator
    \ifx\relax#1\relax\else
      \cite{#1}%
    \fi
    \ifnum\markdownLaTeXCitationsCounter>1\relax
      \space  % Insert a space before the prenote in later citations
    \fi
    #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
    \relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXBasicCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobblethree
  \fi\markdownLaTeXBasicCitations{#1#2#6},}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% Natbib implementation
```

```
\def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
        \citep{#1,#5}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  Before a postnote (#4), dump the accumulator
      \ifx\relax#1\relax\else
        \citep{#1}%
      \fi
      \citep[][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  Before a prenote (#3), dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citep{#1}%
    \fi
    \citep[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
    \relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
```

```
    \@gobbletwo
  \fi\markdownLaTeXNatbibCitations{#1,#5}}
\def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
        \citet{#1,#5}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  After a prenote or a postnote, dump the accumulator
      \ifx\relax#1\relax\else
        \citet{#1}%
      \fi
      , \citet[#3][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
      \relax
        ,
      \else
        \ifnum
        \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
        \relax
          ,
        \fi
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibTextCitations
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  After a prenote or a postnote, dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citet{#1}%
    \fi
    , \citet[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
    \relax
      ,
    \else
```

```
      \ifnum
      \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
      \relax
        ,
      \fi
    \fi
    \expandafter\expandafter\expandafter
    \markdownLaTeXNatbibTextCitations
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibTextCitations{#1,#5}}

% BibLaTeX implementation
\def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
  \relax
    \autocites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
\def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
  \relax
    \textcites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}

\markdownSetup{rendererPrototypes = {
  cite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
```

```
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    }},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    }}}}
```

3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```
\RequirePackage{url}
\RequirePackage{expl3}
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererLinkPrototype
  #1#2#3#4
  {
    \tl_set:Nn \l_tmpa_tl { #1 }
    \tl_set:Nn \l_tmpb_tl { #2 }
    \bool_set:Nn
      \l_tmpa_bool
      {
        \tl_if_eq_p:NN
          \l_tmpa_tl
          \l_tmpb_tl
```

```
      }
    \tl_set:Nn \l_tmpa_tl { #4 }
    \bool_set:Nn
      \l_tmpb_bool
      {
        \tl_if_empty_p:N
          \l_tmpa_tl
      }
```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```
    \bool_if:nTF
      {
        \l_tmpa_bool && \l_tmpb_bool
      }
      {
        \markdownLaTeXRendererAutolink { #2 } { #3 }
      }
      {
        \markdownLaTeXRendererDirectOrIndirectLink
          { #1 } { #2 } { #3 } { #4 }
      }
  }
\def\markdownLaTeXRendererAutolink#1#2{
```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```
  \tl_set:Nn
    \l_tmpa_tl
    { #2 }
  \tl_trim_spaces:N
    \l_tmpa_tl
  \tl_set:Nx
    \l_tmpb_tl
    {
      \tl_range:Nnn
        \l_tmpa_tl
        { 1 }
        { 1 }
    }
  \str_if_eq:NNTF
    \l_tmpb_tl
    \c_hash_str
    {
      \tl_set:Nx
        \l_tmpb_tl
        {
```

```
          \tl_range:Nnn
            \l_tmpa_tl
            { 2 }
            { -1 }
        }
      \exp_args:NV
        \ref
        \l_tmpb_tl
    }
    {
      \url { #2 }
    }
}
\ExplSyntaxOff
\def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
  #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

3.3.4.8 Tables

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

```
\newcount\markdownLaTeXRowCounter
\newcount\markdownLaTeXRowTotal
\newcount\markdownLaTeXColumnCounter
\newcount\markdownLaTeXColumnTotal
\newtoks\markdownLaTeXTable
\newtoks\markdownLaTeXTableAlignment
\newtoks\markdownLaTeXTableEnd
\AtBeginDocument{%
  \@ifpackageloaded{booktabs}{%
    \def\markdownLaTeXTopRule{\toprule}%
    \def\markdownLaTeXMidRule{\midrule}%
    \def\markdownLaTeXBottomRule{\bottomrule}%
  }{%
    \def\markdownLaTeXTopRule{\hline}%
    \def\markdownLaTeXMidRule{\hline}%
    \def\markdownLaTeXBottomRule{\hline}%
  }%
}
\markdownSetup{rendererPrototypes={
  table = {%
    \markdownLaTeXTable={}%
    \markdownLaTeXTableAlignment={}%
    \markdownLaTeXTableEnd={%
      \markdownLaTeXBottomRule
      \end{tabular}}%
    \ifx\empty#1\empty\else
```

```
      \addto@hook\markdownLaTeXTable{%
        \begin{table}
        \centering}%
      \addto@hook\markdownLaTeXTableEnd{%
        \caption{#1}}%
    \fi
  }
}}
```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as LaTeX labels for referencing tables.

```
\ExplSyntaxOn
\seq_new:N
  \l_@@_table_identifiers_seq
\markdownSetup {
  rendererPrototypes = {
    table += {
      \seq_map_inline:Nn
        \l_@@_table_identifiers_seq
        {
          \addto@hook
            \markdownLaTeXTableEnd
            { \label { ##1 } }
        }
    },
  }
}
\markdownSetup {
  rendererPrototypes = {
    tableAttributeContextBegin = {
      \group_begin:
      \markdownSetup {
        rendererPrototypes = {
          attributeIdentifier = {
            \seq_put_right:Nn
              \l_@@_table_identifiers_seq
              { ##1 }
          },
        },
      }
    },
    tableAttributeContextEnd = {
      \group_end:
    },
  },
}
\ExplSyntaxOff
```

```
\markdownSetup{rendererPrototypes={
  table += {%
    \ifx\empty#1\empty\else
      \addto@hook\markdownLaTeXTableEnd{%
        \end{table}}%
    \fi
    \addto@hook\markdownLaTeXTable{\begin{tabular}}%
    \markdownLaTeXRowCounter=0%
    \markdownLaTeXRowTotal=#2%
    \markdownLaTeXColumnTotal=#3%
    \markdownLaTeXRenderTableRow
  }
}}
\def\markdownLaTeXRenderTableRow#1{%
  \markdownLaTeXColumnCounter=0%
  \ifnum\markdownLaTeXRowCounter=0\relax
    \markdownLaTeXReadAlignments#1%
    \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
        \the\markdownLaTeXTableAlignment}}%
    \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
  \else
    \markdownLaTeXRenderTableCell#1%
  \fi
  \ifnum\markdownLaTeXRowCounter=1\relax
    \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
  \fi
  \advance\markdownLaTeXRowCounter by 1\relax
  \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
    \the\markdownLaTeXTable
    \the\markdownLaTeXTableEnd
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableRow}
\def\markdownLaTeXReadAlignments#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \if#1d%
    \addto@hook\markdownLaTeXTableAlignment{l}%
  \else
    \addto@hook\markdownLaTeXTableAlignment{#1}%
  \fi
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
    \expandafter\@gobble
  \fi\markdownLaTeXReadAlignments}
\def\markdownLaTeXRenderTableCell#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
    \addto@hook\markdownLaTeXTable{#1&}%
```

```
  \else
    \addto@hook\markdownLaTeXTable{#1\\}%
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableCell}
```

3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

\markdownIfOption{lineBlocks}{%
  \RequirePackage{verse}
  \markdownSetup{rendererPrototypes={
    lineBlockBegin = {%
      \begingroup
        \def\markdownRendererHardLineBreak{\\}%
        \begin{verse}%
    },
    lineBlockEnd = {%
        \end{verse}%
      \endgroup
    },
  }}
}{}

```

3.3.4.10 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown / jekyllData }
  {
    author .code:n = {
      \author
        { #1 }
    },
    date .code:n = {
      \date
        { #1 }
    },
    title .code:n = {
      \title
        { #1 }
```

To complement the default setup of our key–values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```
      \AddToHook
        { begindocument / end }
        { \maketitle }
    },
  }
```

3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the soul package or the lua-ul package and use it to implement marked text.

```
\@@_if_option:nT
  { mark }
  {
    \sys_if_engine_luatex:TF
      {
        \RequirePackage
          { luacolor }
        \RequirePackage
          { lua-ul }
        \markdownSetup
          {
            rendererPrototypes = {
              mark = {
                \highLight
                  { #1 }
              },
            }
          }
      }
      {
        \RequirePackage
          { xcolor }
        \RequirePackage
          { soul }
        \markdownSetup
          {
            rendererPrototypes = {
              mark = {
                \hl
                  { #1 }
              },
            }
          }
```

```
      }
  }
```

3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the soul package or the lua-ul package and use it to implement strike-throughs.

```
\@@_if_option:nT
  { strikeThrough }
  {
    \sys_if_engine_luatex:TF
      {
        \RequirePackage
          { lua-ul }
        \markdownSetup
          {
            rendererPrototypes = {
              strikeThrough = {
                \strikeThrough
                  { #1 }
              },
            }
          }
      }
      {
        \RequirePackage
          { soul }
        \markdownSetup
          {
            rendererPrototypes = {
              strikeThrough = {
                \st
                  { #1 }
              },
            }
          }
      }
  }
```

3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form ⟨*key*⟩=⟨*value*⟩ set the corresponding keys of the graphicx package to the corresponding

values and we will register any identifiers, so that they can be used as LaTeX labels for referencing figures.

```
\seq_new:N
  \l_@@_image_identifiers_seq
\markdownSetup {
  rendererPrototypes = {
    image = {
      \tl_if_empty:nTF
        { #4 }
        {
          \begin { center }
            \includegraphics
              [ alt = { #1 } ]
              { #3 }
          \end { center }
        }
        {
          \begin { figure }
            \begin { center }
              \includegraphics
                [ alt = { #1 } ]
                { #3 }
              \caption { #4 }
              \seq_map_inline:Nn
                \l_@@_image_identifiers_seq
                { \label { ##1 } }
            \end { center }
          \end { figure }
        }
    },
  }
}
\@@_if_option:nT
  { linkAttributes }
  {
    \RequirePackage { graphicx }
  }
\markdownSetup {
  rendererPrototypes = {
    imageAttributeContextBegin = {
      \group_begin:
      \markdownSetup {
        rendererPrototypes = {
          attributeIdentifier = {
            \seq_put_right:Nn
              \l_@@_image_identifiers_seq
              { ##1 }
```

```
          },
          attributeKeyValue = {
            \setkeys
              { Gin }
              { { ##1 } = { ##2 } }
          },
        },
      }
    },
    imageAttributeContextEnd = {
      \group_end:
    },
  },
}
\ExplSyntaxOff
```

3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package luaxml when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_luaxml_print_html:n
  {
    \luabridge_now:n
      {
        local~input_file = assert(io.open(" #1 ", "r"))
        local~input = assert(input_file:read("*a"))
        assert(input_file:close())
        input = "<body>" .. input .. "</body>"
        local~dom = require("luaxml-domobject").html_parse(input)
        local~output = require("luaxml-htmltemplates"):process_dom(dom)
        print(output)
      }
  }
\cs_gset_protected:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline:nn
              { #1 }
```

```
            { tex }
        }
      { html }
        {
```

If we are using TEX4ht[37], we will pass HTML elements to the output HTML document unchanged.

```
          \cs_if_exist:NTF
            \HCode
            {
              \if_mode_vertical:
                \IgnorePar
                \EndP
              \fi:
              \special
                { t4ht* < #1 }
            }
            {
              \@@_luaxml_print_html:n
                { #1 }
            }
        }
    }
    {
      \@@_plain_tex_default_input_raw_inline:nn
        { #1 }
        { #2 }
    }
}
\cs_gset_protected:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block:nn
              { #1 }
              { tex }
          }
        { html }
          {
```

[37] See https://tug.org/tex4ht/.

If we are using TEX4ht[38], we will pass HTML elements to the output HTML document unchanged.

```
            \cs_if_exist:NTF
              \HCode
              {
                \if_mode_vertical:
                  \IgnorePar
                \fi:
                \EndP
                \special
                  { t4ht* < #1 }
                \par
                \ShowPar
              }
              {
                \@@_luaxml_print_html:n
                  { #1 }
              }
          }
        }
        {
          \@@_plain_tex_default_input_raw_block:nn
            { #1 }
            { #2 }
        }
    }
```

3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as LATEX labels for referencing the last LATEX counter that has been incremented in e.g. ordered lists.

```
\seq_new:N
  \l_@@_bracketed_span_identifiers_seq
\markdownSetup {
  rendererPrototypes = {
    bracketedSpanAttributeContextBegin = {
      \group_begin:
      \markdownSetup {
        rendererPrototypes = {
          attributeIdentifier = {
            \seq_put_right:Nn
              \l_@@_bracketed_span_identifiers_seq
              { ##1 }
          },
```

[38] See https://tug.org/tex4ht/.

```
        },
      }
    },
    bracketedSpanAttributeContextEnd = {
      \seq_map_inline:Nn
        \l_@@_bracketed_span_identifiers_seq
        { \label { ##1 } }
      \group_end:
    },
  },
}
\ExplSyntaxOff
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
```

3.3.5 Miscellanea

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

```
\newcommand\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat}%
```

3.4 ConTEXt Implementation

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

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

```
\def\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \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.

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

3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConTEXt interface (see Section 2.4.2).

```
\long\def\inputmarkdown{%
  \dosingleempty
  \doinputmarkdown}%
\long\def\doinputmarkdown[#1]#2{%
  \begingroup
    \iffirstargument
      \setupmarkdown[#1]%
    \fi
    \markdownInput{#2}%
  \endgroup}%
\long\def\inputyaml{%
  \dosingleempty
  \doinputyaml}%
\long\def\doinputyaml[#1]#2{%
  \doinputmarkdown
    [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%
```

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

In Knuth's TEX, trailing spaces are removed very early on when a line is being put to the input buffer. [18, sec. 31]. According to Eijkhout [19, sec. 2.2], this is because "these spaces are hard to see in an editor". At the moment, there is no option to suppress this behavior in (Lua)TEX, but ConTEXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTEXt MkIV and therefore to insert hard line breaks into markdown text.

```
\startluacode
  document.markdown_buffering = false
  local function preserve_trailing_spaces(line)
    if document.markdown_buffering then
      line = line:gsub("[ \t][ \t]$", "\t\t")
    end
    return line
  end
  resolvers.installinputlinehandler(preserve_trailing_spaces)
\stopluacode
\begingroup
  \catcode`\|=0%
  \catcode`\\=12%
```

```
  |gdef|startmarkdown{%
    |ctxlua{document.markdown_buffering = true}%
    |markdownReadAndConvert{\stopmarkdown}%
                           {|stopmarkdown}}%
  |gdef|stopmarkdown{%
    |ctxlua{document.markdown_buffering = false}%
    |markdownEnd}%
  |gdef|startyaml{%
    |begingroup
    |ctxlua{document.markdown_buffering = true}%
    |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
    |markdownReadAndConvert{\stopyaml}%
                           {|stopyaml}}%
  |gdef|stopyaml{%
    |ctxlua{document.markdown_buffering = false}%
    |yamlEnd}%
|endgroup
```

3.4.2 Themes

This section overrides the plain TEX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConTEXt themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_context_loaded_themes_linenos_prop
\prop_new:N \g_@@_context_loaded_themes_versions_prop
\cs_gset:Nn
  \@@_load_theme:nnn
  {
```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme`⟨*munged theme name*⟩`.tex` exists. If it does, load it. Otherwise, try loading a plain TEX theme instead.

```
    \bool_if:nTF
      {
        \bool_lazy_or_p:nn
          {
            \prop_if_in_p:Nn
              \g_@@_context_built_in_themes_prop
              { #1 }
          }
          {
            \file_if_exist_p:n
              { t - markdown theme #3.tex }
          }
```

```
    }
    {
      \prop_get:NnNTF
        \g_@@_context_loaded_themes_linenos_prop
        { #1 }
        \l_tmpa_tl
        {
          \prop_get:NnN
            \g_@@_context_loaded_themes_versions_prop
            { #1 }
            \l_tmpb_tl
          \str_if_eq:nVTF
            { #2 }
            \l_tmpb_tl
            {
              \msg_warning:nnnVn
                { markdown }
                { repeatedly-loaded-context-theme }
                { #1 }
                \l_tmpa_tl
                { #2 }
            }
            {
              \msg_error:nnnnVV
                { markdown }
                { different-versions-of-context-theme }
                { #1 }
                { #2 }
                \l_tmpb_tl
                \l_tmpa_tl
            }
        }
        {
          \prop_gput:Nnx
            \g_@@_context_loaded_themes_linenos_prop
            { #1 }
            { \tex_the:D \tex_inputlineno:D }  % noqa: W200
          \prop_gput:Nnn
            \g_@@_context_loaded_themes_versions_prop
            { #1 }
            { #2 }
```

Load built-in plain TEX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```
          \prop_if_in:NnTF
            \g_@@_context_built_in_themes_prop
            { #1 }
```

```
              {
                \msg_info:nnnn
                  { markdown }
                  { loading-built-in-context-theme }
                  { #1 }
                  { #2 }
                \prop_item:Nn
                  \g_@@_context_built_in_themes_prop
                  { #1 }
              }
              {
                \msg_info:nnnn
                  { markdown }
                  { loading-context-theme }
                  { #1 }
                  { #2 }
                \usemodule
                  [ t ]
                  [ markdown theme #3 ]
              }
          }
      }
      {
        \@@_plain_tex_load_theme:nnn
          { #1 }
          { #2 }
          { #3 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-built-in-context-theme }
  { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { loading-context-theme }
  { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-context-theme }
  {
    Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\msg_new:nnn
  { markdown }
  { different-versions-of-context-theme }
```

```
  {
    Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
    but~version~#3~has~already~been~loaded~on~line~#4
  }
\ExplSyntaxOff
```

The `witiko/markdown/defaults` ConTEXt theme provides default definitions for token renderer prototypes. First, the ConTEXt theme loads the plain TEX theme with the default definitions for plain TEX:

```
\markdownLoadPlainTeXTheme
```

Next, the ConTEXt theme overrides some of the plain TEX definitions. See Section 3.4.3 for the actual definitions.

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
\markdownIfOption{plain}{\iffalse}{\iftrue}
\def\markdownRendererHardLineBreakPrototype{\blank}%
\def\markdownRendererLeftBracePrototype{\textbraceleft}%
\def\markdownRendererRightBracePrototype{\textbraceright}%
\def\markdownRendererDollarSignPrototype{\textdollar}%
\def\markdownRendererPercentSignPrototype{\percent}%
\def\markdownRendererUnderscorePrototype{\textunderscore}%
\def\markdownRendererCircumflexPrototype{\textcircumflex}%
\def\markdownRendererBackslashPrototype{\textbackslash}%
\def\markdownRendererTildePrototype{\textasciitilde}%
\def\markdownRendererPipePrototype{\char`|}%
\def\markdownRendererLinkPrototype#1#2#3#4{%
  \useURL[#1][#3][][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
  \fi\tt<\hyphenatedurl{#3}>}}%
\usemodule[database]
\defineseparatedlist
  [MarkdownConTeXtCSV]
  [separator={,},
   before=\bTABLE,after=\eTABLE,
   first=\bTR,last=\eTR,
   left=\bTD,right=\eTD]
\def\markdownConTeXtCSV{csv}
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \def\markdownConTeXtCSV@arg{#1}%
  \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
    \placetable[][tab:#1]{#4}{%
      \processseparatedfile[MarkdownConTeXtCSV][#3]}%
  \else
    \markdownInput{#3}%
```

```
  \fi}%
\def\markdownRendererImagePrototype#1#2#3#4{%
  \placefigure[][]{#4}{\externalfigure[#3]}}%
\def\markdownRendererUlBeginPrototype{\startitemize}%
\def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
\def\markdownRendererUlItemPrototype{\item}%
\def\markdownRendererUlEndPrototype{\stopitemize}%
\def\markdownRendererUlEndTightPrototype{\stopitemize}%
\def\markdownRendererOlBeginPrototype{\startitemize[n]}%
\def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
\def\markdownRendererOlItemPrototype{\item}%
\def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
\def\markdownRendererOlEndPrototype{\stopitemize}%
\def\markdownRendererOlEndTightPrototype{\stopitemize}%
\definedescription
  [MarkdownConTeXtDlItemPrototype]
  [location=hanging,
   margin=standard,
   headstyle=bold]%
\definestartstop
  [MarkdownConTeXtDlPrototype]
  [before=\blank,
   after=\blank]%
\definestartstop
  [MarkdownConTeXtDlTightPrototype]
  [before=\blank\startpacked,
   after=\stoppacked\blank]%
\def\markdownRendererDlBeginPrototype{%
  \startMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlBeginTightPrototype{%
  \startMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererDlItemPrototype#1{%
  \startMarkdownConTeXtDlItemPrototype{#1}}%
\def\markdownRendererDlItemEndPrototype{%
  \stopMarkdownConTeXtDlItemPrototype}%
\def\markdownRendererDlEndPrototype{%
  \stopMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlEndTightPrototype{%
  \stopMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererEmphasisPrototype#1{{\em#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
\def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
\def\markdownRendererLineBlockBeginPrototype{%
  \begingroup
    \def\markdownRendererHardLineBreak{
    }%
```

```
    \startlines
}%
\def\markdownRendererLineBlockEndPrototype{%
    \stoplines
  \endgroup
}%
\def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
```

3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputFencedCodePrototype#1#2#3
  {
    \tl_if_empty:nTF
      { #2 }
      { \markdownRendererInputVerbatim{#1} }
```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTEXt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```
\definetyping [latex]
\setuptyping  [latex] [option=TEX]

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

```
      {
        \regex_extract_once:nnN
          { \w* }
          { #2 }
          \l_tmpa_seq
        \seq_pop_left:NN
          \l_tmpa_seq
```

```
          \l_tmpa_tl
        \typefile[ \l_tmpa_tl ][] {#1}
      }
  }
\ExplSyntaxOff
\def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
\def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
\def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
\def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
\def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
\def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
\def\markdownRendererThematicBreakPrototype{%
  \blackrule[height=1pt, width=\hsize]}%
\def\markdownRendererNotePrototype#1{\footnote{#1}}%
\def\markdownRendererTickedBoxPrototype{$\boxtimes$}
\def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
\def\markdownRendererUntickedBoxPrototype{$\square$}
\def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
\def\markdownRendererSuperscriptPrototype#1{\high{#1}}
\def\markdownRendererSubscriptPrototype#1{\low{#1}}
\def\markdownRendererDisplayMathPrototype#1{%
  \startformula#1\stopformula}%
```

3.4.3.2 Tables

There is a basic implementation of tables.

```
\newcount\markdownConTeXtRowCounter
\newcount\markdownConTeXtRowTotal
\newcount\markdownConTeXtColumnCounter
\newcount\markdownConTeXtColumnTotal
\newtoks\markdownConTeXtTable
\newtoks\markdownConTeXtTableFloat
\def\markdownRendererTablePrototype#1#2#3{%
  \markdownConTeXtTable={}%
  \ifx\empty#1\empty
    \markdownConTeXtTableFloat={%
      \the\markdownConTeXtTable}%
  \else
    \markdownConTeXtTableFloat={%
      \placetable{#1}{\the\markdownConTeXtTable}}%
  \fi
  \begingroup
  \setupTABLE[r][each][topframe=off, bottomframe=off,
                       leftframe=off, rightframe=off]
  \setupTABLE[c][each][topframe=off, bottomframe=off,
                       leftframe=off, rightframe=off]
  \setupTABLE[r][1][topframe=on, bottomframe=on]
```

```
  \setupTABLE[r][#1][bottomframe=on]
  \markdownConTeXtRowCounter=0%
  \markdownConTeXtRowTotal=#2%
  \markdownConTeXtColumnTotal=#3%
  \markdownConTeXtRenderTableRow}
\def\markdownConTeXtRenderTableRow#1{%
  \markdownConTeXtColumnCounter=0%
  \ifnum\markdownConTeXtRowCounter=0\relax
    \markdownConTeXtReadAlignments#1%
    \markdownConTeXtTable={\bTABLE}%
  \else
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\bTR}%
    \markdownConTeXtRenderTableCell#1%
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTR}%
  \fi
  \advance\markdownConTeXtRowCounter by 1\relax
  \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTABLE}%
    \the\markdownConTeXtTableFloat
    \endgroup
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtRenderTableRow}
\def\markdownConTeXtReadAlignments#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \if#1d%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1l%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1c%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
  \fi\if#1r%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
  \fi
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
  \else
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtReadAlignments}
\def\markdownConTeXtRenderTableCell#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \markdownConTeXtTable=\expandafter{%
    \the\markdownConTeXtTable\bTD#1\eTD}%
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
  \else
    \expandafter\gobbleoneargument
```

```
\fi\markdownConTeXtRenderTableCell}
```

3.4.3.3 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline:nn
              { #1 }
              { context }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { context }
          {
            \@@_plain_tex_default_input_raw_block:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset_eq:NN
  \markdownRendererInputRawBlockPrototype
```

```
  \markdownRendererInputRawInlinePrototype
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
\ExplSyntaxOff
\stopmodule
\protect
```

At the end of the ConTEXt module, we load the `witiko/markdown/defaults` ConTEXt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
\ExplSyntaxOn
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_context_tl
  {
    \use:c
      { ExplSyntaxOff }
    \@@_if_option:nF
      { noDefaults }
      {
        \@@_if_option:nTF
          { experimental }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults@experimental }
          }
          {
            \@@_setup:n
              { theme = witiko/markdown/defaults }
          }
      }
    \use:c
      { ExplSyntaxOn }
  }
\ExplSyntaxOff
\stopmodule
\protect
```

References

[1] LuaTEX development team. *LuaTEX reference manual*. Version 1.10 (stable). July 23, 2021. URL: https://www.pragma-ade.com/general/manuals/luatex.pdf (visited on 09/30/2022).

[2] LATEX Project. *l3kernel. LATEX3 programming conventions*. Dec. 25, 2024. URL: https://ctan.org/pkg/l3kernel (visited on 01/06/2025).

[3] Frank Mittelbach, Ulrike Fischer, and LaTeX Project. *The documentmetadata-support code*. June 1, 2024. URL: https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf (visited on 10/21/2024).

[4] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[5] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[6] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[7] Bonita Sharif and Jonathan I. Maletic. "An Eye Tracking Study on camelCase and under_score Identifier Styles." In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: 10.1109/ICPC.2010.41.

[8] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[9] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/doc.pdf (visited on 02/19/2018).

[10] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf (visited on 02/11/2021).

[11] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: https://github.com/Witiko/markdown/discussions/514 (visited on 10/21/2024).

[12] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: https://tex.stackexchange.com/q/716362/70941 (visited on 04/28/2024).

[13] Vít Starý Novotný. *Routing YAML metadata to expl3 key–values*. Markdown Enhancement Proposal. Oct. 14, 2024. URL: https://github.com/witiko/markdown/discussions/517 (visited on 01/06/2025).

[14] Frank Mittelbach. *LaTeX's hook management*. June 26, 2024. URL: https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf (visited on 10/02/2024).

[15] Geoffrey M. Poore. *The minted Package. Highlighted source code in LaTeX*. July 19, 2017. URL: https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf (visited on 09/01/2020).

[16] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[17] Johannes Braams et al. *The LaTeX 2ε Sources*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/source2e.pdf (visited on 01/08/2018).

[18] Donald Ervin Knuth. *TeX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

[19] Victor Eijkhout. *TeX by Topic. A TeXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

Index

autoIdentifiers *21*, 33, 85, 101

blankBeforeBlockquote *21*
blankBeforeCodeFence *22*
blankBeforeDivFence *22*
blankBeforeHeading *22*
blankBeforeList *22*
bracketedSpans *23*, 87, 461
breakableBlockquotes *23*

cacheDir 4, 17, 19, 59, 159, 172, 373, 393, 413
citationNbsps *23*
citations *24*, 90, 91
codeSpans *24*
contentBlocks 20, *25*, 34
contentBlocksLanguageMap 20
contentLevel *25*

debugExtensions 9, 20, *26*, 317
debugExtensionsFileName 20, 26
defaultOptions *10*, 51, 372, 373
definitionLists *26*, 95

eagerCache *17*, 371
ensureJekyllData *27*
entities.char_entity *218*
entities.dec_entity *218*
entities.hex_entity *218*
entities.hex_entity_with_x_char *218*
escape_minimal *222*

escape_programmatic_text *222*
escape_typographic_text *222*
expandtabs *277*
expectJekyllData *27*, 27
experimental 5, *17*, 431
extensions 28, *167*, 322
extensions.bracketed_spans *323*
extensions.citations *324*
extensions.content_blocks *328*
extensions.definition_lists *331*
extensions.fancy_lists *333*
extensions.fenced_code *339*
extensions.fenced_divs *345*
extensions.header_attributes *349*
extensions.inline_code_attributes *351*
extensions.jekyll_data *367*
extensions.line_blocks *351*
extensions.link_attributes *353*
extensions.mark *352*
extensions.notes *354*
extensions.pipe_table *357*
extensions.raw_inline *361*
extensions.strike_through *362*
extensions.subscripts *362*
extensions.superscripts *363*
extensions.tex_math *364*

fancyLists *30*, 112–117, 431
fencedCode *30*, 39, 92, 99, 118, 385, 387
fencedCodeAttributes *31*, 85, 99, 387
fencedDiv 100
fencedDivs *31*, 41
finalizeCache 17, 20, *32*, 32, 59, 158, 371, 372
frozenCache 20, 32, 59, 75, 158, 385, 393
frozenCacheCounter 32, 372, 420, 421
frozenCacheFileName 20, 32, 59, 372

\g_markdown_diagrams_infostrings_prop *389*
gfmAutoIdentifiers 21, *32*, 85, 101

hashEnumerators *33*
headerAttributes *33*, 41, 85, 101
html *34*, 104, 444

hybrid *34*, 34, 40, 46, 48, 62, 78, 119, 159, 223, 278, 420

inlineCodeAttributes *35*, 85, 93
inlineNotes *36*
\input 55, 56
\inputmarkdown *162*, 163, 165, 463
inputTempFileName 60, 62, 414, 415, 417, 418
\inputyaml *162*, 164, 463
iterlines *277*

jekyllData 3, 27, 28, *36*, 128–131, 134

\l_file_search_path_seq 419
languages_json *328*, 328
lineBlocks *37*, 107
linkAttributes *37*, 85, 105, 109, 298, 457

mark *38*, 110, 456
\markdown 154, 155, 424
markdown *153*, 153, 154, 423
markdown* *153*, 153, 154, 158, 423
\markdownBegin *53*, 53–56, 151–153, 155, 162, 163
\markdownCleanup *413*
\markdownConvert *413*
\markdownEnd *53*, 53–56, 152–155, 162, 163
\markdownError *151*, 151
\markdownEscape *53*, 56, 421
\markdownIfOption *58*
\markdownIfSnippetExists *80*
\markdownInfo *151*, 151
\markdownInput *53*, 55, 56, 153, 156, 158, 163, 419, 422
\markdownInputFilename *412*
\markdownInputFileStream *413*
\markdownInputPlainTeX *422*
\markdownLoadPlainTeXTheme *160*, 166, 383
\markdownLuaExecute *416*, 419
\markdownLuaOptions *409*, 413
\markdownMakeOther *151*, 462
\markdownOptionFinalizeCache *59*
\markdownOptionFrozenCache *59*
\markdownOptionHybrid *62*
\markdownOptionInputTempFileName *59*
\markdownOptionNoDefaults *61*

\markdownOptionOutputDir *60*, 60, 63
\markdownOptionPlain *60*
\markdownOptionStripPercentSigns *61*
\markdownOutputFileStream *413*
\markdownPrepare *412*
\markdownPrepareInputFilename *412*
\markdownPrepareLuaOptions *409*
\markdownReadAndConvert *151*, 414, 423, 424, 463
\markdownReadAndConvertProcessLine *415*, 416
\markdownReadAndConvertStripPercentSigns *414*
\markdownReadAndConvertTab *413*
\markdownRendererAttributeClassName *85*
\markdownRendererAttributeIdentifier *85*
\markdownRendererAttributeKeyValue *85*
\markdownRendererBlockQuoteBegin *86*
\markdownRendererBlockQuoteEnd *87*
\markdownRendererBracketedSpanAttributeContextBegin *87*
\markdownRendererBracketedSpanAttributeContextEnd *87*
\markdownRendererCite *90*, 91
\markdownRendererCodeSpan *92*
\markdownRendererCodeSpanAttributeContextBegin *93*
\markdownRendererCodeSpanAttributeContextEnd *93*
\markdownRendererContentBlock *93*, 94
\markdownRendererContentBlockCode *94*
\markdownRendererContentBlockOnlineImage *94*
\markdownRendererDisplayMath *125*
\markdownRendererDlBegin *95*
\markdownRendererDlBeginTight *95*
\markdownRendererDlDefinitionBegin *96*
\markdownRendererDlDefinitionEnd *97*
\markdownRendererDlEnd *97*
\markdownRendererDlEndTight *98*
\markdownRendererDlItem *96*
\markdownRendererDlItemEnd *96*
\markdownRendererDocumentBegin *110*
\markdownRendererDocumentEnd *110*
\markdownRendererEllipsis 41, *98*
\markdownRendererEmphasis *98*, 138
\markdownRendererError *127*
\markdownRendererFancyOlBegin *112*, 113
\markdownRendererFancyOlBeginTight *113*
\markdownRendererFancyOlEnd *117*

\markdownRendererFancyOlEndTight *117*
\markdownRendererFancyOlItem *115*
\markdownRendererFancyOlItemEnd *115*
\markdownRendererFancyOlItemWithNumber *115*
\markdownRendererFencedCodeAttributeContextBegin *99*
\markdownRendererFencedCodeAttributeContextEnd *99*
\markdownRendererFencedDivAttributeContextBegin *100*
\markdownRendererFencedDivAttributeContextEnd *100*
\markdownRendererHalfTickedBox *126*
\markdownRendererHardLineBreak *108*
\markdownRendererHeaderAttributeContextBegin *101*
\markdownRendererHeaderAttributeContextEnd *101*
\markdownRendererHeadingFive *103*
\markdownRendererHeadingFour *103*
\markdownRendererHeadingOne *101*
\markdownRendererHeadingSix *103*
\markdownRendererHeadingThree *102*
\markdownRendererHeadingTwo *102*
\markdownRendererImage *105*
\markdownRendererImageAttributeContextBegin *105*
\markdownRendererImageAttributeContextEnd *105*
\markdownRendererInlineHtmlComment *104*
\markdownRendererInlineHtmlTag *104*
\markdownRendererInlineMath *125*
\markdownRendererInputBlockHtmlElement *104*
\markdownRendererInputFencedCode *92*
\markdownRendererInputRawBlock *118*
\markdownRendererInputRawInline *117*
\markdownRendererInputVerbatim *91*
\markdownRendererInterblockSeparator *106*
\markdownRendererJekyllDataBegin *128*
\markdownRendererJekyllDataBoolean *130*
\markdownRendererJekyllDataEmpty *134*
\markdownRendererJekyllDataEnd *128*
\markdownRendererJekyllDataMappingBegin *129*
\markdownRendererJekyllDataMappingEnd *129*
\markdownRendererJekyllDataNumber *131*
\markdownRendererJekyllDataProgrammaticString *131*, 131, 132
\markdownRendererJekyllDataSequenceBegin *130*
\markdownRendererJekyllDataSequenceEnd *130*
\markdownRendererJekyllDataString 132, 136
\markdownRendererJekyllDataStringPrototype 147

\markdownRendererJekyllDataTypographicString *131*, 131, 132, 368
\markdownRendererLineBlockBegin *107*
\markdownRendererLineBlockEnd *107*
\markdownRendererLink *108*, 138
\markdownRendererLinkAttributeContextBegin *109*
\markdownRendererLinkAttributeContextEnd *109*
\markdownRendererMark *110*
\markdownRendererNbsp *111*
\markdownRendererNote *111*
\markdownRendererOlBegin *112*
\markdownRendererOlBeginTight *112*
\markdownRendererOlEnd *116*
\markdownRendererOlEndTight *116*
\markdownRendererOlItem 42, *113*
\markdownRendererOlItemEnd *114*
\markdownRendererOlItemWithNumber 42, *114*
\markdownRendererParagraphSeparator *107*
\markdownRendererReplacementCharacter *119*
\markdownRendererSectionBegin *118*
\markdownRendererSectionEnd *118*
\markdownRendererSoftLineBreak *108*
\markdownRendererStrikeThrough *122*
\markdownRendererStrongEmphasis *99*
\markdownRendererSubscript *123*
\markdownRendererSuperscript *123*
\markdownRendererTable *124*
\markdownRendererTableAttributeContextBegin *124*
\markdownRendererTableAttributeContextEnd *124*
\markdownRendererTextCite *91*
\markdownRendererThematicBreak *126*
\markdownRendererTickedBox *126*
\markdownRendererUlBegin *88*
\markdownRendererUlBeginTight *88*
\markdownRendererUlEnd *90*
\markdownRendererUlEndTight *90*
\markdownRendererUlItem *89*
\markdownRendererUlItemEnd *89*
\markdownRendererUntickedBox *126*
\markdownRendererWarning *127*
\markdownSetup *57*, 58, 62, 157, 158, 165, 424, 430
\markdownSetupSnippet *79*, 79
\markdownThemeVersion *69*, 69

\markdownWarning *151*, 151
\markinline *53*, 55, 153, 156, 417, 422
\markinlinePlainTeX *422*

new *7*, 18, 371, 373
notes *38*, 111

parsers *238*, 276, 277
parsers.commented_line *257*
parsers.punctuation *239*
pipeTables 7, *39*, 45, 124
preserveTabs *39*, 43, 277

rawAttribute 35, *39*, 40, 117, 118
reader *8*, 29, 167, 238, 276, 323
reader->add_special_character *8*, 9, 29, 317
reader->auto_link_email *306*
reader->auto_link_url *306*
reader->create_parser *277*
reader->finalize_grammar *312*, 378
reader->initialize_named_group *317*
reader->insert_pattern *8*, 9, 29, 313, 319
reader->lookup_note_reference *291*
reader->lookup_reference *291*
reader->normalize_tag *277*
reader->options *276*
reader->parser_functions *277*
reader->parser_functions.name *277*
reader->parsers *276*, 276, 277
reader->register_link *291*
reader->update_rule 313, 316, 319
reader->writer *276*
reader.new *276*, 276, 378
relativeReferences *40*

\setupmarkdown *165*, 165
\setupyaml *165*
shiftHeadings 7, 41
singletonCache *18*
slice 7, 41, 219, 231, 232
smartEllipses *41*, 98, 159
\startmarkdown *162*, 162, 463
startNumber *42*, 113–115

\startyaml *162*, 162, 163, 463
\stopmarkdown *162*, 162, 463
\stopyaml *162*, 162, 163, 463
strikeThrough *42*, 122, 457
stripIndent *43*, 278
stripPercentSigns 414
subscripts *43*, 123
superscripts *44*, 123
syntax *314*, 318

tableAttributes *44*, 124, 453
tableCaptions 7, 44, *45*, 124
taskLists *45*, 126, 444
texComments *46*, 278
texMathDollars 35, *46*, 125
texMathDoubleBackslash 35, *47*, 125
texMathSingleBackslash 35, *47*, 125
tightLists *47*, 88, 90, 95, 98, 112, 113, 116, 117, 431

underscores *48*
unicodeNormalization *18*, 19
unicodeNormalizationForm 18, *19*
util.cache *167*, 168
util.cache_verbatim *168*
util.encode_json_string *168*
util.err *167*
util.escaper *170*
util.expand_tabs_in_line *168*
util.flatten *169*
util.intersperse *170*
util.map *170*
util.pathname *171*
util.rope_last *170*
util.rope_to_string *169*
util.salt *172*
util.table_copy *168*
util.walk *168*, 169, 170
util.warning *172*

walkable_syntax *8*, 20, 26, 312, 313, 316–318
writer *167*, 167, 219, 323
writer->active_attributes *230*, 230–232
writer->attribute_type_levels *230*

writer->attributes 228
writer->block_html_element 226
writer->blockquote 227
writer->bulletitem 225
writer->bulletlist 225
writer->citations 324
writer->code 223
writer->contentblock 329
writer->defer_call 238, 238
writer->definitionlist 331
writer->display_math 364
writer->div_begin 345
writer->div_end 345
writer->document 227
writer->ellipsis 221
writer->emphasis 227
writer->error 223
writer->escape 222
writer->escaped_chars 222, 222
writer->escaped_minimal_strings 221, 222
writer->escaped_strings 222
writer->escaped_uri_chars 221, 222
writer->fancyitem 335
writer->fancylist 334
writer->fencedCode 340
writer->flatten_inlines 219, 219
writer->get_state 237
writer->hard_line_break 221
writer->heading 235
writer->identifier 223
writer->image 224
writer->infostring 223
writer->inline_html_comment 226
writer->inline_html_tag 226
writer->inline_math 364
writer->interblocksep 220
writer->is_writing 219, 219
writer->jekyllData 368
writer->lineblock 351
writer->link 224
writer->mark 352
writer->math 223

writer->nbsp *220*
writer->note *355*
writer->options *219*
writer->ordereditem *226*
writer->orderedlist *225*
writer->paragraph *220*
writer->paragraphsep *220*
writer->plain *220*
writer->pop_attributes *231*, 231, 232
writer->push_attributes *231*, 231, 232
writer->rawBlock *340*
writer->rawInline *361*
writer->set_state *237*
writer->slice_begin *219*
writer->slice_end *219*
writer->soft_line_break *221*
writer->space *220*
writer->span *323*
writer->strike_through *362*
writer->string *223*
writer->strong *227*
writer->subscript *362*
writer->superscript *363*
writer->table *358*
writer->thematic_break *221*
writer->tickbox *227*
writer->undosep *221*, 321
writer->uri *223*
writer->verbatim *227*
writer->warning 172, *223*
writer.new *219*, 219, 378

\yaml 155
yaml *153*, 155, 423
\yamlBegin *53*, 54, 151, 152, 155, 162
\yamlEnd 53, 54, 152, 155, 162
\yamlInput *53*, 56, 153, 156, 164, 422
\yamlSetup *58*