# A Markdown Interpreter for TEX

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

**Version 3.16.4-0-g50ab7075**
**2026-11-27**

## Contents

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

**2 Interfaces** **7**
2.1 Lua Interface . . . . . 8
2.2 Plain TEX Interface . . . . . 54
2.3 LATEX Interface . . . . . 163
2.4 ConTEXt Interface . . . . . 171

**3 Implementation** **176**
3.1 Lua Implementation . . . . 176
3.2 Plain TEX Implementation . 411
3.3 LATEX Implementation . . . 457
3.4 ConTEXt Implementation . 503

**References** **514**

**Index** **515**

## List of Figures

1 A block diagram of the Markdown package . . . . . 7
2 A sequence diagram of typesetting a document using the TEX interface . . . . . 50
3 A sequence diagram of typesetting a document using the Lua CLI . . . . . 51
4 An example directed graph . . . . . 80
5 An example mindmap . . . . . 81
6 An example UML sequence diagram . . . . . 82
7 The banner of the Markdown package . . . . . 83
8 A pushdown automaton that recognizes TEX comments . . . . . 292

## 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. Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation

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

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",
                 "2016-2026 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")
```

**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).

```
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.

Only load the package outside the ConTEXt format, where we can use the resolvers API [1, Section 11.5].

```
if resolvers == nil then
  (function()
```

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

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)()
end
```

All the abovelisted modules are statically linked into the current version of the LuaTEX engine [2, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

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

```
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 [3] from the LATEX3 kernel. 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
```

A fairly recent version of expl3 is required: either a post-release updated TEX Live ⩾ 2025 installation or a TEX Live ⩾ 2026 release-date ISO image:

```
\NeedsTeXFormat{LaTeX2e}[2025-05-14]
\RequirePackage{expl3}[2025-05-14]
```

**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.

Note that this support for TEX engines other than LuaTEX comes with some limitations with respect to file and directory names. Specifically, the filenames of

your .tex files may not contain spaces[4]. If `-output-directory` is provided, it may not contain spaces either.

```
34 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 $\geqslant$ 2008).

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [2, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [2, 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$_\varepsilon$ format is loaded, a TeX engine that extends $\varepsilon$-TeX, and all the plain TeX prerequisites (see Section 1.1.2).

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.5) 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.

```
35 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.

```
36 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 tagging

[4] See https://github.com/Witiko/markdown/issues/573.

causes LaTeX to emulate most interfaces of enumitem [4], so neither package is going to be loaded.

In a future major version, enumitem will replace paralist altogether unless the emulation of enumitem becomes the default in LaTeX even without tagging, in which case both dependencies will disappear.

```
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 strikethroughs and marked text in pdfTeX.

```
soft soul
soft xcolor
```

**lua-ul and luacolor** Packages that are used in the default renderer prototypes for strikethroughs 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[5] 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.[6] community question answering web site under the `markdown` tag.

## 1.3 Acknowledgements

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

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 [5] 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.

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

The TEX implementation of the package draws inspiration from several sources including the source code of LATEX 2ε, 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.

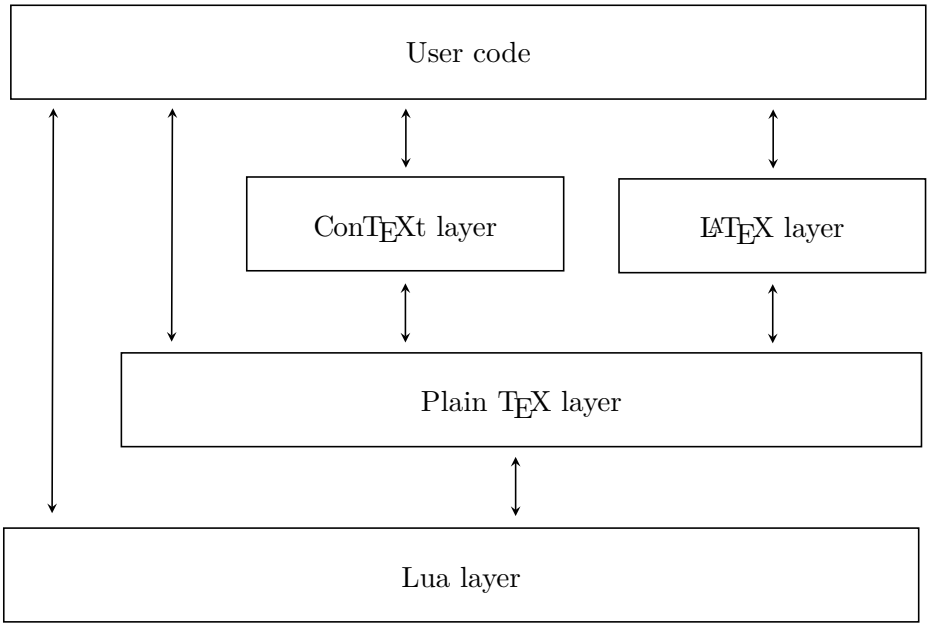


**Figure 1: A block diagram of the Markdown package**

## 2.1 Lua Interface

The Lua interface provides the conversion from UTF8 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.

```
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` 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.

```
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` and `experimentalOptions` tables.

```
local defaultOptions = {}
local experimentalOptions = {}
setmetatable(experimentalOptions, { __index = function (_, key)
  return defaultOptions[key] end })
```

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/experimental Lua options and their types, we will maintain the \g_@@_default_lua_options_prop, \g_@@_experimental_lua_options_seq and \g_@@_lua_option_types_prop property lists and sequences, respectively.

```
\prop_new:N \g_@@_lua_option_types_prop
\prop_new:N \g_@@_default_lua_options_prop
\seq_new:N \g_@@_experimental_lua_options_seq
\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_protected:Nn
  \@@_add_lua_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_lua_tl
      { #1 }
      { #2 }
      { #3 }
  }
\cs_new_protected: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 }
\cs_new_protected:Nn
  \@@_add_experimental_lua_option:n
  {
    \@@_add_experimental_option:Vn
      \c_@@_option_layer_lua_tl
      { #1 }
  }
\cs_new_protected:Nn
  \@@_add_experimental_option:nn
```

```
  {
    \seq_gput_right:cn
      { g_@@_ #1 _options_seq }
      { #2 }
    \seq_gput_right:cn
      { g_@@_experimental_ #1 _options_seq }
      { #2 }
    \prop_gput:cnn
      { g_@@_ #1 _option_types_prop }
      { #2 }
      { boolean }
  }
\cs_generate_variant:Nn
  \@@_add_experimental_option:nn
  { Vn }
\tl_const:Nn \c_@@_option_value_true_tl  { true  }
\tl_const:Nn \c_@@_option_value_false_tl { false }
\cs_new_protected: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:eV
                  { \l_tmpa_tl }
                  \c_@@_option_value_true_tl ||
                \str_if_eq_p:eV
                  { \l_tmpa_tl }
                  \c_@@_option_value_false_tl
              }
              {
                \msg_error:nnne
                  { 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
  { nnne }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { eV }
  { p }
\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_protected: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_generate_variant:Nn
  \msg_error:nnnn
  { nnnV }
\cs_new_protected:Nn
  \@@_get_default_option_value:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_if_in:cnT
          { g__markdown_experimental_ ##1 _options_seq }
          { #1 }
          {
            \@@_get_option_value:nN
              { experimental }
              #2
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
        \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_protected: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: }
          }
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nv
              #2
              { \l_tmpa_tl }
          }
      }
  }
\cs_new_protected: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 }
      }
  }
\regex_const:Nn
  \c_@@_option_regex
  { ^markdownOption }
\prg_new_protected_conditional:Nnn
  \@@_csname_to_option_str:nN
  { T }
  {
    \tl_if_regex_match:nNTF
      { #1 }
      \c_@@_option_regex
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \tl_regex_replace_once:NNn
          \l_tmpa_tl
          \c_@@_option_regex
          { }
        \tl_set:Nn
          \l_tmpb_tl
          {
            \str_lowercase:f
              { \l_tmpa_tl }
          }
        \str_set:Nx
          #2
          {
            \tl_head:f { \l_tmpb_tl }
            \tl_tail:V \l_tmpa_tl
          }
        \prg_return_true:
      }
      { \prg_return_false: }
  }
```

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_protected: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 }
  }
```

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_protected: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_protected: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.

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 has the following effects:

1. The version `experimental` of the theme `witiko/markdown/defaults` will be loaded.
2. Warnings for hard-deprecated features will become errors.
3. The experimental option `htmlOverLinks` will be enabled.

In the future, the effects may extend to other areas 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)[7].

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[8] before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

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

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

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

`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.

```
\str_new:N
  \g_@@_unquoted_jobname_str
\str_gset:NV
  \g_@@_unquoted_jobname_str
  \c_sys_jobname_str
\bool_new:N
  \g_@@_jobname_quoted_bool
\regex_replace_all:nnNTF
  { \A ("|') ( .* ) ("|') \Z }
  { \2 }
```

```
  \g_@@_unquoted_jobname_str
  {
    \bool_gset_true:N
      \g_@@_jobname_quoted_bool
  }
  {
    \bool_gset_false:N
      \g_@@_jobname_quoted_bool
  }
\@@_add_lua_option:nnn
  { cacheDir }
  { path }
  {
    \markdownOptionOutputDir
    / _markdown_
    \str_use:N
      \g_@@_unquoted_jobname_str
  }
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 2.2.5.14 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
    /
```

```
    \str_use:N
      \g_@@_unquoted_jobname_str
    .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

`acronyms`=⟨*acronyms*⟩

Acronyms, initialisms, and other all-caps sequences that will be recognized in markdown text and formatted accordingly.

```
\cs_generate_variant:Nn
  \@@_add_lua_option:nnn
  { nnV }
\@@_add_lua_option:nnV
  { acronyms }
  { clist }
  \c_empty_clist
defaultOptions.acronyms = {}
```

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

`true` Enable the Pandoc auto identifiers syntax extension[9]:

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

```
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
```

`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
```

`blankBeforeHtmlBlock`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following CommonMark HTML block[10].

`false` Do not require a blank line between a paragraph and the following CommonMark HTML block.

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

[10]See https://spec.commonmark.org/0.31.2/#html-blocks.

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

`true` Enable the Pandoc bracketed span syntax extension[11]:

```
[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
```

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

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

`true` Enable the Pandoc citation syntax extension[12]:

```
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
```

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

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

`true` Enable the iA Writer content blocks syntax extension [6]:

```
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`

`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.

`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.

See also the plain TeX macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the LaTeX environment `yaml` in Section 2.3.1, and the ConTeXt macros `\startyaml`,

\stopyaml, and \inputyaml in Section 2.4.1 to see how the options jekyllData, expectJekyllData, and ensureJekyllData can be used together to implement high-level interfaces for processing YAML documents.

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

expectJekyllData=true, false  default: false

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}
```

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}
```

See also the plain TEX macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the LATEX environment `yaml` in Section 2.3.1, and the ConTEXt macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
\@@_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.

```
\@@_add_lua_option:nnV
  { extensions }
  { clist }
  \c_empty_clist
```

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

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

`true` Enable the Pandoc fancy list syntax extension[13]:

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

`false` Disable the Pandoc fancy list syntax extension.

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

```
\@@_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
```

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

`true` Enable the Pandoc fenced code attribute syntax extension[14]:

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

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

`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[15]:

```
::::: {#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
```

`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
```

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

`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[16]:

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

# 123 Sesame Street
```

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

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
```

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

`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 HTML entities, inline HTML tags, and block HTML elements, comments, declarations, processing instructions, and CDATA sections in the input.

HTML entities will be replaced with the corresponding Unicode codepoints and other HTML content will be ignored by default.

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

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

`htmlOutput`=`basic`, `commonmark` default: `basic`

basic

: When the `html` option is enabled, produce basic coarse-grained renderers for all HTML, see Section 2.2.5.3.

commonmark

: When the `html` option is enabled, produce fine-grained renderers for each top-level non-terminal element of CommonMark's grammar for HTML blocks[17] and raw HTML[18], see sections 2.2.5.12 and 2.2.5.13, respectively.

[17] See https://spec.commonmark.org/0.31.2/#html-blocks.
[18] See https://spec.commonmark.org/0.31.2/#raw-html.

```
\@@_add_lua_option:nnn
  { htmlOutput }
  { string }
  { basic }
defaultOptions.htmlOutput = "basic"
```

`htmlOverLinks`=true, false default: `false`

`true` When the option `html` is enabled and a text can be understood either as a hyperlink or HTML, interpret it as HTML.

This is especially relevant when the option `relativeReferences` is enabled, since it makes e.g. `</foo>` a valid hyperlink.

`false` When the option `html` is enabled and a text can be understood either as a hyperlink or HTML, interpret it as a hyperlink.

This is an experimental option. Whenever the option `experimental` is enabled and this option is unspecified, it will be enabled. Like other experimental options, this option will be enabled by default and soft-deprecated in the next major release of the Markdown package.

```
\@@_add_experimental_lua_option:n
  { htmlOverLinks }
defaultOptions.htmlOverLinks = false
experimentalOptions.htmlOverLinks = 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[19]:

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

`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[20]:

```
Here is an inline note.^[Inline 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[21] 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.
```

[19] See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.
[20] See https://pandoc.org/MANUAL.html#extension-inline_notes.
[21] See https://pandoc.org/MANUAL.html#extension-yaml_metadata_block.

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

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

See also the plain TeX macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the LaTeX environment `yaml` in Section 2.3.1, and the ConTeXt macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
\@@_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[22]:

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

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

`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[23]:

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

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

`false` Disable the Pandoc line block syntax extension.

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

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

`true` Enable the Pandoc mark syntax extension[24]:

```
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[25]:

```
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.
```

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

`false` Disable the Pandoc note syntax extension.

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

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

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

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

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

```
\@@_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[26]:

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

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

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

`true` Enable relative references[27] 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.

[26] See https://pandoc.org/MANUAL.html#extension-raw_attribute.
[27] See https://datatracker.ietf.org/doc/html/rfc3986#section-4.2.

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

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

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

```
\@@_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[28]:

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

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

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

[28] 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[29]:

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

`false` Disable the Pandoc subscript syntax extension.

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

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

`true` Enable the Pandoc superscript syntax extension[30]:

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

`false` Disable the Pandoc superscript syntax extension.

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

```
\@@_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):

```
| 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
```

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

`true` Enable the Pandoc table caption syntax extension[31] for pipe tables (see the `pipeTables` option):

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

  : Demonstration of pipe table syntax.
```

`false` Disable the Pandoc table caption syntax extension.

```
\@@_add_lua_option:nnn
  { tableCaptions }
  { boolean }
  { false }
```

[31] See https://pandoc.org/MANUAL.html#extension-table_captions.

```
824 defaultOptions.tableCaptions = false
```

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

`true` Enable the Pandoc task list syntax extension[32]:

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

Our implementation also supports half-ticked task list items:

```
- [/] a half-checked task list item
- [.] another half-checked task list item
```

`false` Disable the Pandoc task list syntax extension.

```
825 \@@_add_lua_option:nnn
826   { taskLists }
827   { boolean }
828   { false }
829 defaultOptions.taskLists = false
```

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

`true` Strip TEX-style comments.

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

Always enabled when `hybrid` is enabled.

`false` Do not strip TEX-style comments.

```
830 \@@_add_lua_option:nnn
831   { texComments }
832   { boolean }
833   { false }
834 defaultOptions.texComments = false
```

[32] See https://pandoc.org/MANUAL.html#extension-task_lists.

texMathDollars=true, false default: false

true Enable the Pandoc dollar math syntax extension[33]:

```
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
```

texMathDoubleBackslash=true, false
default: false

true Enable the Pandoc double backslash math syntax extension[34]:

```
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[35]:

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

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

false Disable the Pandoc single backslash math syntax extension.

[33] See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.
[34] See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.
[35] See https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash.

```
\@@_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:

```
- 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.

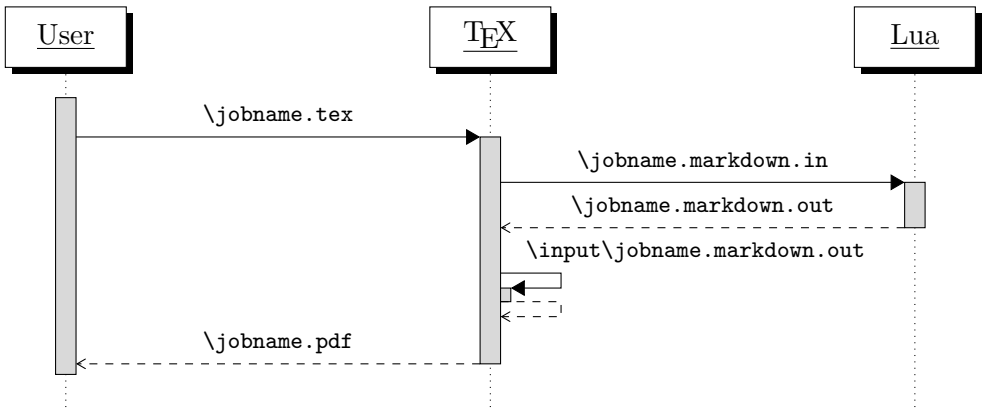


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface**

```
.TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
.SH NAME
markdown2tex \- convert .md files to .tex
.SH SYNOPSIS
local HELP_STRING = "Usage: " .. [[
markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]

```

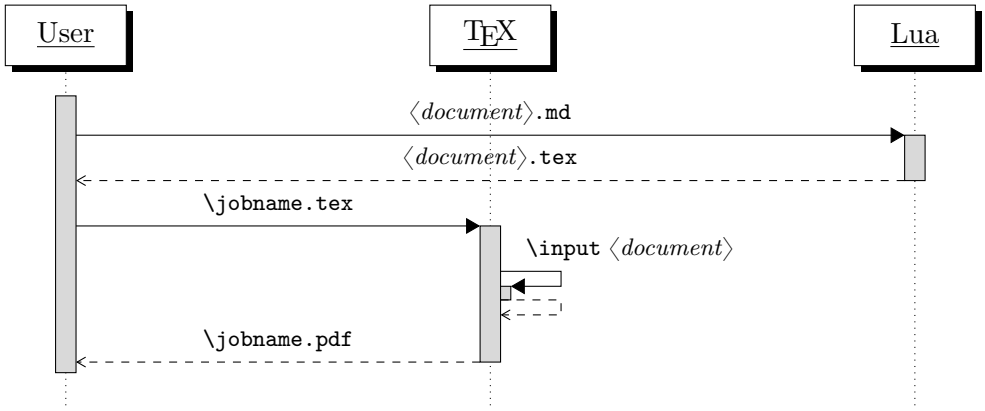


**Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface**

```
.SH DESCRIPTION

OPTIONS are documented in Section 2.2.1 of the Markdown Package User
Manual (https://ctan.org/pkg/markdown).

When OUTPUT_FILE is unspecified, the result of the conversion will be
written to the standard output. When INPUT_FILE is also unspecified, the
result of the conversion will be read from the standard input.

Report bugs to: witiko@mail.muni.cz
Markdown package home page: <https://github.com/witiko/markdown>]]

local VERSION_STRING = [[
markdown2tex (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 [7] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase

variants of options. As a bonus, studies [8] 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)
    return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
  end)
  return cased_option_name
end

local cases = {camel_case, snake_case}
local various_case_options = {}
for option_name, _ in pairs(defaultOptions) do
  for _, case in ipairs(cases) do
    various_case_options[case(option_name)] = option_name
  end
end

local process_options = true
local options = {}
local input_filename
local output_filename
for i = 1, #arg do
  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.

```
    if arg[i] == "--" then
      process_options = false
      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.

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

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

```
      local default_type = type(defaultOptions[key])
      if default_type == "boolean" then
        options[key] = (value == "true")
      elseif default_type == "number" then
        options[key] = tonumber(value)
      elseif default_type == "table" then
        options[key] = {}
        for item in value:gmatch("[^ ,]+") do
          table.insert(options[key], item)
        end
      else
        if default_type ~= "string" then
          if default_type == "nil" then
            warn('Option "' .. key .. '" not recognized.')
          else
            warn('Option "' .. key .. '" type not recognized, ' ..
                 'please file a report to the package maintainer.')
          end
          warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
               key .. '" as a string.')
        end
        options[key] = value
      end
      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.

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

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

```
  if input_filename == nil then
    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.

```
966   elseif output_filename == nil then
967     output_filename = arg[i]
968   else
969     error('Unexpected argument: "' .. arg[i] .. '".')
970   end
971   ::continue::
972 end
```

The command-line Lua interface is implemented by the files `markdown-cli.lua` and `markdown2tex.lua`, which can be invoked from the command line as follows:

```
markdown2tex 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.

```
973 \def\markdownLastModified{(((LASTMODIFIED)))}%
974 \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 [9, 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
```

Besides \markinline{⟨*markdown text*⟩}, any other delimiter such as |⟨*markdown text*⟩| can also be used. This allows potentially unbalanced braces to appear in ⟨*markdown text*⟩.

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/experimental plain TEX options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop`, `\g_@@_experimental_plain_tex_options` and `\g_@@_plain_tex_option_types_prop` property lists and sequences, respectively.

```
\prop_new:N \g_@@_plain_tex_option_types_prop
\prop_new:N \g_@@_default_plain_tex_options_prop
\seq_new:N \g_@@_experimental_plain_tex_options_seq
\seq_gput_right:NV
  \g_@@_option_layers_seq
  \c_@@_option_layer_plain_tex_tl
\cs_new_protected: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_protected: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_protected_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 (`\`).

```
\tl_set:Nn
  \l_tmpa_tl
  {
    \str_use:N
```

```
      \g_@@_unquoted_jobname_str
    .markdown.in
  }
\bool_if:NT
  \g_@@_jobname_quoted_bool
  {
    \tl_put_left:Nn
      \l_tmpa_tl
      { " }
    \tl_put_right:Nn
      \l_tmpa_tl
      { " }
  }
\cs_generate_variant:Nn
  \@@_add_plain_tex_option:nnn
  { nnV }
\@@_add_plain_tex_option:nnV
  { inputTempFileName }
  { path }
  \l_tmpa_tl
```

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.

In MikTEX, this automatic detection is currently only supported with LuaTEX[36]. If you need to use MikTEX and cannot use LuaTEX, you can either a) fix the automatic detection by setting the environmental variable `TEXMF_OUTPUT_DIRECTORY` manually or by setting the `\markdownOptionOutputDir` option manually.

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.

```
\@@_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 2.2.3). 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.

[36] See https://github.com/MiKTeX/miktex/issues/1630.

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 [10] 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_protected: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 [7] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [8] 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_protected:Nn
  \@@_define_option_command:n
  {
```

For experimental options, redirect the option command to the option command `\markdownOptionExperimental`.

```
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_if_in:cnT
          { g_@@_experimental_ ##1 _options_seq }
          { #1 }
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:NTF
      \l_tmpa_bool
      {
        \_@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_if_exist:cF
          { \l_tmpa_tl }
          {
            \cs_gset:cpn
              { \l_tmpa_tl }
              { \markdownOptionExperimental }
          }
      }
      {
```

Use the lt3luabridge library to determine the default value of the `\markdownOptionOutputDir` macro.

```
        \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 one of the following:

1. The `status.output_directory` variable [2, Section 10.2], which is available since LuaTEX 1.18.0 from TeX Live 2024 and in other TEX distributions like MikTEX since ca March 2024. We are only able to read this variable in LuaTEX and not other TEX engines.
2. The `TEXMF_OUTPUT_DIRECTORY` environmental variable, which is available since TEX Live 2024. We are only able to read this variable in TEX Live and not some other TEX distributions like MikTEX.

```
\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 the output directory name 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(
                  (status.output_directory)
                  or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
                  or~"."
                )
              }
            \tl_gset:NV
              \markdownOptionOutputDir
              \l_tmpa_tl
            \group_end:
          }
          {
            \tl_gset:Nn
              \markdownOptionOutputDir
              { . }
          }
      }
  }
\cs_new_protected: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_new_protected:Nn
  \@@_prepend_option_value:nn
  {
    \@@_get_option_value:nN
      { #1 }
      \l_tmpa_clist
    \clist_put_left:Nn
      \l_tmpa_clist
      { #2 }
    \@@_set_option_value:nV
      { #1 }
      \l_tmpa_clist
  }
\cs_new_protected:Nn
  \@@_append_option_value:nn
  {
    \@@_get_option_value:nN
      { #1 }
      \l_tmpa_clist
    \clist_put_right:Nn
      \l_tmpa_clist
      { #2 }
    \@@_set_option_value:nV
      { #1 }
      \l_tmpa_clist
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\cs_new_protected: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_protected: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 .value_required:n = { false },
```

```
            #3 .default:n = { true },
          }
      }
      {
        \keys_define:nn
          { markdown/options }
          {
            #3 .value_required:n = { true },
            #3 .code:n = {
              \@@_set_option_value:nn
                { #2 }
                { ##1 }
            },
          }
      }
```

For comma-list options, 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_set_eq:NN
      \l_tmpb_tl
      \l_tmpa_tl
    \tl_put_right:Nn
      \l_tmpa_tl
      { .value_required:n = { true } }
    \tl_put_right:Nn
      \l_tmpb_tl
      {
        .code:n = {
          \@@_append_option_value:nn
            { #2 }
            { ##1 }
        }
      }
    \keys_define:nV
      { markdown/options }
      \l_tmpa_tl
    \keys_define:nV
      { markdown/options }
      \l_tmpb_tl
```

To achieve parity with token renderers and renderer prototypes (see sections 2.2.5.48 and 2.2.6.2, respectively) the syntax ⟨*key*⟩`+=`⟨*value*⟩ and ⟨*key*⟩`$=`⟨*value*⟩ with the same appending semantics as ⟨*singular key*⟩`=`⟨*value*⟩ is also supported.

```
    % TODO: Remove the condition and the F-branch in TeX Live 2027.
    \@@_if_expl_at_least:nTF
      { 2026-10-28 }
      {
        \keys_define:nn
          { markdown/options }
          {
            #3 .append_method:n = {
              \@@_append_option_value:nn
                { #2 }
                { ##1 }
            },
          }
      }
      {
        \keys_define:nn
          { markdown/options }
          {
            #3~+ .value_required:n = { true },
            #3~+ .code:n = {
              \@@_append_option_value:nn
                { #2 }
                { ##1 }
            },
```

```
                #3 + .value_required:n = { true },
                #3 + .code:n = {
                  \@@_append_option_value:nn
                    { #2 }
                    { ##1 }
                },
              }
          }
        \keys_define:nn
          { markdown/options }
          {
            #3~$ .value_required:n = { true },
            #3~$ .code:n = {
              \@@_append_option_value:nn
                { #2 }
                { ##1 }
            },
            #3 $ .value_required:n = { true },
            #3 $ .code:n = {
              \@@_append_option_value:nn
                { #2 }
                { ##1 }
            },
```

Furthermore, similar to token renderers and renderer prototypes, the syntax ⟨*key*⟩`^=`⟨*value*⟩ is also supported. This syntax prepends ⟨*value*⟩ to the current value as the leftmost item in the list.

```
            #3~^ .value_required:n = { true },
            #3~^ .code:n = {
              \@@_prepend_option_value:nn
                { #2 }
                { ##1 }
            },
            #3 ^ .value_required:n = { true },
            #3 ^ .code:n = {
              \@@_prepend_option_value:nn
                { #2 }
                { ##1 }
            },
          }
      }
  }
\prg_new_conditional:Nnn
  \@@_if_expl_at_least:n
  { p, T, TF }
  {
    \bool_if:nTF
      {
        \str_if_eq_p:en
          { \ExplLoaderFileDate }
```

```
          { #1 } ||
        \str_compare_p:eNn
          { \ExplLoaderFileDate }
          >
          { #1 }
      }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\prg_new_conditional:Nnn
  \@@_if_expl_at_most:n
  { p }
  {
    \bool_if:nTF
      {
        \str_if_eq_p:en
          { \ExplLoaderFileDate }
          { #1 } ||
        \str_compare_p:eNn
          { \ExplLoaderFileDate }
          <
          { #1 }
      }
      { \prg_return_true: }
      { \prg_return_false: }
  }
\cs_generate_variant:Nn
  \clist_set:Nn
  { NV }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { p, F }
\prg_generate_conditional_variant:Nnn
  \str_compare:nNn
  { eNn }
  { p }
\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 [11, 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` [12].

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

```
    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_protected: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 strip a leading slash, if present, for consistency with snippets, where a leading slash distinguishes absolute snippet names from relative ones. Theme names, however, are currently always absolute, so stripping the slash is only a syntactic normalization and has no semantic effect.

```
\str_if_eq:enT
  {
    \tl_head:V
      \g_@@_current_theme_tl
  }
  { / }
  {
    \tl_gset:Ne
      \g_@@_current_theme_tl
      {
        \tl_tail:V
          \g_@@_current_theme_tl
      }
  }
```

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 }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { T }
```

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
```

1714 `\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 and for the infostrings `dot` and `plantuml`, the key-value attribute `format` can be used to specify the output image format. The remaining attributes are treated as image attributes.

```
\documentclass{article}
\usepackage[relativeReferences, texComments]{markdown}
\markdownSetup{import=witiko/diagrams@v2}
\begin{document}
\begin{markdown}
``` dot {caption="An example directed graph" %
          format=svg 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.

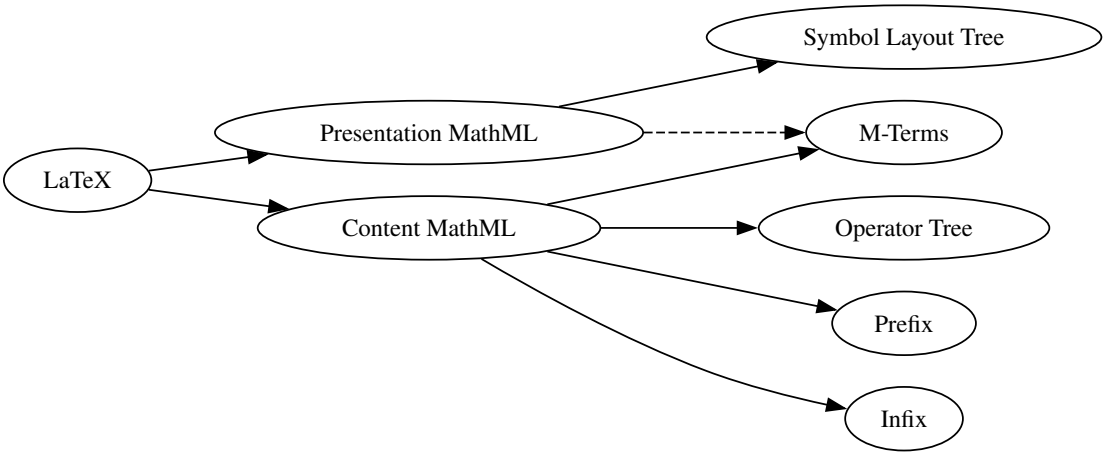


**Figure 4: An example directed graph**

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/glossaries** A theme that defines snippets with integrations for the LATEX package glossaries. See Section 2.3.5 for more details.

**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.

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

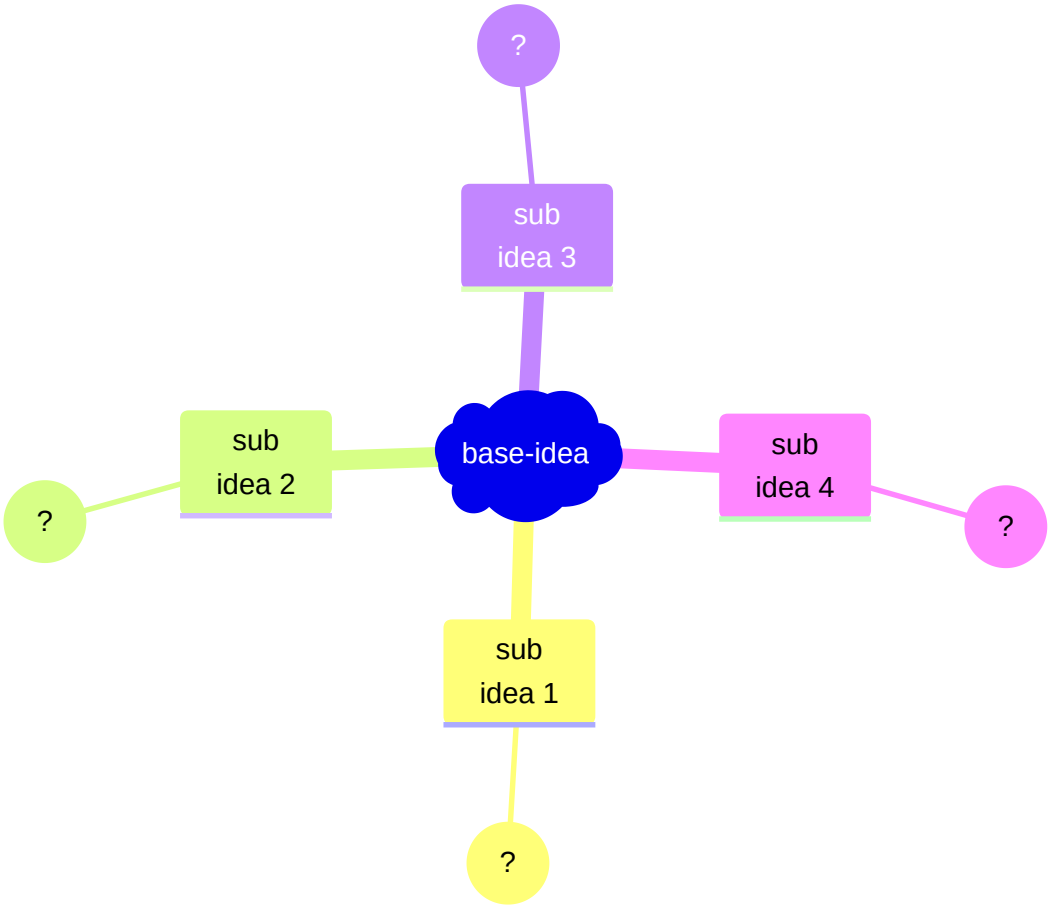


**Figure 5: An example mindmap**

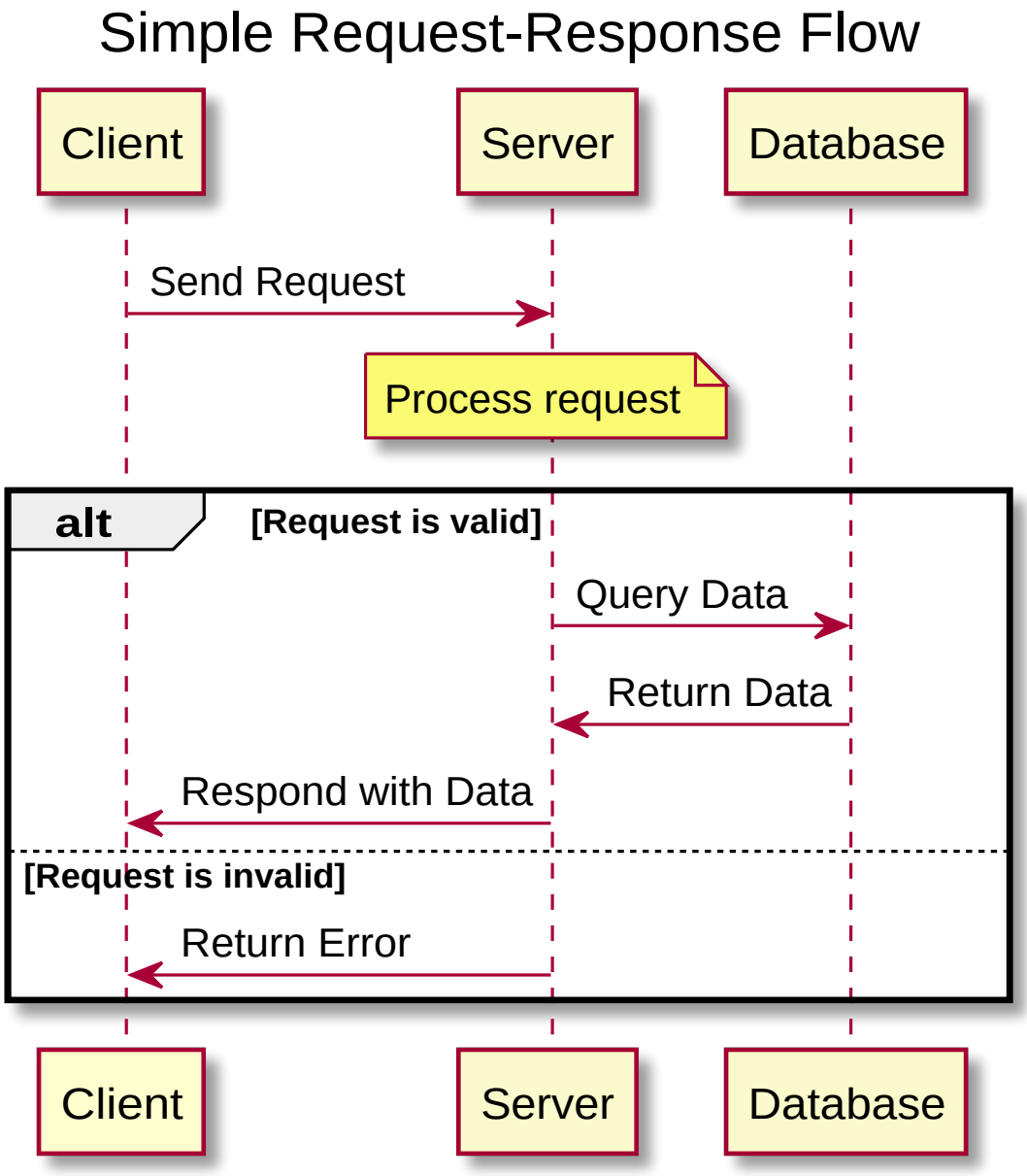


Figure 6: An example UML sequence diagram

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

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

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



Chapter 1

# Introduction

## 1.1 Section

### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

**Figure 7: The banner of the Markdown package**

```
\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.

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_protected:Nn
  \@@_setup_snippet:nn
  {
    \tl_if_empty:nT
      { #1 }
      {
        \msg_error:nnn
          { markdown }
          { empty-snippet-name }
          { #1 }
      }
    \@@_resolve_snippet_name:nN
      { #1 }
      \l_tmpa_tl
    \@@_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_protected_conditional:Nnn
```

```
  \@@_if_snippet_exists:n
  { TF, T }
  {
    \@@_resolve_snippet_name:nN
      { #1 }
      \l_@@_current_snippet_tl
    \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
\cs_new_protected:Nn
  \@@_resolve_snippet_name:nN
  {
```

If the provided snippet name starts with a slash (`/`), consider it as *absolute* and take the rest of the name as the full name of the defined snippet, regardless of the currently processed theme.

```
    \str_if_eq:enTF
      {
        \tl_head:n
          { #1 }
      }
      { / }
      {
        \tl_set:Ne
          #2
          {
            \tl_tail:n
              { #1 }
          }
      }
      {
```

Otherwise, consider the provided snippet name as *relative* and prepend the name of the currently processed theme to it, if any. The result is then the full name of the defined snippet.

```
        \tl_set:NV
          #2
          \g_@@_current_theme_tl
        \tl_put_right:Nn
          #2
          { #1 }
      }
  }
\prg_generate_conditional_variant:Nnn
```

```
  \str_if_eq:nn
  { en }
  { TF }
```

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

```
\keys_define:nn
  { markdown/options }
  {
    snippet .value_required:n = { true },
    snippet .code:n = {
      \@@_resolve_snippet_name:nN
        { #1 }
        \l_tmpa_tl
      \@@_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 .value_required:n = { false },
    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 Acronym Renderers

The following macro is only produced when the option `acronyms` is non-empty.

`\markdownRendererAcronym` represents an acronym, initialism, or another all-caps sequence. The macro receives a single attribute that corresponds to the sequence.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererAcronym
  {
    \markdownRendererAcronymPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { acronym }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { acronym }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.2 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 an 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
```

#### 2.2.5.3 Basic html Renderers

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. The macro receives a single argument that corresponds to the contents of the HTML comment.

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. 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.

These macros will only be produced when the `html` option is enabled and the `htmlOutput` option is set to `basic`.

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

#### 2.2.5.4 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
```

#### 2.2.5.5 Bracketed Spans

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

The `\markdownRendererBracketedSpan` macro represents an inline bracketed span. It receives a single argument that corresponds to the inline bracketed span.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererBracketedSpan
  {
    \markdownRendererBracketedSpanPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpan }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpan }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.6 Bracketed Span Attribute Context Renderers

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

The `\markdownRendererBracketedSpanAttributeContextBegin` and `\markdownRendererBracke` 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
```

#### 2.2.5.7 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
```

#### 2.2.5.8 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
```

#### 2.2.5.9 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
```

#### 2.2.5.10 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
```

#### 2.2.5.11 Code Span Attribute Context Renderers

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

The \markdownRendererCodeSpanAttributeContextBegin and \markdownRendererCodeSpanAtt 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
```

#### 2.2.5.12 CommonMark Block html Renderers

When the `html` option is enabled and `htmlOutput` is set to `commonmark`, the following fine-grained renderers are produced instead of `\markdownRendererInputBlockHtmlElement` to distinguish between different top-level non-terminal elements of CommonMark's grammar for raw HTML[37]:

1. `\markdownRendererInputBlockHtmlCdataElement` – A CDATA block-level HTML element such as `<script>`, `<pre>`, `<style>`, or `<textarea>` that starts on a line of its own
2. `\markdownRendererBlockHtmlComment` – A comment that starts on a line of its own
3. `\markdownRendererInputBlockHtmlProcessingInstruction` – A processing instruction that starts on a line of its own
4. `\markdownRendererInputBlockHtmlDeclaration` – A declaration that starts on a line of its own
5. `\markdownRendererInputBlockHtmlCdataSection` – A CDATA section that starts on a line of its own
6. `\markdownRendererInputBlockHtmlPcdataElement` – A PCDATA block-level HTML element that starts on a line of its own
7. `\markdownRendererBlockHtmlStandaloneTag` – A non-CDATA HTML tag on a line of its own

Each of these macros receive a single argument. For all macros except 2 and 7, it is the filename of a file containing the corresponding HTML text. In contrast, macros 2 and 7 receive the text within the HTML comment and the HTML text, respectively.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInputBlockHtmlCdataElement
  {
    \markdownRendererInputBlockHtmlCdataElementPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlCdataElement }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlCdataElement }
  { 1 }
```

[37] See https://spec.commonmark.org/0.31.2/#html-blocks.

```
\cs_gset_protected:Npn
  \markdownRendererBlockHtmlComment
  {
    \markdownRendererBlockHtmlCommentPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockHtmlComment }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlComment }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInputBlockHtmlProcessingInstruction
  {
    \markdownRendererInputBlockHtmlProcessingInstructionPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlProcessingInstruction }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlProcessingInstruction }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInputBlockHtmlDeclaration
  {
    \markdownRendererInputBlockHtmlDeclarationPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlDeclaration }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlDeclaration }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInputBlockHtmlCdataSection
  {
    \markdownRendererInputBlockHtmlCdataSectionPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlCdataSection }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlCdataSection }
  { 1 }
\cs_gset_protected:Npn
```

```
  \markdownRendererInputBlockHtmlPcdataElement
  {
    \markdownRendererInputBlockHtmlPcdataElementPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlPcdataElement }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlPcdataElement }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererBlockHtmlStandaloneTag
  {
    \markdownRendererBlockHtmlStandaloneTagPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockHtmlStandaloneTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlStandaloneTag }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.13 CommonMark Raw html Renderers

When the `html` option is enabled and `htmlOutput` is set to `commonmark`, the following renderers are produced instead of `\markdownRendererInlineHtmlTag` to distinguish between different top-level non-terminal elements of CommonMark's grammar for raw HTML[38]:

- `\markdownRendererInlineHtmlProcessingInstruction` – A processing instruction
- `\markdownRendererInlineHtmlDeclaration` – A declaration
- `\markdownRendererInlineHtmlCdataSection` – A CDATA section
- `\markdownRendererInlineHtmlOpenTag` – An opening tag
- `\markdownRendererInlineHtmlCloseTag` – A closing tag
- `\markdownRendererInlineHtmlEmptyTag` – An empty tag

Each of these macros receives a single argument with the HTML text.

```
\ExplSyntaxOn
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlProcessingInstruction
  {
    \markdownRendererInlineHtmlProcessingInstructionPrototype
```

[38] See https://spec.commonmark.org/0.31.2/#raw-html.

```
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlProcessingInstruction }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlProcessingInstruction }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlDeclaration
  {
    \markdownRendererInlineHtmlDeclarationPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlDeclaration }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlDeclaration }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlCdataSection
  {
    \markdownRendererInlineHtmlCdataSectionPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlCdataSection }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlCdataSection }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlOpenTag
  {
    \markdownRendererInlineHtmlOpenTagPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlOpenTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlOpenTag }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlCloseTag
  {
    \markdownRendererInlineHtmlCloseTagPrototype
  }
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlCloseTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlCloseTag }
  { 1 }
\cs_gset_protected:Npn
  \markdownRendererInlineHtmlEmptyTag
  {
    \markdownRendererInlineHtmlEmptyTagPrototype
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlEmptyTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlEmptyTag }
  { 1 }
\ExplSyntaxOff
```

#### 2.2.5.14 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[39] 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 [6] is a good starting point.

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

#### 2.2.5.15 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
```

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

```
\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
```

#### 2.2.5.16 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
```

#### 2.2.5.17 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
```

#### 2.2.5.18 Fenced Code Attribute Context Renderers

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

The \markdownRendererFencedCodeAttributeContextBegin and \markdownRendererFencedCod 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
```

#### 2.2.5.19 Fenced Div Attribute Context Renderers

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

The \markdownRendererFencedDivAttributeContextBegin and \markdownRendererFencedDivA 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
```

#### 2.2.5.20 Header Attribute Context Renderers

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

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttribu` 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
```

#### 2.2.5.21 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
```

#### 2.2.5.22 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
```

#### 2.2.5.23 Image Attribute Context Renderers

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

The `\markdownRendererImageAttributeContextBegin` and `\markdownRendererImageAttribute` 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
```

#### 2.2.5.24 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
```

#### 2.2.5.25 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
```

#### 2.2.5.26 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
```

#### 2.2.5.27 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
```

#### 2.2.5.28 Link Attribute Context Renderers

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

The \markdownRendererLinkAttributeContextBegin and \markdownRendererLinkAttributeCo 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
```

#### 2.2.5.29 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
```

#### 2.2.5.30 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
```

#### 2.2.5.31 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
```

#### 2.2.5.32 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
```

#### 2.2.5.33 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
```

#### 2.2.5.34 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
```

#### 2.2.5.35 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
```

#### 2.2.5.36 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
```

#### 2.2.5.37 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
```

#### 2.2.5.38 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
```

#### 2.2.5.39 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 }
```

#### 2.2.5.40 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
```

#### 2.2.5.41 Table Attribute Context Renderers

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

The `\markdownRendererTableAttributeContextBegin` and `\markdownRendererTableAttribute` 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
```

#### 2.2.5.42 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
```

#### 2.2.5.43 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
```

#### 2.2.5.44 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
```

#### 2.2.5.45 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
```

#### 2.2.5.46 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
```

#### 2.2.5.47 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 \markdownRendererJekyllDataPro 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 \markdownRendererJekyllDataTyp 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 \markdownRendererJekyllDataTypographicStr macro is more appropriate for texts that are supposed to be typeset with TEX, such as document titles, author names, or exam questions, the \markdownRendererJekyllDataProgrammaticStrin 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 \markdownRendererJekyllDataPr macro was not produced. The \markdownRendererJekyllDataString macro 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
```

#### 2.2.5.48 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_protected:Nn \@@_define_renderers:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer:n
          { ##1 }
      }
  }
\cs_new_protected: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_protected: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_@@_prepending_renderer_bool
\bool_new:N
  \g_@@_appending_renderer_bool
\bool_new:N
  \g_@@_unprotected_renderer_bool
\cs_new_protected:Nn \@@_define_renderer:nNn
```

```
{
  \keys_define:nn
    { markdown/options/renderers }
    {
      #1 .value_required:n = { true },
      #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_@@_prepending_renderer_bool ||
            \g_@@_appending_renderer_bool
          }
          {
            \@@_tl_set_from_cs:NNn
              \l_tmpa_tl
              #2
              { #3 }
            \bool_if:NTF
              \g_@@_prepending_renderer_bool
              {
                \tl_put_right:NV
                  \l_@@_renderer_definition_tl
                  \l_tmpa_tl
              }
              {
                \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` [13]. 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 .value_required:n = { true },
    renderers .code:n = {
      \bool_gset_false:N
        \g_@@_unprotected_renderer_bool
      \keys_set:nn
        { markdown/options/renderers }
        { #1 }
    },
    unprotectedRenderers .value_required:n = { true },
    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_@@_prepending_key_regex
  { \^$ }
\regex_const:Nn
  \c_@@_appending_key_regex
  { [\$+]$ }
\regex_const:Nn
  \c_@@_prepending_or_appending_key_regex
  { [\^\$+]$ }
\cs_new_protected:Nn
  \@@_redefine_renderer:n
  {
```

Besides defining renderers at once, we can also define them incrementally using the prepending (`^=`) and appending (`$=` or `+=`) operators. This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  renderers = {
    % Start with empty renderers.
    headerAttributeContextBegin = \begingroup,
    headerAttributeContextEnd = \endgroup,
    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 += {...},
        },
      }
    },
  },
}
```

```
    \regex_if_match:NVT
      \c_@@_prepending_key_regex
      \l_keys_key_str
      {
        \bool_gset_true:N
          \g_@@_prepending_renderer_bool
      }
    \regex_if_match:NVT
      \c_@@_appending_key_regex
      \l_keys_key_str
      {
        \bool_gset_true:N
          \g_@@_appending_renderer_bool
      }
    \regex_if_match:NVTF
      \c_@@_prepending_or_appending_key_regex
      \l_keys_key_str
      {
        \tl_set:NV
          \l_tmpa_tl
          \l_keys_key_str
        \regex_replace_once:NnN
          \c_@@_prepending_or_appending_key_regex
          { }
          \l_tmpa_tl
        \tl_set:Nx
          \l_tmpb_tl
          { { \l_tmpa_tl } = }
        \tl_put_right:Nn
          \l_tmpb_tl
          { { #1 } }
        \keys_set:nV
          { markdown/options/renderers }
          \l_tmpb_tl
        \bool_gset_false:N
          \g_@@_prepending_renderer_bool
        \bool_gset_false:N
          \g_@@_appending_renderer_bool
      }
```

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.
}
```

```
4014      {
4015        \@@_glob_seq:VnN
4016          \l_keys_key_str
4017          { g_@@_renderers_seq }
4018          \l_@@_renderer_glob_results_seq
4019        \seq_if_empty:NTF
4020          \l_@@_renderer_glob_results_seq
4021          {
4022            \msg_error:nnV
4023              { markdown }
4024              { undefined-renderer }
4025              \l_keys_key_str
4026          }
4027          {
4028            \tl_set:Nn
4029              \l_@@_renderer_glob_definition_tl
4030              { \exp_not:n { #1 } }
4031            \seq_map_inline:Nn
4032              \l_@@_renderer_glob_results_seq
4033              {
4034                \tl_set:Nn
4035                  \l_tmpa_tl
4036                  { { ##1 } = }
4037                \tl_put_right:Nx
4038                  \l_tmpa_tl
4039                  { { \l_@@_renderer_glob_definition_tl } }
4040                \keys_set:nV
```

```
                  { markdown/options/renderers }
                  \l_tmpa_tl
              }
          }
      }
  }
\keys_define:nn
  { markdown/options/renderers }
  {
    unknown .value_required:n = { true },
    unknown .code:n = {
      \@@_redefine_renderer:n
        { #1 }
    },
  }
% TODO: Remove the condition in TeX Live 2027.
\@@_if_expl_at_least:nT
  { 2026-10-28 }
  {
    \keys_define:nn
      { markdown/options/renderers }
      {
        unknown .append_method:n = {
          \str_put_right:Nn
            \l_keys_key_str
            { + }
          \@@_redefine_renderer:n
            { #1 }
        },
      }
  }
```

Fix `unknown` keys not supporting `.append_method:n` in some versions of LaTeX3. [14]

```
\bool_if:nT
  {
    \@@_if_expl_at_least_p:n
      { 2026-10-28 } &&
    \@@_if_expl_at_most_p:n
      { 2026-11-16 }
  }
  {
    \cs_gset_protected:Npn \__keys_execute:NN #1#2
      {
        \cs_if_exist:cTF { #1 \l_keys_path_str #2 }
          { \__keys_execute:Nn #1 { \l_keys_path_str #2 } }
          {
            \__keys_inherit:NNN #1 \__keys_execute:Nn #2
            \str_if_empty:NT \l__keys_inherit_str
              {
                \bool_if:NTF \l__keys_only_known_bool
```

```
                { \__keys_store_unused: }
                {
                  \str_if_eq:NNTF #1 \c__keys_code_root_str
                    { \__keys_execute_unknown: }
                    {
                      \__keys_execute_unknown_aux:N
                        \c__keys_append_root_str
                    }
                }
            }
        }
    \cs_gset_protected:Npn \__keys_execute_unknown:
      {
        \str_if_eq:VnTF \l__keys_aug_str { + }
          { \__keys_execute:NN \c__keys_append_root_str \c_empty_str }
          { \__keys_execute_unknown_aux:N \c__keys_code_root_str }
      }
    \cs_new_protected:Npn \__keys_execute_unknown_aux:N #1
      {
        \bool_if:NTF \l__keys_only_known_bool
          { \__keys_store_unused: }
          {
            \cs_if_exist:cTF
              { #1 \l__keys_module_str / unknown }
              {
                \bool_if:NT \l__keys_no_value_bool
                  {
                    \cs_if_exist:cT
                      {
                        \c__keys_default_root_str
                        \l__keys_module_str
                        / unknown
                      }
                      {
                        \tl_set_eq:Nc
                          \l_keys_value_tl
                          {
                            \c__keys_default_root_str
                            \l__keys_module_str
                            / unknown
                          }
                      }
                  }
                \__keys_execute:Nno #1
                  { \l__keys_module_str / unknown } { \l_keys_value_tl }
              }
              {
                \str_if_eq:NNTF #1 \c__keys_code_root_str
```

```
                  {
                    \msg_error:nnee { keys } { unknown }
                      { \l_keys_path_str } { \l__keys_module_str }
                  }
                  {
                    \msg_error:nnee { keys } { augmented-key-invalid }
                      { \l_keys_key_str }
                      { \exp_not:V \l_keys_value_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_if_match:Nn
  { NV }
  { T, TF }
\prop_new:N
  \g_@@_glob_cache_prop
\tl_new:N
  \l_@@_current_glob_tl
\cs_new_protected: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_if_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 [3] 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 .value_required:n = { true },
        #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 .value_required:n = { true },
    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
    }
  }
\ExplSyntaxOff
```

For complex YAML metadata, the option `jekyllDataKeyValue`=⟨*module*⟩ [15] 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` defines unprotected functions, which are easier to expand and may be preferable for programming.

```
\ExplSyntaxOn
\cs_new_protected:Nn \@@_define_renderer_prototypes:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
        \@@_define_renderer_prototype:n
          { ##1 }
      }
  }
\cs_new_protected: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_protected: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_@@_prepending_renderer_prototype_bool
\bool_new:N
  \g_@@_appending_renderer_prototype_bool
\bool_new:N
  \g_@@_unprotected_renderer_prototype_bool
\cs_new_protected:Nn \@@_define_renderer_prototype:nNn
  {
    \keys_define:nn
      { markdown/options/renderer-prototypes }
      {
        #1 .value_required:n = { true },
        #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_@@_prepending_renderer_prototype_bool ||
              \g_@@_appending_renderer_prototype_bool
            }
            {
              \@@_tl_set_from_cs:NNn
```

```
              \l_tmpa_tl
              #2
              { #3 }
            \bool_if:NTF
              \g_@@_prepending_renderer_prototype_bool
              {
                \tl_put_right:NV
                  \l_@@_renderer_prototype_definition_tl
                  \l_tmpa_tl
              }
              {
                \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 `\markdownRendererImagePrototy` 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.
  }
}
```

```
\cs_new_protected:Nn
    \@@_redefine_renderer_prototype:n
    {
```

Besides defining renderer prototypes at once, we can also define them incrementally using the prepending (`^=`) and appending (`$=` or `+=`) operators. This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = \begingroup,
    headerAttributeContextEnd = \endgroup,
    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 += {...},
        },
      }
    },
  },
}
```

```
\regex_if_match:NVT
  \c_@@_prepending_key_regex
  \l_keys_key_str
  {
    \bool_gset_true:N
      \g_@@_prepending_renderer_prototype_bool
  }
\regex_if_match:NVT
  \c_@@_appending_key_regex
  \l_keys_key_str
  {
    \bool_gset_true:N
      \g_@@_appending_renderer_prototype_bool
  }
\regex_if_match:NVTF
  \c_@@_prepending_or_appending_key_regex
  \l_keys_key_str
  {
    \tl_set:NV
      \l_tmpa_tl
      \l_keys_key_str
    \regex_replace_once:NnN
      \c_@@_prepending_or_appending_key_regex
      { }
      \l_tmpa_tl
    \tl_set:Nx
      \l_tmpb_tl
      { { \l_tmpa_tl } = }
    \tl_put_right:Nn
      \l_tmpb_tl
      { { #1 } }
    \keys_set:nV
      { markdown/options/renderer-prototypes }
      \l_tmpb_tl
    \bool_gset_false:N
      \g_@@_prepending_renderer_prototype_bool
    \bool_gset_false:N
      \g_@@_appending_renderer_prototype_bool
  }
```

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.
}
```

```
      {
        \@@_glob_seq:VnN
          \l_keys_key_str
          { g_@@_renderers_seq }
          \l_@@_renderer_glob_results_seq
        \seq_if_empty:NTF
          \l_@@_renderer_glob_results_seq
          {
            \msg_error:nnV
              { markdown }
              { undefined-renderer-prototype }
              \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
              }
          }
      }
  }
\keys_define:nn
  { markdown/options/renderer-prototypes }
  {
    unknown .value_required:n = { true },
    unknown .code:n = {
      \@@_redefine_renderer_prototype:n
        { #1 }
    },
  }
% TODO: Remove the condition in TeX Live 2027.
\@@_if_expl_at_least:nT
  { 2026-10-28 }
  {
    \keys_define:nn
      { markdown/options/renderer-prototypes }
      {
        unknown .append_method:n = {
          \str_put_right:Nn
            \l_keys_key_str
            { + }
          \@@_redefine_renderer_prototype:n
            { #1 }
        },
      }
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer-prototype }
  {
    Renderer~prototype~#1~is~undefined.
  }
\@@_with_various_cases:nn
```

```
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .value_required:n = { true },
        #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 .value_required:n = { true },
        #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 .value_required:n = { true },
    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 2.2.5.48) 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 $2_\varepsilon$ parses package options.

### 2.3.1 Typesetting Markdown and YAML

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.

```
4584 \newenvironment{markdown}\relax\relax
4585 \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.

```
4586 \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. [16, 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 "foo emphasis: _bar_ baz!", 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 "foo _bar_ baz!", 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 2.2.5.17. 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 2.2.5.42.

### 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 [17, 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.

```
4587 \DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
4588 \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
```

See Section 3.3.2 for implementation details of the built-in LaTeX themes.

### 2.3.5 Snippets

In Section 2.2.4, we described the concept of snippets.

Built-in LaTeX snippets provided with the Markdown package include:

**witiko/glossaries/import-acronyms** Imports all acronyms from the LaTeX package glossaries, appends them to the `acronyms` option and updates the corresponding renderers to recognize these acronyms in running text and link them to the corresponding glossary entries.

This snippet is provided by the theme `witiko/glossaries@v1`.

See Section 3.3.3 for implementation details of the built-in LaTeX snippets.

## 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.

```
4656 \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_protected: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_protected: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
```

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).

Furthermore, here are some abbreviations that we inherited from the Lunamark library and which are used throughout the Lua implementation.

```
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 Unicode Support

To start off, we load a Lua file named `markdown-unicode-data.lua` that contains our implementation of various Unicode algorithms.

```
local early_warnings = {}
local unicode_data = require("markdown-unicode-data")
if metadata.version ~= unicode_data.metadata.version then
  table.insert(
    early_warnings,
    "markdown.lua " .. metadata.version .. " used with " ..
    "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
  )
end
```

In the following subsections, we'll write a second-order file named `markdown-unicode-data-generator`. The code from this file will be executed during the compilation of the Markdown package and the standard output will be stored in a generated file named `markdown-unicode-data.lua`. First, let's define a couple helper functions.

The function `depth_first_search` performs the depth first search algorithm on a tree with nodes being tables with the key `_type` equal to either `intermediate` or `leaf` and with edges labeled with bytes.

Since the algorithm is implemented using recursion, it should only be used for the traversal of shallow trees to prevent exceeding the maximum recursion depth (usually 100).

```
local function depth_first_search(node, path, visit, leave)
  assert(node._type == "intermediate")
  visit(node, path)
  for label, child in pairs(node) do
    if label == "_type" then
      goto continue
    end
    if child._type ~= "intermediate" then
      goto continue
    end
    depth_first_search(child, path .. label, visit, leave)
    ::continue::
  end
  for label, child in pairs(node) do
    if label == "_type" then
      goto continue
```

```
    end
    if child._type ~= "leaf" then
      goto continue
    end
    visit(child, path)
    ::continue::
  end
  leave(node, path)
end
```

This function is defined in the file `markdown-parser.lua` as well, so that it can be used to define the `extensions.acronyms` built-in syntax extension. The function `serialize_byte_parser` produces the Lua code for a PEG parser that matches a single byte.

```
local function serialize_byte_parser(byte)
  if byte == '"' then
    return "P('" .. byte .. "')"
  elseif byte == "\\" then
    return 'P("\\\\")'
  else
    return 'P("' .. byte .. '")'
  end
end
```

The function `serialize_byte_range_parser` produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```
local function serialize_byte_range_parser(start_byte, end_byte)
  assert(start_byte <= end_byte)
  if start_byte == "\\" then
    start_byte = "\\\\"
  end
  if end_byte == "\\" then
    end_byte = "\\\\"
  end
  if start_byte == '"' or end_byte == '"' then
    return "R('" .. start_byte .. end_byte .. "')"
  else
    return 'R("' .. start_byte .. end_byte .. '")'
  end
end
```

The function `serialize_replacement` produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.

```
local function serialize_replacement(codepoints)
  assert(#codepoints > 0)
  local buffer = {'"'}
  for _, codepoint in ipairs(codepoints) do
    local code = utf8.char(codepoint)
    for i = 1, #code do
      local byte = code:sub(i, i)
```

```
      if byte == '"' then
        table.insert(buffer, '\\"')
      elseif byte == "\\" then
        table.insert(buffer, "\\\\")
      else
        table.insert(buffer, byte)
      end
    end
  end
  table.insert(buffer, '"')
  return table.concat(buffer)
end
```

The function `read_decompositions` is an iterator that reads a file `UnicodeData.txt` that has previously been opened for reading and yields all canonical and compatibility decompositions from that file.

```
local function read_decompositions(file)
  return function()
    local from_codepoint, mapping
    for line in file:lines() do
      from_codepoint, mapping
        = line:match("^(%x+);[^;]*;%a*;%d+;%a*;([<%a>%x ]*)")
      assert(from_codepoint ~= nil)
      if #mapping > 0 then
        break
      end
    end
    if #mapping == 0 then
      return nil
    end
    from_codepoint = tonumber(from_codepoint, 16)
    local to_codepoints, decomposition_type = {}, "canonical"
    for raw_codepoint in mapping:gmatch("%S+") do
      assert(#raw_codepoint > 0)
      if raw_codepoint:sub(1, 1) == "<" then
        decomposition_type = "compatibility"
      else
        local codepoint = tonumber(raw_codepoint, 16)
        table.insert(to_codepoints, codepoint)
      end
    end
    assert(#to_codepoints > 0)
    return decomposition_type, from_codepoint, to_codepoints
  end
end
```

Next, let's define some aliases in the generated file.

```
print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
print("local fail = P(false)")
print("-- luacheck: push no max line length")
```

#### 3.1.1.1 Canonical and Compatibility Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using either the canonical or the compatibility decomposition [18, Section 3.7] are organized in table `unicode_data.decomposition_mapping` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```
;(function()
  local file = assert(io.open("UnicodeData.txt", "r"),
    [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given decomposition type and code length.

```
  local decomposition_types = {"canonical", "compatibility"}
  local prefix_trees = {}
  for _, decomposition_type in ipairs(decomposition_types) do
    prefix_trees[decomposition_type] = {}
    for char_length = 1, 4 do
      prefix_trees[decomposition_type][char_length]
        = {_type = "intermediate"}
    end
  end
  for decomposition_type, from_codepoint, to_codepoints
      in read_decompositions(file) do
    local from_code = utf8.char(from_codepoint)
    local node = prefix_trees[decomposition_type][#from_code]
    for i = 1, #from_code do
      local from_byte = from_code:sub(i, i)
      if i < #from_code then
        if node[from_byte] == nil then
          node[from_byte] = {_type = "intermediate"}
        end
        node = node[from_byte]
      else
        table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
      end
    end
  end
  assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.decomposition_mapping = {}")
  for _, decomposition_type in ipairs(decomposition_types) do
    print("M.decomposition_mapping." .. decomposition_type .. " = {}")
    for length, prefix_tree in pairs(prefix_trees[decomposition_type])
    do
      local subparsers = {}
      depth_first_search(prefix_tree, "", function(node, path)
        if node._type == "leaf" then
          local from_byte, to_codepoints = table.unpack(node)
```

```
          local suffix = serialize_byte_parser(from_byte)
            .. " / " .. serialize_replacement(to_codepoints)
          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 = serialize_byte_parser(byte)
          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.decomposition_mapping." .. decomposition_type
            .. "[" .. length .. "] = " .. (subparsers[path] or "fail")
          )
        end
      end)
    end
  end
end)()
```

#### 3.1.1.2 Hangul Syllable Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using the Hangul syllable decomposition [18, Section 3.12.2] are also organized in table `unicode_data.decomposition_mapping`, previously defined in Section 3.1.1.1 based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `HangulSyllableType.txt`.

```
;(function()
  local file = assert(io.open("HangulSyllableType.txt", "r"),
    [[Could not open file "HangulSyllableType.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given syllable type and code length.

```
local syllable_types = {"LV", "LVT"}
local prefix_trees = {}
for _, syllable_type in ipairs(syllable_types) do
  prefix_trees[syllable_type] = {}
  for char_length = 1, 4 do
    prefix_trees[syllable_type][char_length]
      = {_type = "intermediate"}
  end
end
for line in file:lines() do
  if #line == 0 or line:sub(1, 1) == "#" then
    goto continue
  end
  local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
  assert(codepoint ~= nil)
  if prefix_trees[syllable_type] == nil then
    goto continue
  end
  local codepoint_start, codepoint_end
  if codepoint:find("%.%.") then
    codepoint_start, codepoint_end
      = codepoint:match("^(%x+)%.%.(%x+)$")
  else
    codepoint_start, codepoint_end = codepoint, codepoint
  end
  codepoint_start = tonumber(codepoint_start, 16)
  codepoint_end = tonumber(codepoint_end, 16)
  local prev_code, prev_leaf_node
  for codepoint = codepoint_start, codepoint_end do
    local code = utf8.char(codepoint)
    local node = prefix_trees[syllable_type][#code]
    for i = 1, #code do
      local byte = code:sub(i, i)
      if i < #code then
        if node[byte] == nil then
          node[byte] = {_type = "intermediate"}
        end
        node = node[byte]
      else
        local leaf_node
        if (
              prev_code ~= nil
              and #prev_code == #code
              and code:sub(1, #code - 1)
                == prev_code:sub(1, #code - 1)
            ) then
          assert(prev_leaf_node ~= nil)
          leaf_node = prev_leaf_node
          leaf_node[2] = byte
```

```
          else
            leaf_node = {byte, byte, _type = "leaf"}
            table.insert(node, leaf_node)
          end
          prev_code, prev_leaf_node = code, leaf_node
        end
      end
    end
    ::continue::
  end
  assert(file:close())
```

Next, we will define constants and functions with the Hangul syllable (de)composition algorithm.

```
  print(string.format("local s_base = %d", tonumber("AC00", 16)))
  print(string.format("local l_base = %d", tonumber("1100", 16)))
  print(string.format("local v_base = %d", tonumber("1161", 16)))
  print(string.format("local t_base = %d", tonumber("11A7", 16)))
  print(string.format("local l_count = %d", 19))
  print(string.format("local v_count = %d", 21))
  print(string.format("local t_count = %d", 28))
  print("local n_count = v_count * t_count")
  print("local s_count = l_count * n_count")

  print("local function hangul_decompose_LV(byte)")
  print("  local s = utf8.codepoint(byte)")
  print("  local s_index = s - s_base")
  print("  local l_index = s_index // n_count")
  print("  local v_index = (s_index % n_count) // t_count")
  print("  local l_part = l_base + l_index")
  print("  local v_part = v_base + v_index")
  print("  return utf8.char(l_part) .. utf8.char(v_part)")
  print("end")

  print("local function hangul_decompose_LVT(byte)")
  print("  local s = utf8.codepoint(byte)")
  print("  local s_index = s - s_base")
  print("  local lv_index = (s_index // t_count) * t_count")
  print("  local t_index = s_index % t_count")
  print("  local lv_part = s_base + lv_index")
  print("  local t_part = t_base + t_index")
  print("  return utf8.char(lv_part) .. utf8.char(t_part)")
  print("end")

  print("function M.hangul_compose(first_byte, second_byte)")
  print("  local last = utf8.codepoint(first_byte)")
  print("  local ch = utf8.codepoint(second_byte)")
  print("  local l_index = last - l_base")
  print("  if 0 <= l_index and l_index < l_count then")
  print("    local v_index = ch - v_base")
```

```
  print("    if 0 <= v_index and v_index < v_count then")
  print("      return utf8.char(")
  print("        s_base + (l_index * v_count + v_index) * t_count")
  print("      )")
  print("    end")
  print("  end")
  print("  local s_index = last - s_base")
  print("  if 0 <= s_index and s_index < s_count")
  print("      and s_index % t_count == 0 then")
  print("    local t_index = ch - t_base")
  print("    if 0 < t_index and t_index < t_count then")
  print("      return utf8.char(last + t_index)")
  print("    end")
  print("  end")
  print("  return nil")
  print("end")
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.decomposition_mapping.hangul = {}")
  for _, syllable_type in ipairs(syllable_types) do
    print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
    for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
      local subparsers = {}
      depth_first_search(prefix_tree, "", function(node, path)
        if node._type == "leaf" then
          local start_byte, end_byte = table.unpack(node)
          local suffix
          if start_byte == end_byte then
            suffix = serialize_byte_parser(start_byte)
          else
            assert(start_byte < end_byte)
            suffix = serialize_byte_range_parser(start_byte, end_byte)
          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 = serialize_byte_parser(byte)
          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
          if subparsers[path] then
            print(
              "M.decomposition_mapping.hangul." .. syllable_type
              .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
              .. " / hangul_decompose_" .. syllable_type
            )
          else
            print(
              "M.decomposition_mapping.hangul." .. syllable_type
              .. "[" .. length .. "] = fail"
            )
          end
        end
      end)
    end
  end
end)()
```

#### 3.1.1.3 Canonical Composition

Low-level parsers that map pairs of UTF-8-encoded Unicode characters from a canonical or compatibility decomposition into their primary composites [18, Section 3.11.6] are organized in table `unicode_data.composition_mapping` based on the number of bytes the characters occupy after conversion to UTF-8.

First, let's read the file `DerivedNormalizationProps.txt` and record all canonical decomposable characters that are not full composition exclusions.

```
;(function()
  local file = assert(io.open("DerivedNormalizationProps.txt", "r"),
    [[Could not open file "DerivedNormalizationProps.txt"]])
  local full_composition_exclusions = {}
  for line in file:lines() do
    if #line == 0 or line:sub(1, 1) == "#" then
      goto continue
    end
    local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
    assert(codepoint ~= nil)
    if property ~= "Full_Composition_Exclusion" then
      goto continue
    end
    local codepoint_start, codepoint_end
    if codepoint:find("%.%.") then
      codepoint_start, codepoint_end
```

```
      = codepoint:match("^(%x+)%.%.(%x+)$")
  else
    codepoint_start, codepoint_end = codepoint, codepoint
  end
  codepoint_start = tonumber(codepoint_start, 16)
  codepoint_end = tonumber(codepoint_end, 16)
  for codepoint = codepoint_start, codepoint_end do
    full_composition_exclusions[codepoint] = true
  end
  ::continue::
end
assert(file:close())
```

Next, let's also read the file `UnicodeData.txt`.

```
file = assert(io.open("UnicodeData.txt", "r"),
  [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all pairs of codepoints of given code lengths.

```
local prefix_trees = {starters = {}, both = {}}
for first_char_length = 1, 4 do
  prefix_trees.starters[first_char_length]
    = {_type = "intermediate"}
  prefix_trees.both[first_char_length] = {}
  for second_char_length = 1, 4 do
    prefix_trees.both[first_char_length][second_char_length]
      = {_type = "intermediate"}
  end
end
local seen_starter_codes = {}
for decomposition_type, to_codepoint, from_codepoints
    in read_decompositions(file) do
  if (
        decomposition_type ~= "canonical"
        or #from_codepoints ~= 2
        or full_composition_exclusions[to_codepoint]
      ) then
    goto continue
  end
  local starter_code = utf8.char(from_codepoints[1])
  local combining_character_code = utf8.char(from_codepoints[2])
  local starter_node = prefix_trees.starters[#starter_code]
  local both_node
    = prefix_trees.both[#starter_code][#combining_character_code]
  for i = 1, #starter_code do
    local from_byte = starter_code:sub(i, i)
    if both_node[from_byte] == nil then
      both_node[from_byte] = {_type = "intermediate"}
    end
    both_node = both_node[from_byte]
```

```
      if i < #starter_code then
        if starter_node[from_byte] == nil then
          starter_node[from_byte] = {_type = "intermediate"}
        end
        starter_node = starter_node[from_byte]
      elseif seen_starter_codes[starter_code] == nil then
        seen_starter_codes[starter_code] = true
        table.insert(starter_node, {from_byte, _type = "leaf"})
      end
    end
    for i = 1, #combining_character_code do
      local from_byte = combining_character_code:sub(i, i)
      if i < #combining_character_code then
        if both_node[from_byte] == nil then
          both_node[from_byte] = {_type = "intermediate"}
        end
        both_node = both_node[from_byte]
      else
        table.insert(
          both_node,
          {from_byte, to_codepoint, _type = "leaf"}
        )
      end
    end
    ::continue::
  end
  assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.composition_mapping = {starters = {}, both = {}}")
  for first_char_length = 1, 4 do
    local prefix_tree = prefix_trees.starters[first_char_length]
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if node._type == "leaf" then
        local from_byte = table.unpack(node)
        local suffix = serialize_byte_parser(from_byte)
        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 = serialize_byte_parser(byte)
        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.composition_mapping.starters["
          .. first_char_length .. "] = " .. (subparsers[path] or "fail")
        )
      end
    end)
    print(
      string.format(
        "M.composition_mapping.both[%d] = {}",
        first_char_length
      )
    )
    for second_char_length = 1, 4 do
      prefix_tree
        = prefix_trees.both[first_char_length][second_char_length]
      subparsers = {}
      depth_first_search(prefix_tree, "", function(node, path)
        if node._type == "leaf" then
          local from_byte, to_codepoint = table.unpack(node)
          local suffix = serialize_byte_parser(from_byte)
            .. " / " .. serialize_replacement({to_codepoint})
          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 = serialize_byte_parser(byte)
          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.composition_mapping.both[" .. first_char_length .. "]["
              .. second_char_length .. "] = "
              .. (subparsers[path] or "fail")
          )
        end
      end)
    end
  end
end)()
```

#### 3.1.1.4 Case Folding

Low-level parsers that case-fold UTF-8-encoded Unicode characters using the full mapping (C and F) [18, Section 3.13.3] are organized in table `unicode_data.casefold_mapping` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `CaseFolding.txt`.

```
;(function()
  local file = assert(io.open("CaseFolding.txt", "r"),
    [[Could not open file "CaseFolding.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given code length.

```
  local prefix_trees = {}
  for char_length = 1, 4 do
    prefix_trees[char_length] = {_type = "intermediate"}
  end
  for line in file:lines() do
    if #line == 0 or line:sub(1, 1) == "#" then
      goto continue
    end
    local raw_from_codepoint, status, raw_to_codepoints
      = line:match("^(%x+); ([CFST]); ([%x ]+);")
    assert(raw_from_codepoint ~= nil)
    assert(status ~= nil)
    assert(raw_to_codepoints ~= nil)
    if status ~= "C" and status ~= "F" then
      goto continue
    end
    local from_codepoint = tonumber(raw_from_codepoint, 16)
    local to_codepoints = {}
    for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
```

```
      local codepoint = tonumber(raw_codepoint, 16)
      table.insert(to_codepoints, codepoint)
    end
    local from_code = utf8.char(from_codepoint)
    local node = prefix_trees[#from_code]
    for i = 1, #from_code do
      local from_byte = from_code:sub(i, i)
      if i < #from_code then
        if node[from_byte] == nil then
          node[from_byte] = {_type = "intermediate"}
        end
        node = node[from_byte]
      else
        table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
      end
    end
    ::continue::
  end
  assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.casefold_mapping = {}")
  for length, prefix_tree in pairs(prefix_trees) do
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if node._type == "leaf" then
        local from_byte, to_codepoints = table.unpack(node)
        local suffix = serialize_byte_parser(from_byte)
          .. " / " .. serialize_replacement(to_codepoints)
        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 = serialize_byte_parser(byte)
        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.casefold_mapping[" .. length .. "] = "
          .. (subparsers[path] or "fail")
        )
      end
    end)
  end
end)()
```

#### 3.1.1.5 Character Categories

Low-level parsers of UTF-8-encoded Unicode characters from different general categories [18, Section 4.5] are organized in table `unicode_data.categories` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```
;(function()
  local file = assert(io.open("UnicodeData.txt", "r"),
    [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode category and code length.

```
  local categories = {"L", "N", "P", "Pc", "S", "Z"}
  local prefix_trees = {}
  for _, category in ipairs(categories) do
    prefix_trees[category] = {}
    for char_length = 1, 4 do
      prefix_trees[category][char_length] = {_type = "intermediate"}
    end
  end
  for line in file:lines() do
    local codepoint, full_category = line:match("^(%x+);[^;]*;(%a*)")
    assert(#full_category >= 1)
    local major_category = full_category:sub(1, 1)
    for _, category in ipairs({full_category, major_category}) do
      if prefix_trees[category] == nil then
        goto continue
      end
      local code = utf8.char(tonumber(codepoint, 16))
      local node = prefix_trees[category][#code]
      for i = 1, #code do
        local byte = code:sub(i, i)
        if i < #code then
          if node[byte] == nil then
            node[byte] = {_type = "intermediate"}
          end
          node = node[byte]
```

```
        else
          table.insert(node, {byte, _type = "leaf"})
        end
      end
      ::continue::
    end
  end
  assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.categories = {}")
  for _, category in ipairs(categories) do
    print("M.categories." .. category .. " = {}")
    for length, prefix_tree in pairs(prefix_trees[category]) do
      local subparsers = {}
      depth_first_search(prefix_tree, "", function(node, path)
        if node._type == "leaf" then
          local byte = node[1]
          local suffix = serialize_byte_parser(byte)
          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 = serialize_byte_parser(byte)
          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.categories." .. category .. "[" .. length .. "] = "
            .. (subparsers[path] or "fail")
          )
        end
      end)
    end
```

```
  end
end)()
```

#### 3.1.1.6 Canonical Ordering Classes

Low-level parsers of UTF-8-encoded Unicode characters from different character classes [18, Section 3.11] are organized in table `unicode_data.ccc` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```
;(function()
  local file = assert(io.open("UnicodeData.txt", "r"),
    [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode combining class and code length.

```
  local prefix_trees = {}
  for char_length = 1, 4 do
    prefix_trees[char_length] = {_type = "intermediate"}
  end
  for line in file:lines() do
    local codepoint, combining_class
      = line:match("^(%x+);[^;]*;%a*;(%d+)")
    combining_class = tonumber(combining_class)
    if combining_class == 0 then
      goto continue
    end
    local code = utf8.char(tonumber(codepoint, 16))
    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] = {_type = "intermediate"}
        end
        node = node[byte]
      else
        table.insert(node, {byte, combining_class, _type = "leaf"})
      end
    end
    ::continue::
  end
  assert(file:close())
```

Next, we will construct parsers out of the prefix trees.

```
  print("M.ccc = {}")
  for length, prefix_tree in pairs(prefix_trees) do
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if node._type == "leaf" then
        local byte, combining_class = table.unpack(node)
```

```
        local suffix = serialize_byte_parser(byte)
          .. " * Cc(" .. tostring(combining_class) .. ")"
        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 = serialize_byte_parser(byte)
        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.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
        )
      end
    end)
  end
end)()
print("-- luacheck: pop")
print("return M")
```

### 3.1.2 Utility Functions

This section documents the utility functions back in the file `markdown-parser.lua` 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)
```

```
5459 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 and the result of `transform(string)` is returned as the second return value in case it's useful to the caller. Regardless, the pathname is always returned as the first return value.

```
5460 function util.cache(dir, string, salt, transform, suffix)
5461   local digest = md5.sumhexa(string .. (salt or ""))
5462   local name = util.pathname(dir, digest .. suffix)
5463   local file = io.open(name, "r")
5464   local result = nil
5465   if file == nil then -- If no cache entry exists, create a new one.
5466     file = assert(io.open(name, "w"),
5467       [[Could not open file "]] .. name .. [[" for writing]])
5468     result = string
5469     if transform ~= nil then
5470       result = transform(result)
5471     end
5472     assert(file:write(result))
5473     assert(file:close())
5474   end
5475   return name, result
5476 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.

```
5477 function util.cache_verbatim(dir, string)
5478   local name = util.cache(dir, string, nil, nil, ".verbatim")
5479   return name
5480 end
```

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

```
5481 function util.table_copy(t)
5482   local u = { }
5483   for k, v in pairs(t) do u[k] = v end
5484   return setmetatable(u, getmetatable(t))
5485 end
```

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

```
5486 function util.encode_json_string(s)
5487   s = s:gsub([[\]], [[\\]])
5488   s = s:gsub([["]], [[\"]])
5489   return [["]] .. s .. [["]]
5490 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 [19, Chapter 21].

```
5491 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.

```
util.warning = (function()
  local function warning(s)
    io.stderr:write("Warning: " .. s .. "\n")
  end

  for _, message in ipairs(early_warnings) do
    warning(message)
  end

  return warning
end)()
```

The `util.casefold` method performs a full case-folding of a UTF-8-encoded Unicode string `s` based on the low-level parsers in `unicode_data.casefold_mapping`, defined in Section 3.1.1.4. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
util.casefold = (function()
  local fail, any = P(false), P(1)
  local eof = -any
```

First, define a parser that will case-fold a character.

```
  local fold_character = fail
  for n = 1, 4 do
    fold_character
      = fold_character
      + unicode_data.casefold_mapping[n]
  end
```

```
  fold_character
    = fold_character
    + C(any)
```

Next, define a parser that will case-fold a string.

```
  local fold_string = Ct(fold_character^0) * eof
  return function(s, form)
    local result = table.concat(lpeg.match(fold_string, s))
    assert(result ~= nil)
```

For NFD and NFKD normalization forms, normalize the case-folded string and then repeat the fold-and-normalize operation.

```
    if form == "nfd" or form == "nfkd" then
      result = util.normalize(result, form)
      result = table.concat(lpeg.match(fold_string, result))
      assert(result ~= nil)
      result = util.normalize(result, form)
    end
    return result
  end
end)()
```

The `util.canonically_order` method performs a Unicode canonical ordering of a string UTF-8-encoded Unicode `s` based on the low-level parsers in `unicode_data.ccc`, defined in Section 3.1.1.6. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
util.canonically_order = (function()
  local fail, any = P(false), P(1)
  local eof = -any
  local cont = R("\128\191")
  local utf8_character
    = R("\0\127")
    + R("\194\223") * cont
    + R("\224\239") * cont * cont
    + R("\240\244") * cont * cont * cont
```

First, define a parser that will determine the combining class of a character.

```
  local classify_character = fail
  for n = 1, 4 do
    classify_character
      = classify_character
      + unicode_data.ccc[n]
  end
  classify_character
    = classify_character
    + utf8_character * Cc(0)
```

Next, define a parser that will determine the combining classes of all characters in a string.

```
  local classify_string = Ct(classify_character^0) * eof
```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```
return function(s)
  local s_len = utf8.len(s)
  if s == false or s_len <= 1 then
    return s
  end
```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```
  local classes = lpeg.match(classify_string, s)
  if classes == nil then
    return s
  end
  assert(#classes == s_len)
```

Again, check whether the string is trivially ordered. If it is, return it without any changes. Otherwise, construct a list of ranges of non-starter characters that must be ordered.

```
  local non_starter_ranges = {}
  local first_non_starter, last_non_starter = nil, nil
  for i = 1, #classes do
    if first_non_starter == nil then
      if classes[i] ~= 0 then
        first_non_starter, last_non_starter = i, i
      end
    else
      if classes[i] == 0 then
        table.insert(
          non_starter_ranges,
          {first_non_starter, last_non_starter}
        )
        first_non_starter, last_non_starter = nil, nil
      else
        last_non_starter = i
      end
    end
  end
  if first_non_starter ~= nil then
    table.insert(
      non_starter_ranges,
      {first_non_starter, last_non_starter}
    )
  end
  if #non_starter_ranges == 0 then
    return s
  end
  local max_range_length = 0
  for _, range in ipairs(non_starter_ranges) do
    local range_start, range_end = table.unpack(range)
    local range_length = range_end - range_start + 1
```

```
      if range_length > max_range_length then
        max_range_length = range_length
      end
    end
    if max_range_length <= 1 then
      return s
    end
```

Then, construct a buffer of all characters in the string.

```
    local buffer = {}
    for _, code in utf8.codes(s) do
      local char = utf8.char(code)
      table.insert(buffer, char)
    end
    assert(#buffer == s_len)
```

Next, perform a local bubble sort over the ranges of non-starter characters.

```
    for _, range in ipairs(non_starter_ranges) do
      local range_start, range_end = table.unpack(range)
      local range_length = range_end - range_start + 1
      for _ = 1, range_length - 1 do
        local swapped = false
        for i = range_start, range_end - 1 do
          local j = i + 1
          if classes[i] > classes[j] then
            classes[i], classes[j] = classes[j], classes[i]
            buffer[i], buffer[j] = buffer[j], buffer[i]
            swapped = true
          end
        end
        if not swapped then
          break
        end
      end
    end
```

Finally, concatenate the buffer and return an ordered string.

```
    return table.concat(buffer, "")
  end
end)()
```

The `util.decompose` method performs either the canonical or the compatibility decomposition of a UTF-8-encoded Unicode string `s` based on the low-level parsers in `unicode_data.decomposition_mapping`, defined in sections 3.1.1.1 and 3.1.1.2. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
util.decompose = (function()
  local fail, any = P(false), P(1)
  local eof = -any
  local decomposition_types = {"canonical", "compatibility"}
```

First, define parsers that will decompose a character.

```
  local decompose_character = {}
  for _, decomposition_type in ipairs(decomposition_types) do
    decompose_character[decomposition_type] = fail
    for n = 1, 4 do
      decompose_character[decomposition_type]
        = decompose_character[decomposition_type]
        + unicode_data.decomposition_mapping[decomposition_type][n]
    end
    decompose_character[decomposition_type]
      = decompose_character[decomposition_type]
      + C(any)
  end
  local hangul = unicode_data.decomposition_mapping.hangul
  decompose_character.hangul = {}
  for syllable_type, _ in pairs(hangul) do
    decompose_character.hangul[syllable_type] = fail
    for n = 1, 4 do
      decompose_character.hangul[syllable_type]
        = decompose_character.hangul[syllable_type]
        + hangul[syllable_type][n]
    end
    decompose_character.hangul[syllable_type]
      = decompose_character.hangul[syllable_type]
      + C(any)
  end
```

Next, define a parser that will decompose a string.

```
  local decompose_string = {}
  for _, decomposition_type in ipairs(decomposition_types) do
    decompose_string[decomposition_type]
      = Ct(decompose_character[decomposition_type]^0) * eof
  end
  decompose_string.hangul = {}
  for syllable_type, _ in pairs(hangul) do
    decompose_string.hangul[syllable_type]
      = Ct(decompose_character.hangul[syllable_type]^0) * eof
  end
  local function _decompose(s, parser)
    assert(s ~= nil)
    local result = table.concat(lpeg.match(parser, s), "")
    assert(result ~= nil)
    return result
  end
  return function(s, decomposition_type)
    assert(
      decomposition_type == "canonical"
      or decomposition_type == "compatibility"
    )
    local prev_s
```

```
    local next_s = s
    repeat
      prev_s = next_s
      local function decompose(...) next_s = _decompose(next_s, ...) end
      decompose(decompose_string.canonical)
      if decomposition_type == "compatibility" then
        decompose(decompose_string.compatibility)
      end
      decompose(decompose_string.hangul.LVT)
      decompose(decompose_string.hangul.LV)
    until prev_s == next_s
    return util.canonically_order(next_s)
  end
end)()
```

The `util.compose` method performs the canonical composition of a UTF-8-encoded canonically ordered Unicode string `s` based on the low-level parsers in `unicode_data.composition_mapping`, defined in Section 3.1.1.3, and definitions from the Hangul syllable (de)composition algorithm, defined in Section 3.1.1.2. Unlike the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
util.compose = (function()
  local fail, any = P(false), P(1)
  local eof = -any
  local cont = R("\128\191")
  local utf8_character
    = R("\0\127")
    + R("\194\223") * cont
    + R("\224\239") * cont * cont
    + R("\240\244") * cont * cont * cont
```

First, define a parser that will determine the combining class of a character.

```
  local classify_character = fail
  for n = 1, 4 do
    classify_character
      = classify_character
      + unicode_data.ccc[n]
  end
  classify_character
    = classify_character
    + utf8_character * Cc(0)
```

Next, define a parser that will determine the combining classes of all characters in a string.

```
  local classify_string = Ct(classify_character^0) * eof
```

First, define parsers that will compose a pair of UTF-8-encoded Unicode characters into their primary composite.

```
  local compose_characters = fail
  for m = 1, 4 do
```

```
    local starter = #unicode_data.composition_mapping.starters[m]
    local both = fail
    for n = 1, 4 do
      both = (
        both
        + #unicode_data.composition_mapping.both[m][n]
        * unicode_data.composition_mapping.both[m][n]
      )
    end
    compose_characters = compose_characters + starter * both
  end
```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```
  return function(s)
    local s_len = utf8.len(s)
    if s == false or s_len <= 1 then
      return s
    end
```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```
    local classes = lpeg.match(classify_string, s)
    if classes == nil then
      return s
    end
    assert(#classes == s_len)
```

Otherwise, construct a buffer of all characters in the string.

```
    local buffer = {}
    for _, code in utf8.codes(s) do
      local char = utf8.char(code)
      table.insert(buffer, char)
    end
    assert(#buffer == s_len)
```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```
    local starter = 1
    while starter <= s_len and classes[starter] ~= 0 do
      starter = starter + 1
    end
    local candidate_combining_mark = starter + 1
```

Next, apply the composition rules until we reach the end of the string.

```
    while candidate_combining_mark <= s_len do
      local L = buffer[starter]
      local C = buffer[candidate_combining_mark]
      local P = lpeg.match(compose_characters, L .. C)
      if P ~= nil then
        buffer[starter] = P
```

```
5864          buffer[candidate_combining_mark] = ""
5865        else
5866          if classes[candidate_combining_mark] == 0 then
5867            starter = candidate_combining_mark
5868          end
5869          candidate_combining_mark = candidate_combining_mark + 1
5870        end
5871        assert(starter <= s_len)
5872      end
```

Next, iterate over the string once more and compose Hangul syllables.

```
5873      for i = 1, s_len - 1 do
5874        local last, ch = buffer[i], buffer[i + 1]
5875        if last ~= "" and ch ~= "" then
5876          local composite = unicode_data.hangul_compose(last, ch)
5877          if composite ~= nil then
5878            buffer[i] = ""
5879            buffer[i + 1] = composite
5880          end
5881        end
5882      end
```

Finally, concatenate the buffer and return an ordered string.

```
5883      return table.concat(buffer, "")
5884    end
5885  end)()
```

The `util.normalize` method normalizes a UTF-8-encoded canonically ordered Unicode string `s` using the normalization form `form`.

```
5886  function util.normalize(s, form)
5887    if form == "nfd" then
5888      return util.decompose(s, "canonical")
5889    elseif form == "nfkd" then
5890      return util.decompose(s, "compatibility")
5891    elseif form == "nfc" then
5892      return util.compose(util.decompose(s, "canonical"))
5893    elseif form == "nfkc" then
5894      return util.compose(util.decompose(s, "compatibility"))
5895    else
5896      error(string.format('Unexpected normal form "%s"', form))
5897    end
5898  end
```

The `util.find_file` and `util.find_files` method find the first or all locations of a file, respectively, according to either the resolvers API [1, Section 11.5] from the ConTEXt format or the Kpathsea library.

```
5899  function util.find_file(filename)
5900    if resolvers ~= nil then
5901      return resolvers.findfile(filename)
5902    else
5903      return kpse.find_file(filename)
```

```
  end
end
function util.find_files(filename)
  if resolvers ~= nil then
    return resolvers.findfiles(filename)
  else
    return {kpse.lookup(filename, {all=true})}
  end
end
```

### 3.1.3 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,
  ["lpar"] = 40,
  ["rpar"] = 41,
  ["ast"] = 42,
  ["midast"] = 42,
  ["plus"] = 43,
  ["comma"] = 44,
  ["period"] = 46,
  ["sol"] = 47,
  ["colon"] = 58,
  ["semi"] = 59,
  ["LT"] = 60,
  ["lt"] = 60,
  ["nvlt"] = {60, 8402},
  ["bne"] = {61, 8421},
  ["equals"] = 61,
  ["GT"] = 62,
  ["gt"] = 62,
  ["nvgt"] = {62, 8402},
  ["quest"] = 63,
  ["commat"] = 64,
```

```
5947    ["lbrack"] = 91,
5948    ["lsqb"] = 91,
5949    ["bsol"] = 92,
5950    ["rbrack"] = 93,
5951    ["rsqb"] = 93,
5952    ["Hat"] = 94,
5953    ["UnderBar"] = 95,
5954    ["lowbar"] = 95,
5955    ["DiacriticalGrave"] = 96,
5956    ["grave"] = 96,
5957    ["fjlig"] = {102, 106},
5958    ["lbrace"] = 123,
5959    ["lcub"] = 123,
5960    ["VerticalLine"] = 124,
5961    ["verbar"] = 124,
5962    ["vert"] = 124,
5963    ["rbrace"] = 125,
5964    ["rcub"] = 125,
5965    ["NonBreakingSpace"] = 160,
5966    ["nbsp"] = 160,
5967    ["iexcl"] = 161,
5968    ["cent"] = 162,
5969    ["pound"] = 163,
5970    ["curren"] = 164,
5971    ["yen"] = 165,
5972    ["brvbar"] = 166,
5973    ["sect"] = 167,
5974    ["Dot"] = 168,
5975    ["DoubleDot"] = 168,
5976    ["die"] = 168,
5977    ["uml"] = 168,
5978    ["COPY"] = 169,
5979    ["copy"] = 169,
5980    ["ordf"] = 170,
5981    ["laquo"] = 171,
5982    ["not"] = 172,
5983    ["shy"] = 173,
5984    ["REG"] = 174,
5985    ["circledR"] = 174,
5986    ["reg"] = 174,
5987    ["macr"] = 175,
5988    ["strns"] = 175,
5989    ["deg"] = 176,
5990    ["PlusMinus"] = 177,
5991    ["plusmn"] = 177,
5992    ["pm"] = 177,
5993    ["sup2"] = 178,
5994    ["sup3"] = 179,
5995    ["DiacriticalAcute"] = 180,
```

```
5996    ["acute"] = 180,
5997    ["micro"] = 181,
5998    ["para"] = 182,
5999    ["CenterDot"] = 183,
6000    ["centerdot"] = 183,
6001    ["middot"] = 183,
6002    ["Cedilla"] = 184,
6003    ["cedil"] = 184,
6004    ["sup1"] = 185,
6005    ["ordm"] = 186,
6006    ["raquo"] = 187,
6007    ["frac14"] = 188,
6008    ["frac12"] = 189,
6009    ["half"] = 189,
6010    ["frac34"] = 190,
6011    ["iquest"] = 191,
6012    ["Agrave"] = 192,
6013    ["Aacute"] = 193,
6014    ["Acirc"] = 194,
6015    ["Atilde"] = 195,
6016    ["Auml"] = 196,
6017    ["Aring"] = 197,
6018    ["angst"] = 197,
6019    ["AElig"] = 198,
6020    ["Ccedil"] = 199,
6021    ["Egrave"] = 200,
6022    ["Eacute"] = 201,
6023    ["Ecirc"] = 202,
6024    ["Euml"] = 203,
6025    ["Igrave"] = 204,
6026    ["Iacute"] = 205,
6027    ["Icirc"] = 206,
6028    ["Iuml"] = 207,
6029    ["ETH"] = 208,
6030    ["Ntilde"] = 209,
6031    ["Ograve"] = 210,
6032    ["Oacute"] = 211,
6033    ["Ocirc"] = 212,
6034    ["Otilde"] = 213,
6035    ["Ouml"] = 214,
6036    ["times"] = 215,
6037    ["Oslash"] = 216,
6038    ["Ugrave"] = 217,
6039    ["Uacute"] = 218,
6040    ["Ucirc"] = 219,
6041    ["Uuml"] = 220,
6042    ["Yacute"] = 221,
6043    ["THORN"] = 222,
6044    ["szlig"] = 223,
```

```
6045    ["agrave"] = 224,
6046    ["aacute"] = 225,
6047    ["acirc"] = 226,
6048    ["atilde"] = 227,
6049    ["auml"] = 228,
6050    ["aring"] = 229,
6051    ["aelig"] = 230,
6052    ["ccedil"] = 231,
6053    ["egrave"] = 232,
6054    ["eacute"] = 233,
6055    ["ecirc"] = 234,
6056    ["euml"] = 235,
6057    ["igrave"] = 236,
6058    ["iacute"] = 237,
6059    ["icirc"] = 238,
6060    ["iuml"] = 239,
6061    ["eth"] = 240,
6062    ["ntilde"] = 241,
6063    ["ograve"] = 242,
6064    ["oacute"] = 243,
6065    ["ocirc"] = 244,
6066    ["otilde"] = 245,
6067    ["ouml"] = 246,
6068    ["div"] = 247,
6069    ["divide"] = 247,
6070    ["oslash"] = 248,
6071    ["ugrave"] = 249,
6072    ["uacute"] = 250,
6073    ["ucirc"] = 251,
6074    ["uuml"] = 252,
6075    ["yacute"] = 253,
6076    ["thorn"] = 254,
6077    ["yuml"] = 255,
6078    ["Amacr"] = 256,
6079    ["amacr"] = 257,
6080    ["Abreve"] = 258,
6081    ["abreve"] = 259,
6082    ["Aogon"] = 260,
6083    ["aogon"] = 261,
6084    ["Cacute"] = 262,
6085    ["cacute"] = 263,
6086    ["Ccirc"] = 264,
6087    ["ccirc"] = 265,
6088    ["Cdot"] = 266,
6089    ["cdot"] = 267,
6090    ["Ccaron"] = 268,
6091    ["ccaron"] = 269,
6092    ["Dcaron"] = 270,
6093    ["dcaron"] = 271,
```

```
6094    ["Dstrok"] = 272,
6095    ["dstrok"] = 273,
6096    ["Emacr"] = 274,
6097    ["emacr"] = 275,
6098    ["Edot"] = 278,
6099    ["edot"] = 279,
6100    ["Eogon"] = 280,
6101    ["eogon"] = 281,
6102    ["Ecaron"] = 282,
6103    ["ecaron"] = 283,
6104    ["Gcirc"] = 284,
6105    ["gcirc"] = 285,
6106    ["Gbreve"] = 286,
6107    ["gbreve"] = 287,
6108    ["Gdot"] = 288,
6109    ["gdot"] = 289,
6110    ["Gcedil"] = 290,
6111    ["Hcirc"] = 292,
6112    ["hcirc"] = 293,
6113    ["Hstrok"] = 294,
6114    ["hstrok"] = 295,
6115    ["Itilde"] = 296,
6116    ["itilde"] = 297,
6117    ["Imacr"] = 298,
6118    ["imacr"] = 299,
6119    ["Iogon"] = 302,
6120    ["iogon"] = 303,
6121    ["Idot"] = 304,
6122    ["imath"] = 305,
6123    ["inodot"] = 305,
6124    ["IJlig"] = 306,
6125    ["ijlig"] = 307,
6126    ["Jcirc"] = 308,
6127    ["jcirc"] = 309,
6128    ["Kcedil"] = 310,
6129    ["kcedil"] = 311,
6130    ["kgreen"] = 312,
6131    ["Lacute"] = 313,
6132    ["lacute"] = 314,
6133    ["Lcedil"] = 315,
6134    ["lcedil"] = 316,
6135    ["Lcaron"] = 317,
6136    ["lcaron"] = 318,
6137    ["Lmidot"] = 319,
6138    ["lmidot"] = 320,
6139    ["Lstrok"] = 321,
6140    ["lstrok"] = 322,
6141    ["Nacute"] = 323,
6142    ["nacute"] = 324,
```

```
6143    ["Ncedil"] = 325,
6144    ["ncedil"] = 326,
6145    ["Ncaron"] = 327,
6146    ["ncaron"] = 328,
6147    ["napos"] = 329,
6148    ["ENG"] = 330,
6149    ["eng"] = 331,
6150    ["Omacr"] = 332,
6151    ["omacr"] = 333,
6152    ["Odblac"] = 336,
6153    ["odblac"] = 337,
6154    ["OElig"] = 338,
6155    ["oelig"] = 339,
6156    ["Racute"] = 340,
6157    ["racute"] = 341,
6158    ["Rcedil"] = 342,
6159    ["rcedil"] = 343,
6160    ["Rcaron"] = 344,
6161    ["rcaron"] = 345,
6162    ["Sacute"] = 346,
6163    ["sacute"] = 347,
6164    ["Scirc"] = 348,
6165    ["scirc"] = 349,
6166    ["Scedil"] = 350,
6167    ["scedil"] = 351,
6168    ["Scaron"] = 352,
6169    ["scaron"] = 353,
6170    ["Tcedil"] = 354,
6171    ["tcedil"] = 355,
6172    ["Tcaron"] = 356,
6173    ["tcaron"] = 357,
6174    ["Tstrok"] = 358,
6175    ["tstrok"] = 359,
6176    ["Utilde"] = 360,
6177    ["utilde"] = 361,
6178    ["Umacr"] = 362,
6179    ["umacr"] = 363,
6180    ["Ubreve"] = 364,
6181    ["ubreve"] = 365,
6182    ["Uring"] = 366,
6183    ["uring"] = 367,
6184    ["Udblac"] = 368,
6185    ["udblac"] = 369,
6186    ["Uogon"] = 370,
6187    ["uogon"] = 371,
6188    ["Wcirc"] = 372,
6189    ["wcirc"] = 373,
6190    ["Ycirc"] = 374,
6191    ["ycirc"] = 375,
```

```
6192    ["Yuml"] = 376,
6193    ["Zacute"] = 377,
6194    ["zacute"] = 378,
6195    ["Zdot"] = 379,
6196    ["zdot"] = 380,
6197    ["Zcaron"] = 381,
6198    ["zcaron"] = 382,
6199    ["fnof"] = 402,
6200    ["imped"] = 437,
6200    ["gacute"] = 501,
6201    ["jmath"] = 567,
6202    ["circ"] = 710,
6203    ["Hacek"] = 711,
6204    ["caron"] = 711,
6205    ["Breve"] = 728,
6206    ["breve"] = 728,
6207    ["DiacriticalDot"] = 729,
6208    ["dot"] = 729,
6209    ["ring"] = 730,
6210    ["ogon"] = 731,
6211    ["DiacriticalTilde"] = 732,
6212    ["tilde"] = 732,
6213    ["DiacriticalDoubleAcute"] = 733,
6214    ["dblac"] = 733,
6215    ["DownBreve"] = 785,
6216    ["Alpha"] = 913,
6217    ["Beta"] = 914,
6218    ["Gamma"] = 915,
6219    ["Delta"] = 916,
6220    ["Epsilon"] = 917,
6221    ["Zeta"] = 918,
6222    ["Eta"] = 919,
6223    ["Theta"] = 920,
6224    ["Iota"] = 921,
6225    ["Kappa"] = 922,
6226    ["Lambda"] = 923,
6227    ["Mu"] = 924,
6228    ["Nu"] = 925,
6229    ["Xi"] = 926,
6230    ["Omicron"] = 927,
6231    ["Pi"] = 928,
6232    ["Rho"] = 929,
6233    ["Sigma"] = 931,
6234    ["Tau"] = 932,
6235    ["Upsilon"] = 933,
6236    ["Phi"] = 934,
6237    ["Chi"] = 935,
6238    ["Psi"] = 936,
6239    ["Omega"] = 937,
6240
```

```
6241    ["ohm"] = 937,
6242    ["alpha"] = 945,
6243    ["beta"] = 946,
6244    ["gamma"] = 947,
6245    ["delta"] = 948,
6246    ["epsi"] = 949,
6247    ["epsilon"] = 949,
6248    ["zeta"] = 950,
6249    ["eta"] = 951,
6250    ["theta"] = 952,
6251    ["iota"] = 953,
6252    ["kappa"] = 954,
6253    ["lambda"] = 955,
6254    ["mu"] = 956,
6255    ["nu"] = 957,
6256    ["xi"] = 958,
6257    ["omicron"] = 959,
6258    ["pi"] = 960,
6259    ["rho"] = 961,
6260    ["sigmaf"] = 962,
6261    ["sigmav"] = 962,
6262    ["varsigma"] = 962,
6263    ["sigma"] = 963,
6264    ["tau"] = 964,
6265    ["upsi"] = 965,
6266    ["upsilon"] = 965,
6267    ["phi"] = 966,
6268    ["chi"] = 967,
6269    ["psi"] = 968,
6270    ["omega"] = 969,
6271    ["thetasym"] = 977,
6272    ["thetav"] = 977,
6273    ["vartheta"] = 977,
6274    ["Upsi"] = 978,
6275    ["upsih"] = 978,
6276    ["phiv"] = 981,
6277    ["straightphi"] = 981,
6278    ["varphi"] = 981,
6279    ["piv"] = 982,
6280    ["varpi"] = 982,
6281    ["Gammad"] = 988,
6282    ["digamma"] = 989,
6283    ["gammad"] = 989,
6284    ["kappav"] = 1008,
6285    ["varkappa"] = 1008,
6286    ["rhov"] = 1009,
6287    ["varrho"] = 1009,
6288    ["epsiv"] = 1013,
6289    ["straightepsilon"] = 1013,
```

```
["varepsilon"] = 1013,
["backepsilon"] = 1014,
["bepsi"] = 1014,
["IOcy"] = 1025,
["DJcy"] = 1026,
["GJcy"] = 1027,
["Jukcy"] = 1028,
["DScy"] = 1029,
["Iukcy"] = 1030,
["YIcy"] = 1031,
["Jsercy"] = 1032,
["LJcy"] = 1033,
["NJcy"] = 1034,
["TSHcy"] = 1035,
["KJcy"] = 1036,
["Ubrcy"] = 1038,
["DZcy"] = 1039,
["Acy"] = 1040,
["Bcy"] = 1041,
["Vcy"] = 1042,
["Gcy"] = 1043,
["Dcy"] = 1044,
["IEcy"] = 1045,
["ZHcy"] = 1046,
["Zcy"] = 1047,
["Icy"] = 1048,
["Jcy"] = 1049,
["Kcy"] = 1050,
["Lcy"] = 1051,
["Mcy"] = 1052,
["Ncy"] = 1053,
["Ocy"] = 1054,
["Pcy"] = 1055,
["Rcy"] = 1056,
["Scy"] = 1057,
["Tcy"] = 1058,
["Ucy"] = 1059,
["Fcy"] = 1060,
["KHcy"] = 1061,
["TScy"] = 1062,
["CHcy"] = 1063,
["SHcy"] = 1064,
["SHCHcy"] = 1065,
["HARDcy"] = 1066,
["Ycy"] = 1067,
["SOFTcy"] = 1068,
["Ecy"] = 1069,
["YUcy"] = 1070,
["YAcy"] = 1071,
```

```
6339    ["acy"] = 1072,
6340    ["bcy"] = 1073,
6341    ["vcy"] = 1074,
6342    ["gcy"] = 1075,
6343    ["dcy"] = 1076,
6344    ["iecy"] = 1077,
6345    ["zhcy"] = 1078,
6346    ["zcy"] = 1079,
6347    ["icy"] = 1080,
6348    ["jcy"] = 1081,
6349    ["kcy"] = 1082,
6350    ["lcy"] = 1083,
6351    ["mcy"] = 1084,
6352    ["ncy"] = 1085,
6353    ["ocy"] = 1086,
6354    ["pcy"] = 1087,
6355    ["rcy"] = 1088,
6356    ["scy"] = 1089,
6357    ["tcy"] = 1090,
6358    ["ucy"] = 1091,
6359    ["fcy"] = 1092,
6360    ["khcy"] = 1093,
6361    ["tscy"] = 1094,
6362    ["chcy"] = 1095,
6363    ["shcy"] = 1096,
6364    ["shchcy"] = 1097,
6365    ["hardcy"] = 1098,
6366    ["ycy"] = 1099,
6367    ["softcy"] = 1100,
6368    ["ecy"] = 1101,
6369    ["yucy"] = 1102,
6370    ["yacy"] = 1103,
6371    ["iocy"] = 1105,
6372    ["djcy"] = 1106,
6373    ["gjcy"] = 1107,
6374    ["jukcy"] = 1108,
6375    ["dscy"] = 1109,
6376    ["iukcy"] = 1110,
6377    ["yicy"] = 1111,
6378    ["jsercy"] = 1112,
6379    ["ljcy"] = 1113,
6380    ["njcy"] = 1114,
6381    ["tshcy"] = 1115,
6382    ["kjcy"] = 1116,
6383    ["ubrcy"] = 1118,
6384    ["dzcy"] = 1119,
6385    ["ensp"] = 8194,
6386    ["emsp"] = 8195,
6387    ["emsp13"] = 8196,
```

```
6388    ["emsp14"] = 8197,
6389    ["numsp"] = 8199,
6390    ["puncsp"] = 8200,
6391    ["ThinSpace"] = 8201,
6392    ["thinsp"] = 8201,
6393    ["VeryThinSpace"] = 8202,
6394    ["hairsp"] = 8202,
6395    ["NegativeMediumSpace"] = 8203,
6396    ["NegativeThickSpace"] = 8203,
6397    ["NegativeThinSpace"] = 8203,
6398    ["NegativeVeryThinSpace"] = 8203,
6399    ["ZeroWidthSpace"] = 8203,
6400    ["zwnj"] = 8204,
6401    ["zwj"] = 8205,
6402    ["lrm"] = 8206,
6403    ["rlm"] = 8207,
6404    ["dash"] = 8208,
6405    ["hyphen"] = 8208,
6406    ["ndash"] = 8211,
6407    ["mdash"] = 8212,
6408    ["horbar"] = 8213,
6409    ["Verbar"] = 8214,
6410    ["Vert"] = 8214,
6411    ["OpenCurlyQuote"] = 8216,
6412    ["lsquo"] = 8216,
6413    ["CloseCurlyQuote"] = 8217,
6414    ["rsquo"] = 8217,
6415    ["rsquor"] = 8217,
6416    ["lsquor"] = 8218,
6417    ["sbquo"] = 8218,
6418    ["OpenCurlyDoubleQuote"] = 8220,
6419    ["ldquo"] = 8220,
6420    ["CloseCurlyDoubleQuote"] = 8221,
6421    ["rdquo"] = 8221,
6422    ["rdquor"] = 8221,
6423    ["bdquo"] = 8222,
6424    ["ldquor"] = 8222,
6425    ["dagger"] = 8224,
6426    ["Dagger"] = 8225,
6427    ["ddagger"] = 8225,
6428    ["bull"] = 8226,
6429    ["bullet"] = 8226,
6430    ["nldr"] = 8229,
6431    ["hellip"] = 8230,
6432    ["mldr"] = 8230,
6433    ["permil"] = 8240,
6434    ["pertenk"] = 8241,
6435    ["prime"] = 8242,
6436    ["Prime"] = 8243,
```

```
6437    ["tprime"] = 8244,
6438    ["backprime"] = 8245,
6439    ["bprime"] = 8245,
6440    ["lsaquo"] = 8249,
6441    ["rsaquo"] = 8250,
6442    ["OverBar"] = 8254,
6443    ["oline"] = 8254,
6444    ["caret"] = 8257,
6445    ["hybull"] = 8259,
6446    ["frasl"] = 8260,
6447    ["bsemi"] = 8271,
6448    ["qprime"] = 8279,
6449    ["MediumSpace"] = 8287,
6450    ["ThickSpace"] = {8287, 8202},
6451    ["NoBreak"] = 8288,
6452    ["ApplyFunction"] = 8289,
6453    ["af"] = 8289,
6454    ["InvisibleTimes"] = 8290,
6455    ["it"] = 8290,
6456    ["InvisibleComma"] = 8291,
6457    ["ic"] = 8291,
6458    ["euro"] = 8364,
6459    ["TripleDot"] = 8411,
6460    ["tdot"] = 8411,
6461    ["DotDot"] = 8412,
6462    ["Copf"] = 8450,
6463    ["complexes"] = 8450,
6464    ["incare"] = 8453,
6465    ["gscr"] = 8458,
6466    ["HilbertSpace"] = 8459,
6467    ["Hscr"] = 8459,
6468    ["hamilt"] = 8459,
6469    ["Hfr"] = 8460,
6470    ["Poincareplane"] = 8460,
6471    ["Hopf"] = 8461,
6472    ["quaternions"] = 8461,
6473    ["planckh"] = 8462,
6474    ["hbar"] = 8463,
6475    ["hslash"] = 8463,
6476    ["planck"] = 8463,
6477    ["plankv"] = 8463,
6478    ["Iscr"] = 8464,
6479    ["imagline"] = 8464,
6480    ["Ifr"] = 8465,
6481    ["Im"] = 8465,
6482    ["image"] = 8465,
6483    ["imagpart"] = 8465,
6484    ["Laplacetrf"] = 8466,
6485    ["Lscr"] = 8466,
```

```
6486    ["lagran"] = 8466,
6487    ["ell"] = 8467,
6488    ["Nopf"] = 8469,
6489    ["naturals"] = 8469,
6490    ["numero"] = 8470,
6491    ["copysr"] = 8471,
6492    ["weierp"] = 8472,
6493    ["wp"] = 8472,
6494    ["Popf"] = 8473,
6495    ["primes"] = 8473,
6496    ["Qopf"] = 8474,
6497    ["rationals"] = 8474,
6498    ["Rscr"] = 8475,
6499    ["realine"] = 8475,
6500    ["Re"] = 8476,
6501    ["Rfr"] = 8476,
6502    ["real"] = 8476,
6503    ["realpart"] = 8476,
6504    ["Ropf"] = 8477,
6505    ["reals"] = 8477,
6506    ["rx"] = 8478,
6507    ["TRADE"] = 8482,
6508    ["trade"] = 8482,
6509    ["Zopf"] = 8484,
6510    ["integers"] = 8484,
6511    ["mho"] = 8487,
6512    ["Zfr"] = 8488,
6513    ["zeetrf"] = 8488,
6514    ["iiota"] = 8489,
6515    ["Bernoullis"] = 8492,
6516    ["Bscr"] = 8492,
6517    ["bernou"] = 8492,
6518    ["Cayleys"] = 8493,
6519    ["Cfr"] = 8493,
6520    ["escr"] = 8495,
6521    ["Escr"] = 8496,
6522    ["expectation"] = 8496,
6523    ["Fouriertrf"] = 8497,
6524    ["Fscr"] = 8497,
6525    ["Mellintrf"] = 8499,
6526    ["Mscr"] = 8499,
6527    ["phmmat"] = 8499,
6528    ["order"] = 8500,
6529    ["orderof"] = 8500,
6530    ["oscr"] = 8500,
6531    ["alefsym"] = 8501,
6532    ["aleph"] = 8501,
6533    ["beth"] = 8502,
6534    ["gimel"] = 8503,
```

```
6535    ["daleth"] = 8504,
6536    ["CapitalDifferentialD"] = 8517,
6537    ["DD"] = 8517,
6538    ["DifferentialD"] = 8518,
6539    ["dd"] = 8518,
6540    ["ExponentialE"] = 8519,
6541    ["ee"] = 8519,
6542    ["exponentiale"] = 8519,
6543    ["ImaginaryI"] = 8520,
6544    ["ii"] = 8520,
6545    ["frac13"] = 8531,
6546    ["frac23"] = 8532,
6547    ["frac15"] = 8533,
6548    ["frac25"] = 8534,
6549    ["frac35"] = 8535,
6550    ["frac45"] = 8536,
6551    ["frac16"] = 8537,
6552    ["frac56"] = 8538,
6553    ["frac18"] = 8539,
6554    ["frac38"] = 8540,
6555    ["frac58"] = 8541,
6556    ["frac78"] = 8542,
6557    ["LeftArrow"] = 8592,
6558    ["ShortLeftArrow"] = 8592,
6559    ["larr"] = 8592,
6560    ["leftarrow"] = 8592,
6561    ["slarr"] = 8592,
6562    ["ShortUpArrow"] = 8593,
6563    ["UpArrow"] = 8593,
6564    ["uarr"] = 8593,
6565    ["uparrow"] = 8593,
6566    ["RightArrow"] = 8594,
6567    ["ShortRightArrow"] = 8594,
6568    ["rarr"] = 8594,
6569    ["rightarrow"] = 8594,
6570    ["srarr"] = 8594,
6571    ["DownArrow"] = 8595,
6572    ["ShortDownArrow"] = 8595,
6573    ["darr"] = 8595,
6574    ["downarrow"] = 8595,
6575    ["LeftRightArrow"] = 8596,
6576    ["harr"] = 8596,
6577    ["leftrightarrow"] = 8596,
6578    ["UpDownArrow"] = 8597,
6579    ["updownarrow"] = 8597,
6580    ["varr"] = 8597,
6581    ["UpperLeftArrow"] = 8598,
6582    ["nwarr"] = 8598,
6583    ["nwarrow"] = 8598,
```

```
6584    ["UpperRightArrow"] = 8599,
6585    ["nearr"] = 8599,
6586    ["nearrow"] = 8599,
6587    ["LowerRightArrow"] = 8600,
6588    ["searr"] = 8600,
6589    ["searrow"] = 8600,
6590    ["LowerLeftArrow"] = 8601,
6591    ["swarr"] = 8601,
6592    ["swarrow"] = 8601,
6593    ["nlarr"] = 8602,
6594    ["nleftarrow"] = 8602,
6595    ["nrarr"] = 8603,
6596    ["nrightarrow"] = 8603,
6597    ["nrarrw"] = {8605, 824},
6598    ["rarrw"] = 8605,
6599    ["rightsquigarrow"] = 8605,
6600    ["Larr"] = 8606,
6601    ["twoheadleftarrow"] = 8606,
6602    ["Uarr"] = 8607,
6603    ["Rarr"] = 8608,
6604    ["twoheadrightarrow"] = 8608,
6605    ["Darr"] = 8609,
6606    ["larrtl"] = 8610,
6607    ["leftarrowtail"] = 8610,
6608    ["rarrtl"] = 8611,
6609    ["rightarrowtail"] = 8611,
6610    ["LeftTeeArrow"] = 8612,
6611    ["mapstoleft"] = 8612,
6612    ["UpTeeArrow"] = 8613,
6613    ["mapstoup"] = 8613,
6614    ["RightTeeArrow"] = 8614,
6615    ["map"] = 8614,
6616    ["mapsto"] = 8614,
6617    ["DownTeeArrow"] = 8615,
6618    ["mapstodown"] = 8615,
6619    ["hookleftarrow"] = 8617,
6620    ["larrhk"] = 8617,
6621    ["hookrightarrow"] = 8618,
6622    ["rarrhk"] = 8618,
6623    ["larrlp"] = 8619,
6624    ["looparrowleft"] = 8619,
6625    ["looparrowright"] = 8620,
6626    ["rarrlp"] = 8620,
6627    ["harrw"] = 8621,
6628    ["leftrightsquigarrow"] = 8621,
6629    ["nharr"] = 8622,
6630    ["nleftrightarrow"] = 8622,
6631    ["Lsh"] = 8624,
6632    ["lsh"] = 8624,
```

```
  ["Rsh"] = 8625,
  ["rsh"] = 8625,
  ["ldsh"] = 8626,
  ["rdsh"] = 8627,
  ["crarr"] = 8629,
  ["cularr"] = 8630,
  ["curvearrowleft"] = 8630,
  ["curarr"] = 8631,
  ["curvearrowright"] = 8631,
  ["circlearrowleft"] = 8634,
  ["olarr"] = 8634,
  ["circlearrowright"] = 8635,
  ["orarr"] = 8635,
  ["LeftVector"] = 8636,
  ["leftharpoonup"] = 8636,
  ["lharu"] = 8636,
  ["DownLeftVector"] = 8637,
  ["leftharpoondown"] = 8637,
  ["lhard"] = 8637,
  ["RightUpVector"] = 8638,
  ["uharr"] = 8638,
  ["upharpoonright"] = 8638,
  ["LeftUpVector"] = 8639,
  ["uharl"] = 8639,
  ["upharpoonleft"] = 8639,
  ["RightVector"] = 8640,
  ["rharu"] = 8640,
  ["rightharpoonup"] = 8640,
  ["DownRightVector"] = 8641,
  ["rhard"] = 8641,
  ["rightharpoondown"] = 8641,
  ["RightDownVector"] = 8642,
  ["dharr"] = 8642,
  ["downharpoonright"] = 8642,
  ["LeftDownVector"] = 8643,
  ["dharl"] = 8643,
  ["downharpoonleft"] = 8643,
  ["RightArrowLeftArrow"] = 8644,
  ["rightleftarrows"] = 8644,
  ["rlarr"] = 8644,
  ["UpArrowDownArrow"] = 8645,
  ["udarr"] = 8645,
  ["LeftArrowRightArrow"] = 8646,
  ["leftrightarrows"] = 8646,
  ["lrarr"] = 8646,
  ["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,
["zigrarr"] = 8669,
["LeftArrowBar"] = 8676,
["larrb"] = 8676,
["RightArrowBar"] = 8677,
["rarrb"] = 8677,
```

```
6731    ["DownArrowUpArrow"] = 8693,
6732    ["duarr"] = 8693,
6733    ["loarr"] = 8701,
6734    ["roarr"] = 8702,
6735    ["hoarr"] = 8703,
6736    ["ForAll"] = 8704,
6737    ["forall"] = 8704,
6738    ["comp"] = 8705,
6739    ["complement"] = 8705,
6740    ["PartialD"] = 8706,
6741    ["npart"] = {8706, 824},
6742    ["part"] = 8706,
6743    ["Exists"] = 8707,
6744    ["exist"] = 8707,
6745    ["NotExists"] = 8708,
6746    ["nexist"] = 8708,
6747    ["nexists"] = 8708,
6748    ["empty"] = 8709,
6749    ["emptyset"] = 8709,
6750    ["emptyv"] = 8709,
6751    ["varnothing"] = 8709,
6752    ["Del"] = 8711,
6753    ["nabla"] = 8711,
6754    ["Element"] = 8712,
6755    ["in"] = 8712,
6756    ["isin"] = 8712,
6757    ["isinv"] = 8712,
6758    ["NotElement"] = 8713,
6759    ["notin"] = 8713,
6760    ["notinva"] = 8713,
6761    ["ReverseElement"] = 8715,
6762    ["SuchThat"] = 8715,
6763    ["ni"] = 8715,
6764    ["niv"] = 8715,
6765    ["NotReverseElement"] = 8716,
6766    ["notni"] = 8716,
6767    ["notniva"] = 8716,
6768    ["Product"] = 8719,
6769    ["prod"] = 8719,
6770    ["Coproduct"] = 8720,
6771    ["coprod"] = 8720,
6772    ["Sum"] = 8721,
6773    ["sum"] = 8721,
6774    ["minus"] = 8722,
6775    ["MinusPlus"] = 8723,
6776    ["mnplus"] = 8723,
6777    ["mp"] = 8723,
6778    ["dotplus"] = 8724,
6779    ["plusdo"] = 8724,
```

```
6780    ["Backslash"] = 8726,
6781    ["setminus"] = 8726,
6782    ["setmn"] = 8726,
6783    ["smallsetminus"] = 8726,
6784    ["ssetmn"] = 8726,
6785    ["lowast"] = 8727,
6786    ["SmallCircle"] = 8728,
6787    ["compfn"] = 8728,
6788    ["Sqrt"] = 8730,
6789    ["radic"] = 8730,
6790    ["Proportional"] = 8733,
6791    ["prop"] = 8733,
6792    ["propto"] = 8733,
6793    ["varpropto"] = 8733,
6794    ["vprop"] = 8733,
6795    ["infin"] = 8734,
6796    ["angrt"] = 8735,
6797    ["ang"] = 8736,
6798    ["angle"] = 8736,
6799    ["nang"] = {8736, 8402},
6800    ["angmsd"] = 8737,
6801    ["measuredangle"] = 8737,
6802    ["angsph"] = 8738,
6803    ["VerticalBar"] = 8739,
6804    ["mid"] = 8739,
6805    ["shortmid"] = 8739,
6806    ["smid"] = 8739,
6807    ["NotVerticalBar"] = 8740,
6808    ["nmid"] = 8740,
6809    ["nshortmid"] = 8740,
6810    ["nsmid"] = 8740,
6811    ["DoubleVerticalBar"] = 8741,
6812    ["par"] = 8741,
6813    ["parallel"] = 8741,
6814    ["shortparallel"] = 8741,
6815    ["spar"] = 8741,
6816    ["NotDoubleVerticalBar"] = 8742,
6817    ["npar"] = 8742,
6818    ["nparallel"] = 8742,
6819    ["nshortparallel"] = 8742,
6820    ["nspar"] = 8742,
6821    ["and"] = 8743,
6822    ["wedge"] = 8743,
6823    ["or"] = 8744,
6824    ["vee"] = 8744,
6825    ["cap"] = 8745,
6826    ["caps"] = {8745, 65024},
6827    ["cup"] = 8746,
6828    ["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,
["acE"] = {8766, 819},
["mstpos"] = 8766,
["acd"] = 8767,
["VerticalTilde"] = 8768,
["wr"] = 8768,
["wreath"] = 8768,
["NotTilde"] = 8769,
["nsim"] = 8769,
["EqualTilde"] = 8770,
["NotEqualTilde"] = {8770, 824},
["eqsim"] = 8770,
```

```
6878    ["esim"] = 8770,
6879    ["nesim"] = {8770, 824},
6880    ["TildeEqual"] = 8771,
6881    ["sime"] = 8771,
6882    ["simeq"] = 8771,
6883    ["NotTildeEqual"] = 8772,
6884    ["nsime"] = 8772,
6885    ["nsimeq"] = 8772,
6886    ["TildeFullEqual"] = 8773,
6887    ["cong"] = 8773,
6888    ["simne"] = 8774,
6889    ["NotTildeFullEqual"] = 8775,
6890    ["ncong"] = 8775,
6891    ["TildeTilde"] = 8776,
6892    ["ap"] = 8776,
6893    ["approx"] = 8776,
6894    ["asymp"] = 8776,
6895    ["thickapprox"] = 8776,
6896    ["thkap"] = 8776,
6897    ["NotTildeTilde"] = 8777,
6898    ["nap"] = 8777,
6899    ["napprox"] = 8777,
6900    ["ape"] = 8778,
6901    ["approxeq"] = 8778,
6902    ["apid"] = 8779,
6903    ["napid"] = {8779, 824},
6904    ["backcong"] = 8780,
6905    ["bcong"] = 8780,
6906    ["CupCap"] = 8781,
6907    ["asympeq"] = 8781,
6908    ["nvap"] = {8781, 8402},
6909    ["Bumpeq"] = 8782,
6910    ["HumpDownHump"] = 8782,
6911    ["NotHumpDownHump"] = {8782, 824},
6912    ["bump"] = 8782,
6913    ["nbump"] = {8782, 824},
6914    ["HumpEqual"] = 8783,
6915    ["NotHumpEqual"] = {8783, 824},
6916    ["bumpe"] = 8783,
6917    ["bumpeq"] = 8783,
6918    ["nbumpe"] = {8783, 824},
6919    ["DotEqual"] = 8784,
6920    ["doteq"] = 8784,
6921    ["esdot"] = 8784,
6922    ["nedot"] = {8784, 824},
6923    ["doteqdot"] = 8785,
6924    ["eDot"] = 8785,
6925    ["efDot"] = 8786,
6926    ["fallingdotseq"] = 8786,
```

```
6927    ["erDot"] = 8787,
6928    ["risingdotseq"] = 8787,
6929    ["Assign"] = 8788,
6930    ["colone"] = 8788,
6931    ["coloneq"] = 8788,
6932    ["ecolon"] = 8789,
6933    ["eqcolon"] = 8789,
6934    ["ecir"] = 8790,
6935    ["eqcirc"] = 8790,
6936    ["circeq"] = 8791,
6937    ["cire"] = 8791,
6938    ["wedgeq"] = 8793,
6939    ["veeeq"] = 8794,
6940    ["triangleq"] = 8796,
6941    ["trie"] = 8796,
6942    ["equest"] = 8799,
6943    ["questeq"] = 8799,
6944    ["NotEqual"] = 8800,
6945    ["ne"] = 8800,
6946    ["Congruent"] = 8801,
6947    ["bnequiv"] = {8801, 8421},
6948    ["equiv"] = 8801,
6949    ["NotCongruent"] = 8802,
6950    ["nequiv"] = 8802,
6951    ["le"] = 8804,
6952    ["leq"] = 8804,
6953    ["nvle"] = {8804, 8402},
6954    ["GreaterEqual"] = 8805,
6955    ["ge"] = 8805,
6956    ["geq"] = 8805,
6957    ["nvge"] = {8805, 8402},
6958    ["LessFullEqual"] = 8806,
6959    ["lE"] = 8806,
6960    ["leqq"] = 8806,
6961    ["nlE"] = {8806, 824},
6962    ["nleqq"] = {8806, 824},
6963    ["GreaterFullEqual"] = 8807,
6964    ["NotGreaterFullEqual"] = {8807, 824},
6965    ["gE"] = 8807,
6966    ["geqq"] = 8807,
6967    ["ngE"] = {8807, 824},
6968    ["ngeqq"] = {8807, 824},
6969    ["lnE"] = 8808,
6970    ["lneqq"] = 8808,
6971    ["lvertneqq"] = {8808, 65024},
6972    ["lvnE"] = {8808, 65024},
6973    ["gnE"] = 8809,
6974    ["gneqq"] = 8809,
6975    ["gvertneqq"] = {8809, 65024},
```

```
["gvnE"] = {8809, 65024},
["Lt"] = 8810,
["NestedLessLess"] = 8810,
["NotLessLess"] = {8810, 824},
["ll"] = 8810,
["nLt"] = {8810, 8402},
["nLtv"] = {8810, 824},
["Gt"] = 8811,
["NestedGreaterGreater"] = 8811,
["NotGreaterGreater"] = {8811, 824},
["gg"] = 8811,
["nGt"] = {8811, 8402},
["nGtv"] = {8811, 824},
["between"] = 8812,
["twixt"] = 8812,
["NotCupCap"] = 8813,
["NotLess"] = 8814,
["nless"] = 8814,
["nlt"] = 8814,
["NotGreater"] = 8815,
["ngt"] = 8815,
["ngtr"] = 8815,
["NotLessEqual"] = 8816,
["nle"] = 8816,
["nleq"] = 8816,
["NotGreaterEqual"] = 8817,
["nge"] = 8817,
["ngeq"] = 8817,
["LessTilde"] = 8818,
["lesssim"] = 8818,
["lsim"] = 8818,
["GreaterTilde"] = 8819,
["gsim"] = 8819,
["gtrsim"] = 8819,
["NotLessTilde"] = 8820,
["nlsim"] = 8820,
["NotGreaterTilde"] = 8821,
["ngsim"] = 8821,
["LessGreater"] = 8822,
["lessgtr"] = 8822,
["lg"] = 8822,
["GreaterLess"] = 8823,
["gl"] = 8823,
["gtrless"] = 8823,
["NotLessGreater"] = 8824,
["ntlg"] = 8824,
["NotGreaterLess"] = 8825,
["ntgl"] = 8825,
["Precedes"] = 8826,
```

```
["pr"] = 8826,
["prec"] = 8826,
["Succeeds"] = 8827,
["sc"] = 8827,
["succ"] = 8827,
["PrecedesSlantEqual"] = 8828,
["prcue"] = 8828,
["preccurlyeq"] = 8828,
["SucceedsSlantEqual"] = 8829,
["sccue"] = 8829,
["succcurlyeq"] = 8829,
["PrecedesTilde"] = 8830,
["precsim"] = 8830,
["prsim"] = 8830,
["NotSucceedsTilde"] = {8831, 824},
["SucceedsTilde"] = 8831,
["scsim"] = 8831,
["succsim"] = 8831,
["NotPrecedes"] = 8832,
["npr"] = 8832,
["nprec"] = 8832,
["NotSucceeds"] = 8833,
["nsc"] = 8833,
["nsucc"] = 8833,
["NotSubset"] = {8834, 8402},
["nsubset"] = {8834, 8402},
["sub"] = 8834,
["subset"] = 8834,
["vnsub"] = {8834, 8402},
["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},
["SquareUnion"] = 8852,
["sqcup"] = 8852,
["sqcups"] = {8852, 65024},
["CirclePlus"] = 8853,
["oplus"] = 8853,
["CircleMinus"] = 8854,
["ominus"] = 8854,
["CircleTimes"] = 8855,
["otimes"] = 8855,
["osol"] = 8856,
["CircleDot"] = 8857,
["odot"] = 8857,
["circledcirc"] = 8858,
["ocir"] = 8858,
["circledast"] = 8859,
["oast"] = 8859,
["circleddash"] = 8861,
["odash"] = 8861,
["boxplus"] = 8862,
["plusb"] = 8862,
["boxminus"] = 8863,
```

```
7123    ["minusb"] = 8863,
7124    ["boxtimes"] = 8864,
7125    ["timesb"] = 8864,
7126    ["dotsquare"] = 8865,
7127    ["sdotb"] = 8865,
7128    ["RightTee"] = 8866,
7129    ["vdash"] = 8866,
7130    ["LeftTee"] = 8867,
7131    ["dashv"] = 8867,
7132    ["DownTee"] = 8868,
7133    ["top"] = 8868,
7134    ["UpTee"] = 8869,
7135    ["bot"] = 8869,
7136    ["bottom"] = 8869,
7137    ["perp"] = 8869,
7138    ["models"] = 8871,
7139    ["DoubleRightTee"] = 8872,
7140    ["vDash"] = 8872,
7141    ["Vdash"] = 8873,
7142    ["Vvdash"] = 8874,
7143    ["VDash"] = 8875,
7144    ["nvdash"] = 8876,
7145    ["nvDash"] = 8877,
7146    ["nVdash"] = 8878,
7147    ["nVDash"] = 8879,
7148    ["prurel"] = 8880,
7149    ["LeftTriangle"] = 8882,
7150    ["vartriangleleft"] = 8882,
7151    ["vltri"] = 8882,
7152    ["RightTriangle"] = 8883,
7153    ["vartriangleright"] = 8883,
7154    ["vrtri"] = 8883,
7155    ["LeftTriangleEqual"] = 8884,
7156    ["ltrie"] = 8884,
7157    ["nvltrie"] = {8884, 8402},
7158    ["trianglelefteq"] = 8884,
7159    ["RightTriangleEqual"] = 8885,
7160    ["nvrtrie"] = {8885, 8402},
7161    ["rtrie"] = 8885,
7162    ["trianglerighteq"] = 8885,
7163    ["origof"] = 8886,
7164    ["imof"] = 8887,
7165    ["multimap"] = 8888,
7166    ["mumap"] = 8888,
7167    ["hercon"] = 8889,
7168    ["intcal"] = 8890,
7169    ["intercal"] = 8890,
7170    ["veebar"] = 8891,
7171    ["barvee"] = 8893,
```

```
["angrtvb"] = 8894,
["lrtri"] = 8895,
["Wedge"] = 8896,
["bigwedge"] = 8896,
["xwedge"] = 8896,
["Vee"] = 8897,
["bigvee"] = 8897,
["xvee"] = 8897,
["Intersection"] = 8898,
["bigcap"] = 8898,
["xcap"] = 8898,
["Union"] = 8899,
["bigcup"] = 8899,
["xcup"] = 8899,
["Diamond"] = 8900,
["diam"] = 8900,
["diamond"] = 8900,
["sdot"] = 8901,
["Star"] = 8902,
["sstarf"] = 8902,
["divideontimes"] = 8903,
["divonx"] = 8903,
["bowtie"] = 8904,
["ltimes"] = 8905,
["rtimes"] = 8906,
["leftthreetimes"] = 8907,
["lthree"] = 8907,
["rightthreetimes"] = 8908,
["rthree"] = 8908,
["backsimeq"] = 8909,
["bsime"] = 8909,
["curlyvee"] = 8910,
["cuvee"] = 8910,
["curlywedge"] = 8911,
["cuwed"] = 8911,
["Sub"] = 8912,
["Subset"] = 8912,
["Sup"] = 8913,
["Supset"] = 8913,
["Cap"] = 8914,
["Cup"] = 8915,
["fork"] = 8916,
["pitchfork"] = 8916,
["epar"] = 8917,
["lessdot"] = 8918,
["ltdot"] = 8918,
["gtdot"] = 8919,
["gtrdot"] = 8919,
["Ll"] = 8920,
```

```
["nLl"] = {8920, 824},
["Gg"] = 8921,
["ggg"] = 8921,
["nGg"] = {8921, 824},
["LessEqualGreater"] = 8922,
["leg"] = 8922,
["lesg"] = {8922, 65024},
["lesseqgtr"] = 8922,
["GreaterEqualLess"] = 8923,
["gel"] = 8923,
["gesl"] = {8923, 65024},
["gtreqless"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["NotPrecedesSlantEqual"] = 8928,
["nprcue"] = 8928,
["NotSucceedsSlantEqual"] = 8929,
["nsccue"] = 8929,
["NotSquareSubsetEqual"] = 8930,
["nsqsube"] = 8930,
["NotSquareSupersetEqual"] = 8931,
["nsqsupe"] = 8931,
["lnsim"] = 8934,
["gnsim"] = 8935,
["precnsim"] = 8936,
["prnsim"] = 8936,
["scnsim"] = 8937,
["succnsim"] = 8937,
["NotLeftTriangle"] = 8938,
["nltri"] = 8938,
["ntriangleleft"] = 8938,
["NotRightTriangle"] = 8939,
["nrtri"] = 8939,
["ntriangleright"] = 8939,
["NotLeftTriangleEqual"] = 8940,
["nltrie"] = 8940,
["ntrianglelefteq"] = 8940,
["NotRightTriangleEqual"] = 8941,
["nrtrie"] = 8941,
["ntrianglerighteq"] = 8941,
["vellip"] = 8942,
["ctdot"] = 8943,
["utdot"] = 8944,
["dtdot"] = 8945,
["disin"] = 8946,
["isinsv"] = 8947,
["isins"] = 8948,
```

```
7270    ["isindot"] = 8949,
7271    ["notindot"] = {8949, 824},
7272    ["notinvc"] = 8950,
7273    ["notinvb"] = 8951,
7274    ["isinE"] = 8953,
7275    ["notinE"] = {8953, 824},
7276    ["nisd"] = 8954,
7277    ["xnis"] = 8955,
7278    ["nis"] = 8956,
7279    ["notnivc"] = 8957,
7280    ["notnivb"] = 8958,
7281    ["barwed"] = 8965,
7282    ["barwedge"] = 8965,
7283    ["Barwed"] = 8966,
7284    ["doublebarwedge"] = 8966,
7285    ["LeftCeiling"] = 8968,
7286    ["lceil"] = 8968,
7287    ["RightCeiling"] = 8969,
7288    ["rceil"] = 8969,
7289    ["LeftFloor"] = 8970,
7290    ["lfloor"] = 8970,
7291    ["RightFloor"] = 8971,
7292    ["rfloor"] = 8971,
7293    ["drcrop"] = 8972,
7294    ["dlcrop"] = 8973,
7295    ["urcrop"] = 8974,
7296    ["ulcrop"] = 8975,
7297    ["bnot"] = 8976,
7298    ["profline"] = 8978,
7299    ["profsurf"] = 8979,
7300    ["telrec"] = 8981,
7301    ["target"] = 8982,
7302    ["ulcorn"] = 8988,
7303    ["ulcorner"] = 8988,
7304    ["urcorn"] = 8989,
7305    ["urcorner"] = 8989,
7306    ["dlcorn"] = 8990,
7307    ["llcorner"] = 8990,
7308    ["drcorn"] = 8991,
7309    ["lrcorner"] = 8991,
7310    ["frown"] = 8994,
7311    ["sfrown"] = 8994,
7312    ["smile"] = 8995,
7313    ["ssmile"] = 8995,
7314    ["cylcty"] = 9005,
7315    ["profalar"] = 9006,
7316    ["topbot"] = 9014,
7317    ["ovbar"] = 9021,
7318    ["solbar"] = 9023,
```

```
7319    ["angzarr"] = 9084,
7320    ["lmoust"] = 9136,
7321    ["lmoustache"] = 9136,
7322    ["rmoust"] = 9137,
7323    ["rmoustache"] = 9137,
7324    ["OverBracket"] = 9140,
7325    ["tbrk"] = 9140,
7326    ["UnderBracket"] = 9141,
7327    ["bbrk"] = 9141,
7328    ["bbrktbrk"] = 9142,
7329    ["OverParenthesis"] = 9180,
7330    ["UnderParenthesis"] = 9181,
7331    ["OverBrace"] = 9182,
7332    ["UnderBrace"] = 9183,
7333    ["trpezium"] = 9186,
7334    ["elinters"] = 9191,
7335    ["blank"] = 9251,
7336    ["circledS"] = 9416,
7337    ["oS"] = 9416,
7338    ["HorizontalLine"] = 9472,
7339    ["boxh"] = 9472,
7340    ["boxv"] = 9474,
7341    ["boxdr"] = 9484,
7342    ["boxdl"] = 9488,
7343    ["boxur"] = 9492,
7344    ["boxul"] = 9496,
7345    ["boxvr"] = 9500,
7346    ["boxvl"] = 9508,
7347    ["boxhd"] = 9516,
7348    ["boxhu"] = 9524,
7349    ["boxvh"] = 9532,
7350    ["boxH"] = 9552,
7351    ["boxV"] = 9553,
7352    ["boxdR"] = 9554,
7353    ["boxDr"] = 9555,
7354    ["boxDR"] = 9556,
7355    ["boxdL"] = 9557,
7356    ["boxDl"] = 9558,
7357    ["boxDL"] = 9559,
7358    ["boxuR"] = 9560,
7359    ["boxUr"] = 9561,
7360    ["boxUR"] = 9562,
7361    ["boxuL"] = 9563,
7362    ["boxUl"] = 9564,
7363    ["boxUL"] = 9565,
7364    ["boxvR"] = 9566,
7365    ["boxVr"] = 9567,
7366    ["boxVR"] = 9568,
7367    ["boxvL"] = 9569,
```

```
7368    ["boxVl"] = 9570,
7369    ["boxVL"] = 9571,
7370    ["boxHd"] = 9572,
7371    ["boxhD"] = 9573,
7372    ["boxHD"] = 9574,
7373    ["boxHu"] = 9575,
7374    ["boxhU"] = 9576,
7375    ["boxHU"] = 9577,
7376    ["boxvH"] = 9578,
7377    ["boxVh"] = 9579,
7378    ["boxVH"] = 9580,
7379    ["uhblk"] = 9600,
7380    ["lhblk"] = 9604,
7381    ["block"] = 9608,
7382    ["blk14"] = 9617,
7383    ["blk12"] = 9618,
7384    ["blk34"] = 9619,
7385    ["Square"] = 9633,
7386    ["squ"] = 9633,
7387    ["square"] = 9633,
7388    ["FilledVerySmallSquare"] = 9642,
7389    ["blacksquare"] = 9642,
7390    ["squarf"] = 9642,
7391    ["squf"] = 9642,
7392    ["EmptyVerySmallSquare"] = 9643,
7393    ["rect"] = 9645,
7394    ["marker"] = 9646,
7395    ["fltns"] = 9649,
7396    ["bigtriangleup"] = 9651,
7397    ["xutri"] = 9651,
7398    ["blacktriangle"] = 9652,
7399    ["utrif"] = 9652,
7400    ["triangle"] = 9653,
7401    ["utri"] = 9653,
7402    ["blacktriangleright"] = 9656,
7403    ["rtrif"] = 9656,
7404    ["rtri"] = 9657,
7405    ["triangleright"] = 9657,
7406    ["bigtriangledown"] = 9661,
7407    ["xdtri"] = 9661,
7408    ["blacktriangledown"] = 9662,
7409    ["dtrif"] = 9662,
7410    ["dtri"] = 9663,
7411    ["triangledown"] = 9663,
7412    ["blacktriangleleft"] = 9666,
7413    ["ltrif"] = 9666,
7414    ["ltri"] = 9667,
7415    ["triangleleft"] = 9667,
7416    ["loz"] = 9674,
```

```
7417    ["lozenge"] = 9674,
7418    ["cir"] = 9675,
7419    ["tridot"] = 9708,
7420    ["bigcirc"] = 9711,
7421    ["xcirc"] = 9711,
7422    ["ultri"] = 9720,
7423    ["urtri"] = 9721,
7424    ["lltri"] = 9722,
7425    ["EmptySmallSquare"] = 9723,
7426    ["FilledSmallSquare"] = 9724,
7427    ["bigstar"] = 9733,
7428    ["starf"] = 9733,
7429    ["star"] = 9734,
7430    ["phone"] = 9742,
7431    ["female"] = 9792,
7432    ["male"] = 9794,
7433    ["spades"] = 9824,
7434    ["spadesuit"] = 9824,
7435    ["clubs"] = 9827,
7436    ["clubsuit"] = 9827,
7437    ["hearts"] = 9829,
7438    ["heartsuit"] = 9829,
7439    ["diamondsuit"] = 9830,
7440    ["diams"] = 9830,
7441    ["sung"] = 9834,
7442    ["flat"] = 9837,
7443    ["natur"] = 9838,
7444    ["natural"] = 9838,
7445    ["sharp"] = 9839,
7446    ["check"] = 10003,
7447    ["checkmark"] = 10003,
7448    ["cross"] = 10007,
7449    ["malt"] = 10016,
7450    ["maltese"] = 10016,
7451    ["sext"] = 10038,
7452    ["VerticalSeparator"] = 10072,
7453    ["lbbrk"] = 10098,
7454    ["rbbrk"] = 10099,
7455    ["bsolhsub"] = 10184,
7456    ["suphsol"] = 10185,
7457    ["LeftDoubleBracket"] = 10214,
7458    ["lobrk"] = 10214,
7459    ["RightDoubleBracket"] = 10215,
7460    ["robrk"] = 10215,
7461    ["LeftAngleBracket"] = 10216,
7462    ["lang"] = 10216,
7463    ["langle"] = 10216,
7464    ["RightAngleBracket"] = 10217,
7465    ["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,
```

```
7515    ["rarrbfs"] = 10528,
7516    ["nwarhk"] = 10531,
7517    ["nearhk"] = 10532,
7518    ["hksearow"] = 10533,
7519    ["searhk"] = 10533,
7520    ["hkswarow"] = 10534,
7521    ["swarhk"] = 10534,
7522    ["nwnear"] = 10535,
7523    ["nesear"] = 10536,
7524    ["toea"] = 10536,
7525    ["seswar"] = 10537,
7526    ["tosa"] = 10537,
7527    ["swnwar"] = 10538,
7528    ["nrarrc"] = {10547, 824},
7529    ["rarrc"] = 10547,
7530    ["cudarrr"] = 10549,
7531    ["ldca"] = 10550,
7532    ["rdca"] = 10551,
7533    ["cudarrl"] = 10552,
7534    ["larrpl"] = 10553,
7535    ["curarrm"] = 10556,
7536    ["cularrp"] = 10557,
7537    ["rarrpl"] = 10565,
7538    ["harrcir"] = 10568,
7539    ["Uarrocir"] = 10569,
7540    ["lurdshar"] = 10570,
7541    ["ldrushar"] = 10571,
7542    ["LeftRightVector"] = 10574,
7543    ["RightUpDownVector"] = 10575,
7544    ["DownLeftRightVector"] = 10576,
7545    ["LeftUpDownVector"] = 10577,
7546    ["LeftVectorBar"] = 10578,
7547    ["RightVectorBar"] = 10579,
7548    ["RightUpVectorBar"] = 10580,
7549    ["RightDownVectorBar"] = 10581,
7550    ["DownLeftVectorBar"] = 10582,
7551    ["DownRightVectorBar"] = 10583,
7552    ["LeftUpVectorBar"] = 10584,
7553    ["LeftDownVectorBar"] = 10585,
7554    ["LeftTeeVector"] = 10586,
7555    ["RightTeeVector"] = 10587,
7556    ["RightUpTeeVector"] = 10588,
7557    ["RightDownTeeVector"] = 10589,
7558    ["DownLeftTeeVector"] = 10590,
7559    ["DownRightTeeVector"] = 10591,
7560    ["LeftUpTeeVector"] = 10592,
7561    ["LeftDownTeeVector"] = 10593,
7562    ["lHar"] = 10594,
7563    ["uHar"] = 10595,
```

```
["rHar"] = 10596,
["dHar"] = 10597,
["luruhar"] = 10598,
["ldrdhar"] = 10599,
["ruluhar"] = 10600,
["rdldhar"] = 10601,
["lharul"] = 10602,
["llhard"] = 10603,
["rharul"] = 10604,
["lrhard"] = 10605,
["UpEquilibrium"] = 10606,
["udhar"] = 10606,
["ReverseUpEquilibrium"] = 10607,
["duhar"] = 10607,
["RoundImplies"] = 10608,
["erarr"] = 10609,
["simrarr"] = 10610,
["larrsim"] = 10611,
["rarrsim"] = 10612,
["rarrap"] = 10613,
["ltlarr"] = 10614,
["gtrarr"] = 10616,
["subrarr"] = 10617,
["suplarr"] = 10619,
["lfisht"] = 10620,
["rfisht"] = 10621,
["ufisht"] = 10622,
["dfisht"] = 10623,
["lopar"] = 10629,
["ropar"] = 10630,
["lbrke"] = 10635,
["rbrke"] = 10636,
["lbrkslu"] = 10637,
["rbrksld"] = 10638,
["lbrksld"] = 10639,
["rbrkslu"] = 10640,
["langd"] = 10641,
["rangd"] = 10642,
["lparlt"] = 10643,
["rpargt"] = 10644,
["gtlPar"] = 10645,
["ltrPar"] = 10646,
["vzigzag"] = 10650,
["vangrt"] = 10652,
["angrtvbd"] = 10653,
["ange"] = 10660,
["range"] = 10661,
["dwangle"] = 10662,
["uwangle"] = 10663,
```

```
["angmsdaa"] = 10664,
["angmsdab"] = 10665,
["angmsdac"] = 10666,
["angmsdad"] = 10667,
["angmsdae"] = 10668,
["angmsdaf"] = 10669,
["angmsdag"] = 10670,
["angmsdah"] = 10671,
["bemptyv"] = 10672,
["demptyv"] = 10673,
["cemptyv"] = 10674,
["raemptyv"] = 10675,
["laemptyv"] = 10676,
["ohbar"] = 10677,
["omid"] = 10678,
["opar"] = 10679,
["operp"] = 10681,
["olcross"] = 10683,
["odsold"] = 10684,
["olcir"] = 10686,
["ofcir"] = 10687,
["olt"] = 10688,
["ogt"] = 10689,
["cirscir"] = 10690,
["cirE"] = 10691,
["solb"] = 10692,
["bsolb"] = 10693,
["boxbox"] = 10697,
["trisb"] = 10701,
["rtriltri"] = 10702,
["LeftTriangleBar"] = 10703,
["NotLeftTriangleBar"] = {10703, 824},
["NotRightTriangleBar"] = {10704, 824},
["RightTriangleBar"] = 10704,
["iinfin"] = 10716,
["infintie"] = 10717,
["nvinfin"] = 10718,
["eparsl"] = 10723,
["smeparsl"] = 10724,
["eqvparsl"] = 10725,
["blacklozenge"] = 10731,
["lozf"] = 10731,
["RuleDelayed"] = 10740,
["dsol"] = 10742,
["bigodot"] = 10752,
["xodot"] = 10752,
["bigoplus"] = 10753,
["xoplus"] = 10753,
["bigotimes"] = 10754,
```

```
7662    ["xotime"] = 10754,
7663    ["biguplus"] = 10756,
7664    ["xuplus"] = 10756,
7665    ["bigsqcup"] = 10758,
7666    ["xsqcup"] = 10758,
7667    ["iiiint"] = 10764,
7668    ["qint"] = 10764,
7669    ["fpartint"] = 10765,
7670    ["cirfnint"] = 10768,
7671    ["awint"] = 10769,
7672    ["rppolint"] = 10770,
7673    ["scpolint"] = 10771,
7674    ["npolint"] = 10772,
7675    ["pointint"] = 10773,
7676    ["quatint"] = 10774,
7677    ["intlarhk"] = 10775,
7678    ["pluscir"] = 10786,
7679    ["plusacir"] = 10787,
7680    ["simplus"] = 10788,
7681    ["plusdu"] = 10789,
7682    ["plussim"] = 10790,
7683    ["plustwo"] = 10791,
7684    ["mcomma"] = 10793,
7685    ["minusdu"] = 10794,
7686    ["loplus"] = 10797,
7687    ["roplus"] = 10798,
7688    ["Cross"] = 10799,
7689    ["timesd"] = 10800,
7690    ["timesbar"] = 10801,
7691    ["smashp"] = 10803,
7692    ["lotimes"] = 10804,
7693    ["rotimes"] = 10805,
7694    ["otimesas"] = 10806,
7695    ["Otimes"] = 10807,
7696    ["odiv"] = 10808,
7697    ["triplus"] = 10809,
7698    ["triminus"] = 10810,
7699    ["tritime"] = 10811,
7700    ["intprod"] = 10812,
7701    ["iprod"] = 10812,
7702    ["amalg"] = 10815,
7703    ["capdot"] = 10816,
7704    ["ncup"] = 10818,
7705    ["ncap"] = 10819,
7706    ["capand"] = 10820,
7707    ["cupor"] = 10821,
7708    ["cupcap"] = 10822,
7709    ["capcup"] = 10823,
7710    ["cupbrcap"] = 10824,
```

```
7711    ["capbrcup"] = 10825,
7712    ["cupcup"] = 10826,
7713    ["capcap"] = 10827,
7714    ["ccups"] = 10828,
7715    ["ccaps"] = 10829,
7716    ["ccupssm"] = 10832,
7717    ["And"] = 10835,
7718    ["Or"] = 10836,
7719    ["andand"] = 10837,
7720    ["oror"] = 10838,
7721    ["orslope"] = 10839,
7722    ["andslope"] = 10840,
7723    ["andv"] = 10842,
7724    ["orv"] = 10843,
7725    ["andd"] = 10844,
7726    ["ord"] = 10845,
7727    ["wedbar"] = 10847,
7728    ["sdote"] = 10854,
7729    ["simdot"] = 10858,
7730    ["congdot"] = 10861,
7731    ["ncongdot"] = {10861, 824},
7732    ["easter"] = 10862,
7733    ["apacir"] = 10863,
7734    ["apE"] = 10864,
7735    ["napE"] = {10864, 824},
7736    ["eplus"] = 10865,
7737    ["pluse"] = 10866,
7738    ["Esim"] = 10867,
7739    ["Colone"] = 10868,
7740    ["Equal"] = 10869,
7741    ["ddotseq"] = 10871,
7742    ["eDDot"] = 10871,
7743    ["equivDD"] = 10872,
7744    ["ltcir"] = 10873,
7745    ["gtcir"] = 10874,
7746    ["ltquest"] = 10875,
7747    ["gtquest"] = 10876,
7748    ["LessSlantEqual"] = 10877,
7749    ["NotLessSlantEqual"] = {10877, 824},
7750    ["leqslant"] = 10877,
7751    ["les"] = 10877,
7752    ["nleqslant"] = {10877, 824},
7753    ["nles"] = {10877, 824},
7754    ["GreaterSlantEqual"] = 10878,
7755    ["NotGreaterSlantEqual"] = {10878, 824},
7756    ["geqslant"] = 10878,
7757    ["ges"] = 10878,
7758    ["ngeqslant"] = {10878, 824},
7759    ["nges"] = {10878, 824},
```

```
7760    ["lesdot"] = 10879,
7761    ["gesdot"] = 10880,
7762    ["lesdoto"] = 10881,
7763    ["gesdoto"] = 10882,
7764    ["lesdotor"] = 10883,
7765    ["gesdotol"] = 10884,
7766    ["lap"] = 10885,
7767    ["lessapprox"] = 10885,
7768    ["gap"] = 10886,
7769    ["gtrapprox"] = 10886,
7770    ["lne"] = 10887,
7771    ["lneq"] = 10887,
7772    ["gne"] = 10888,
7773    ["gneq"] = 10888,
7774    ["lnap"] = 10889,
7775    ["lnapprox"] = 10889,
7776    ["gnap"] = 10890,
7777    ["gnapprox"] = 10890,
7778    ["lEg"] = 10891,
7779    ["lesseqqgtr"] = 10891,
7780    ["gEl"] = 10892,
7781    ["gtreqqless"] = 10892,
7782    ["lsime"] = 10893,
7783    ["gsime"] = 10894,
7784    ["lsimg"] = 10895,
7785    ["gsiml"] = 10896,
7786    ["lgE"] = 10897,
7787    ["glE"] = 10898,
7788    ["lesges"] = 10899,
7789    ["gesles"] = 10900,
7790    ["els"] = 10901,
7791    ["eqslantless"] = 10901,
7792    ["egs"] = 10902,
7793    ["eqslantgtr"] = 10902,
7794    ["elsdot"] = 10903,
7795    ["egsdot"] = 10904,
7796    ["el"] = 10905,
7797    ["eg"] = 10906,
7798    ["siml"] = 10909,
7799    ["simg"] = 10910,
7800    ["simlE"] = 10911,
7801    ["simgE"] = 10912,
7802    ["LessLess"] = 10913,
7803    ["NotNestedLessLess"] = {10913, 824},
7804    ["GreaterGreater"] = 10914,
7805    ["NotNestedGreaterGreater"] = {10914, 824},
7806    ["glj"] = 10916,
7807    ["gla"] = 10917,
7808    ["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,
```

```
7907    ["ffllig"] = 64260,
7908    ["Ascr"] = 119964,
7909    ["Cscr"] = 119966,
7910    ["Dscr"] = 119967,
7911    ["Gscr"] = 119970,
7912    ["Jscr"] = 119973,
7913    ["Kscr"] = 119974,
7914    ["Nscr"] = 119977,
7915    ["Oscr"] = 119978,
7916    ["Pscr"] = 119979,
7917    ["Qscr"] = 119980,
7918    ["Sscr"] = 119982,
7919    ["Tscr"] = 119983,
7920    ["Uscr"] = 119984,
7921    ["Vscr"] = 119985,
7922    ["Wscr"] = 119986,
7923    ["Xscr"] = 119987,
7924    ["Yscr"] = 119988,
7925    ["Zscr"] = 119989,
7926    ["ascr"] = 119990,
7927    ["bscr"] = 119991,
7928    ["cscr"] = 119992,
7929    ["dscr"] = 119993,
7930    ["fscr"] = 119995,
7931    ["hscr"] = 119997,
7932    ["iscr"] = 119998,
7933    ["jscr"] = 119999,
7934    ["kscr"] = 120000,
7935    ["lscr"] = 120001,
7936    ["mscr"] = 120002,
7937    ["nscr"] = 120003,
7938    ["pscr"] = 120005,
7939    ["qscr"] = 120006,
7940    ["rscr"] = 120007,
7941    ["sscr"] = 120008,
7942    ["tscr"] = 120009,
7943    ["uscr"] = 120010,
7944    ["vscr"] = 120011,
7945    ["wscr"] = 120012,
7946    ["xscr"] = 120013,
7947    ["yscr"] = 120014,
7948    ["zscr"] = 120015,
7949    ["Afr"] = 120068,
7950    ["Bfr"] = 120069,
7951    ["Dfr"] = 120071,
7952    ["Efr"] = 120072,
7953    ["Ffr"] = 120073,
7954    ["Gfr"] = 120074,
7955    ["Jfr"] = 120077,
```

```
7956    ["Kfr"] = 120078,
7957    ["Lfr"] = 120079,
7958    ["Mfr"] = 120080,
7959    ["Nfr"] = 120081,
7960    ["Ofr"] = 120082,
7961    ["Pfr"] = 120083,
7962    ["Qfr"] = 120084,
7963    ["Sfr"] = 120086,
7964    ["Tfr"] = 120087,
7965    ["Ufr"] = 120088,
7966    ["Vfr"] = 120089,
7967    ["Wfr"] = 120090,
7968    ["Xfr"] = 120091,
7969    ["Yfr"] = 120092,
7970    ["afr"] = 120094,
7971    ["bfr"] = 120095,
7972    ["cfr"] = 120096,
7973    ["dfr"] = 120097,
7974    ["efr"] = 120098,
7975    ["ffr"] = 120099,
7976    ["gfr"] = 120100,
7977    ["hfr"] = 120101,
7978    ["ifr"] = 120102,
7979    ["jfr"] = 120103,
7980    ["kfr"] = 120104,
7981    ["lfr"] = 120105,
7982    ["mfr"] = 120106,
7983    ["nfr"] = 120107,
7984    ["ofr"] = 120108,
7985    ["pfr"] = 120109,
7986    ["qfr"] = 120110,
7987    ["rfr"] = 120111,
7988    ["sfr"] = 120112,
7989    ["tfr"] = 120113,
7990    ["ufr"] = 120114,
7991    ["vfr"] = 120115,
7992    ["wfr"] = 120116,
7993    ["xfr"] = 120117,
7994    ["yfr"] = 120118,
7995    ["zfr"] = 120119,
7996    ["Aopf"] = 120120,
7997    ["Bopf"] = 120121,
7998    ["Dopf"] = 120123,
7999    ["Eopf"] = 120124,
8000    ["Fopf"] = 120125,
8001    ["Gopf"] = 120126,
8002    ["Iopf"] = 120128,
8003    ["Jopf"] = 120129,
8004    ["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 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 utf8.char(n)
end
```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_c` 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 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, utf8.char(code_point))
    end
  return table.concat(char_table)
end
```

### 3.1.4 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.

```
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.

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

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

```
  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.

```
  self.flatten_inlines = false
```

#### 3.1.4.1 Slicing

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

```
  local slice_specifiers = {}
  for specifier in options.slice:gmatch("[^%s]+") do
    table.insert(slice_specifiers, specifier)
  end

  if #slice_specifiers == 2 then
    self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
    local slice_begin_type = self.slice_begin:sub(1, 1)
    if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
      self.slice_begin = "^" .. self.slice_begin
    end
    local slice_end_type = self.slice_end:sub(1, 1)
    if slice_end_type ~= "^" and slice_end_type ~= "$" 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
```

```
8110    else
8111      self.is_writing = false
8112    end
```

#### 3.1.4.2 Basic Formatter Variables and Functions

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

```
8113    self.space = " "
```

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

```
8114    self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
8115    function self.plain(s)
8116      return s
8117    end
```

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

```
8118    function self.paragraph(s)
8119      if not self.is_writing then return "" end
8120      return s
8121    end
```

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

```
8122    self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
8123    function self.interblocksep()
8124      if not self.is_writing then return "" end
8125      return self.interblocksep_text
8126    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.

```
8127    self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
8128    function self.paragraphsep()
8129      if not self.is_writing then return "" end
8130      return self.paragraphsep_text
8131    end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
8132    self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
8133    function self.undosep()
8134      if not self.is_writing then return "" end
8135      return self.undosep_text
8136    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
```

#### 3.1.4.3 Escaping Special Characters

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_st` 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.

```
  self.escape = escape_typographic_text
  self.math = escape_minimal
  if options.hybrid then
    self.identifier = escape_minimal
    self.string = escape_minimal
```

```
8201      self.uri = escape_minimal
8202      self.infostring = escape_minimal
8203    else
8204      self.identifier = escape_programmatic_text
8205      self.string = escape_typographic_text
8206      self.uri = escape_programmatic_text
8207      self.infostring = escape_programmatic_text
8208    end
```

#### 3.1.4.4 Formatters of Warnings and Errors

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```
8209    function self.warning(t, m)
8210      return {"\\markdownRendererWarning{", self.escape(t), "}{",
8211              escape_minimal(t), "}{", self.escape(m or ""), "}{",
8212              escape_minimal(m or ""), "}"}
8213    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.

```
8214    function self.error(t, m)
8215      return {"\\markdownRendererError{", self.escape(t), "}{",
8216              escape_minimal(t), "}{", self.escape(m or ""), "}{",
8217              escape_minimal(m or ""), "}"}
8218    end
```

#### 3.1.4.5 Formatter of Code Spans

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

```
8219    function self.code(s, attributes)
8220      if self.flatten_inlines then return s end
8221      local buf = {}
8222      if attributes ~= nil then
8223        table.insert(buf,
8224                     "\\markdownRendererCodeSpanAttributeContextBegin\n")
8225        table.insert(buf, self.attributes(attributes))
8226      end
8227      table.insert(buf,
8228                   {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
8229      if attributes ~= nil then
8230        table.insert(buf,
8231                     "\\markdownRendererCodeSpanAttributeContextEnd{}")
8232      end
8233      return buf
8234    end
```

#### 3.1.4.6 Formatter of Hyperlinks

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
```

#### 3.1.4.7 Formatter of Images

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),"}",
                     "{",tit,"}"})
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererImageAttributeContextEnd{}")
  end
  return buf
end
```

#### 3.1.4.8 Formatters of Lists

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.

```
function self.bulletlist(items,tight)
  if not self.is_writing then return "" end
  local buffer = {}
  for _,item in ipairs(items) do
    if item ~= "" then
      buffer[#buffer + 1] = self.bulletitem(item)
    end
  end
  local contents = util.intersperse(buffer,"\n")
  if tight and options.tightLists then
    return {"\\markdownRendererUlBeginTight\n",contents,
      "\n\\markdownRendererUlEndTight "}
  else
    return {"\\markdownRendererUlBegin\n",contents,
      "\n\\markdownRendererUlEnd "}
  end
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.

```
function self.bulletitem(s)
  return {"\\markdownRendererUlItem ",s,
          "\\markdownRendererUlItemEnd "}
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.

```
function self.orderedlist(items,tight,startnum)
  if not self.is_writing then return "" end
  local buffer = {}
  local num = startnum
  for _,item in ipairs(items) do
    if item ~= "" then
      buffer[#buffer + 1] = self.ordereditem(item,num)
    end
    if num ~= nil and item ~= "" then
      num = num + 1
    end
  end
  local contents = util.intersperse(buffer,"\n")
  if tight and options.tightLists then
    return {"\\markdownRendererOlBeginTight\n",contents,
            "\n\\markdownRendererOlEndTight "}
  else
```

```
8309        return {"\\markdownRendererOlBegin\n",contents,
8310                "\n\\markdownRendererOlEnd "}
8311      end
8312    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.

```
8313    function self.ordereditem(s,num)
8314      if num ~= nil then
8315        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
8316                "\\markdownRendererOlItemEnd "}
8317      else
8318        return {"\\markdownRendererOlItem ",s,
8319                "\\markdownRendererOlItemEnd "}
8320      end
8321    end
```

#### 3.1.4.9 Formatters of html Tags, Elements, and Comments

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.

```
8322    function self.inline_html_comment(contents)
8323      if self.flatten_inlines then return contents end
8324      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
8325    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.

```
8326    function self.inline_html_tag(contents)
8327      if self.flatten_inlines then return contents end
8328      return {"\\markdownRendererInlineHtmlTag{",
8329              self.string(contents),"}"}
8330    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.

```
8331    function self.block_html_element(s)
8332      if not self.is_writing then return "" end
8333      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8334      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
8335    end
```

Define `writer->block_html_cdata_element` as a function that transforms the contents of a block HTML CDATA element such as `<script>`, `<pre>`, `<style>`, or `<textarea>` (CommonMark HTML block type 1) to the output format.

```
8336    function self.block_html_cdata_element(s)
```

```
8337      if not self.is_writing then return "" end
8338      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8339      return {"\\markdownRendererInputBlockHtmlCdataElement{",name,"}"}
8340    end
```

Define `writer->block_html_comment` as a function that transforms the contents of a block HTML comment (CommonMark HTML block type 2) to the output format, where `s` is the full comment text.

```
8341    function self.block_html_comment(contents)
8342      if self.flatten_inlines then return contents end
8343      return {"\\markdownRendererBlockHtmlComment{",contents,"}"}
8344    end
```

Define `writer->block_html_processing_instruction` as a function that transforms the contents of a block HTML processing instruction (CommonMark HTML block type 3) to the output format.

```
8345    function self.block_html_processing_instruction(s)
8346      if not self.is_writing then return "" end
8347      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8348      return {
8349        "\\markdownRendererInputBlockHtmlProcessingInstruction{",name,"}"}
8350    end
```

Define `writer->block_html_declaration` as a function that transforms the contents of a block HTML declaration (CommonMark HTML block type 4) to the output format.

```
8351    function self.block_html_declaration(s)
8352      if not self.is_writing then return "" end
8353      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8354      return {
8355        "\\markdownRendererInputBlockHtmlDeclaration{",name,"}"}
8356    end
```

Define `writer->block_html_cdata_section` as a function that transforms the contents of a block HTML CDATA section (CommonMark HTML block type 5) to the output format.

```
8357    function self.block_html_cdata_section(s)
8358      if not self.is_writing then return "" end
8359      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8360      return {"\\markdownRendererInputBlockHtmlCdataSection{",name,"}"}
8361    end
```

Define `writer->block_html_pcdata_element` as a function that transforms the contents of a block HTML PCDATA element such as `<div>` or `<table>` (CommonMark HTML block type 6) to the output format.

```
8362    function self.block_html_pcdata_element(s)
8363      if not self.is_writing then return "" end
8364      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8365      return {"\\markdownRendererInputBlockHtmlPcdataElement{",name,"}"}
8366    end
```

Define writer->block_html_standalone_tag as a function that transforms the contents of a block HTML element that does not fall into types 1–6 (CommonMark HTML block type 7) to the output format.

```
function self.block_html_standalone_tag(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererBlockHtmlStandaloneTag{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_processing_instruction as a function that transforms the contents of an inline HTML processing instruction to the output format.

```
function self.inline_html_processing_instruction(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlProcessingInstruction{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_declaration as a function that transforms the contents of an inline HTML declaration to the output format.

```
function self.inline_html_declaration(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlDeclaration{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_cdata_section as a function that transforms the contents of an inline HTML CDATA section to the output format.

```
function self.inline_html_cdata_section(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlCdataSection{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_open_tag as a function that transforms the contents of an inline HTML opening tag to the output format.

```
function self.inline_html_open_tag(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlOpenTag{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_close_tag as a function that transforms the contents of an inline HTML closing tag to the output format.

```
function self.inline_html_close_tag(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlCloseTag{",
          self.string(contents),"}"}
end
```

Define writer->inline_html_empty_tag as a function that transforms the contents of an inline HTML self-closing (empty) tag to the output format.

```
function self.inline_html_empty_tag(contents)
  if self.flatten_inlines then return contents end
  return {"\\markdownRendererInlineHtmlEmptyTag{",
          self.string(contents),"}"}
end
```

#### 3.1.4.10 Formatter of Emphasis

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

```
function self.emphasis(s)
  if self.flatten_inlines then return s end
  return {"\\markdownRendererEmphasis{",s,"}"}
end
```

#### 3.1.4.11 Formatter of Strong Emphasis

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

```
function self.strong(s)
  if self.flatten_inlines then return s end
  return {"\\markdownRendererStrongEmphasis{",s,"}"}
end
```

#### 3.1.4.12 Formatter of Tickboxes

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

```
function self.tickbox(f)
  if f == 1.0 then
    return "☒ "
  elseif f == 0.0 then
    return "☐ "
  else
    return "⊡ "
  end
end
```

#### 3.1.4.13 Formatter of Blockquotes

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

```
function self.blockquote(s)
  if not self.is_writing then return "" end
  return {"\\markdownRendererBlockQuoteBegin\n",s,
    "\\markdownRendererBlockQuoteEnd "}
end
```

#### 3.1.4.14 Formatter of Code Blocks

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

```
8424  function self.verbatim(s)
8425    if not self.is_writing then return "" end
8426    s = s:gsub("\n$", "")
8427    local name = util.cache_verbatim(options.cacheDir, s)
8428    return {"\\markdownRendererInputVerbatim{",name,"}"}
8429  end
```

#### 3.1.4.15 Formatter of Documents

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

```
8430  function self.document(d)
8431    local buf = {"\\markdownRendererDocumentBegin\n"}
8432
8433    -- warn against the `hybrid` option
8434    if options.hybrid then
8435      local text = "The `hybrid` option has been soft-deprecated."
8436      local more = "Consider using one of the following better options "
8437                .. "for mixing TeX and markdown: `contentBlocks`, "
8438                .. "`rawAttribute`, `texComments`, `texMathDollars`, "
8439                .. "`texMathSingleBackslash`, and "
8440                .. "`texMathDoubleBackslash`. "
8441                .. "For more information, see the user manual at "
8442                .. "<https://witiko.github.io/markdown/>."
8443      table.insert(buf, self.warning(text, more))
8444    end
8445
8446    -- insert the text of the document
8447    table.insert(buf, d)
8448
8449    -- pop all attributes
8450    table.insert(buf, self.pop_attributes())
8451
8452    table.insert(buf, "\\markdownRendererDocumentEnd")
8453
8454    return buf
8455  end
```

#### 3.1.4.16 Formatter of Attributes

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```
8456  local seen_identifiers = {}
8457  local key_value_regex = "([^= ]+)%s*=%s*(.*)"
8458  local function normalize_attributes(attributes, auto_identifiers)
8459    -- normalize attributes
```

```
8460    local normalized_attributes = {}
8461    local has_explicit_identifiers = false
8462    local key, value
8463    for _, attribute in ipairs(attributes or {}) do
8464      if attribute:sub(1, 1) == "#" then
8465        table.insert(normalized_attributes, attribute)
8466        has_explicit_identifiers = true
8467        seen_identifiers[attribute:sub(2)] = true
8468      elseif attribute:sub(1, 1) == "." then
8469        table.insert(normalized_attributes, attribute)
8470      else
8471        key, value = attribute:match(key_value_regex)
8472        if key:lower() == "id" then
8473          table.insert(normalized_attributes, "#" .. value)
8474        elseif key:lower() == "class" then
8475          local classes = {}
8476          for class in value:gmatch("%S+") do
8477            table.insert(classes, class)
8478          end
8479          table.sort(classes)
8480          for _, class in ipairs(classes) do
8481            table.insert(normalized_attributes, "." .. class)
8482          end
8483        else
8484          table.insert(normalized_attributes, attribute)
8485        end
8486      end
8487    end
8488
8489    -- if no explicit identifiers exist, add auto identifiers
8490    if not has_explicit_identifiers and auto_identifiers ~= nil then
8491      local seen_auto_identifiers = {}
8492      for _, auto_identifier in ipairs(auto_identifiers) do
8493        if seen_auto_identifiers[auto_identifier] == nil then
8494          seen_auto_identifiers[auto_identifier] = true
8495          if seen_identifiers[auto_identifier] == nil then
8496            seen_identifiers[auto_identifier] = true
8497            table.insert(normalized_attributes,
8498                         "#" .. auto_identifier)
8499          else
8500            local auto_identifier_number = 1
8501            while true do
8502              local numbered_auto_identifier = auto_identifier .. "-"
8503                                               .. auto_identifier_number
8504              if seen_identifiers[numbered_auto_identifier] == nil then
8505                seen_identifiers[numbered_auto_identifier] = true
8506                table.insert(normalized_attributes,
8507                             "#" .. numbered_auto_identifier)
8508                break
```

```
            end
            auto_identifier_number = auto_identifier_number + 1
          end
        end
      end
    end

    -- sort and deduplicate normalized attributes
    table.sort(normalized_attributes)
    local seen_normalized_attributes = {}
    local deduplicated_normalized_attributes = {}
    for _, attribute in ipairs(normalized_attributes) do
      if seen_normalized_attributes[attribute] == nil then
        seen_normalized_attributes[attribute] = true
        table.insert(deduplicated_normalized_attributes, attribute)
      end
    end

    return deduplicated_normalized_attributes
  end

  function self.attributes(attributes, should_normalize_attributes)
    local normalized_attributes
    if should_normalize_attributes == false then
      normalized_attributes = attributes
    else
      normalized_attributes = normalize_attributes(attributes)
    end

    local buf = {}
    local key, value
    for _, attribute in ipairs(normalized_attributes) do
      if attribute:sub(1, 1) == "#" then
        table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
                           attribute:sub(2), "}"})
      elseif attribute:sub(1, 1) == "." then
        table.insert(buf, {"\\markdownRendererAttributeClassName{",
                           attribute:sub(2), "}"})
      else
        key, value = attribute:match(key_value_regex)
        table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
                           key, "}{", value, "}"})
      end
    end

    return buf
  end
```

#### 3.1.4.17 Tracking Active Attributes

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

```
8557  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`.

```
8558  self.attribute_type_levels = {}
8559  setmetatable(self.attribute_type_levels,
8560               { __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`.

```
8561  local function apply_attributes()
8562    local buf = {}
8563    for i = 1, #self.active_attributes do
8564      local start_output = self.active_attributes[i][3]
8565      if start_output ~= nil then
8566        table.insert(buf, start_output)
8567      end
8568    end
8569    return buf
8570  end
8571
8572  local function tear_down_attributes()
8573    local buf = {}
8574    for i = #self.active_attributes, 1, -1 do
8575      local end_output = self.active_attributes[i][4]
8576      if end_output ~= nil then
8577        table.insert(buf, end_output)
8578      end
8579    end
8580    return buf
8581  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.

```
8582  function self.push_attributes(attribute_type, attributes,
8583                                start_output, end_output)
8584    local attribute_type_level
8585      = self.attribute_type_levels[attribute_type]
8586    self.attribute_type_levels[attribute_type]
8587      = attribute_type_level + 1
8588
8589    -- index attributes in a hash table for easy lookup
```

```
8590    attributes = attributes or {}
8591    for i = 1, #attributes do
8592      attributes[attributes[i]] = true
8593    end
8594
8595    local buf = {}
8596    -- handle slicing
8597    if attributes["#" .. self.slice_end_identifier] ~= nil and
8598       self.slice_end_type == "^" then
8599      if self.is_writing then
8600        table.insert(buf, self.undosep())
8601        table.insert(buf, tear_down_attributes())
8602      end
8603      self.is_writing = false
8604    end
8605    if attributes["#" .. self.slice_begin_identifier] ~= nil and
8606       self.slice_begin_type == "^" then
8607      table.insert(buf, apply_attributes())
8608      self.is_writing = true
8609    end
8610    if self.is_writing and start_output ~= nil then
8611      table.insert(buf, start_output)
8612    end
8613    table.insert(self.active_attributes,
8614                 {attribute_type, attributes,
8615                  start_output, end_output})
8616    return buf
8617  end
8618
```

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`).

```
8619  function self.pop_attributes(attribute_type)
8620    local buf = {}
8621    -- pop attributes until we find attributes of correct type
8622    -- or until no attributes remain
8623    local current_attribute_type = false
8624    while current_attribute_type ~= attribute_type and
8625          #self.active_attributes > 0 do
8626      local attributes, _, end_output
8627      current_attribute_type, attributes, _, end_output = table.unpack(
8628        self.active_attributes[#self.active_attributes])
8629      local attribute_type_level
8630        = self.attribute_type_levels[current_attribute_type]
8631      self.attribute_type_levels[current_attribute_type]
8632        = attribute_type_level - 1
```

```
      if self.is_writing and end_output ~= nil then
        table.insert(buf, end_output)
      end
      table.remove(self.active_attributes, #self.active_attributes)
      -- handle slicing
      if attributes["#" .. self.slice_end_identifier] ~= nil
         and self.slice_end_type == "$" then
        if self.is_writing then
          table.insert(buf, self.undosep())
          table.insert(buf, tear_down_attributes())
        end
        self.is_writing = false
      end
      if attributes["#" .. self.slice_begin_identifier] ~= nil and
         self.slice_begin_type == "$" then
        self.is_writing = true
        table.insert(buf, apply_attributes())
      end
    end
    return buf
  end
```

#### 3.1.4.18 Automatically Generated Identifiers for Headings

Create an auto identifier string by stripping and converting characters from string `s`.

```
  local function create_auto_identifier(s)
    local buffer = {}
    local prev_space = false
    local letter_found = false
    local normalized_s = s
    if not options.unicodeNormalization
       or options.unicodeNormalizationForm ~= "nfc" then
      normalized_s = util.normalize(normalized_s, "nfc")
    end

    for _, code in utf8.codes(normalized_s) do
      local char = utf8.char(code)

      -- Remove everything up to the first letter.
      if not letter_found then
        local is_letter = lpeg.match(
          parsers.unicode.alpha,
          char
        )
        if is_letter then
          letter_found = true
        else
          goto continue
        end
      end
```

```
8679
8680      -- Remove all non-alphanumeric characters, except underscores,
8681      -- hyphens, and periods.
8682      if not lpeg.match(
8683        ( parsers.underscore
8684        + parsers.dash
8685        + parsers.period
8686        + parsers.unicode.word
8687        + #parsers.unicode.whitespace ),
8688        char
8689      ) then
8690        goto continue
8691      end
8692
8693      -- Replace all spaces and newlines with hyphens.
8694      if lpeg.match(
8695        ( parsers.newline
8696        + #parsers.unicode.whitespace ),
8697        char
8698      ) then
8699        char = "-"
8700        if prev_space then
8701          goto continue
8702        else
8703          prev_space = true
8704        end
8705      else
8706        -- Case-fold all alphabetic characters.
8707        local form = nil
8708        if options.unicodeNormalization then
8709          form = options.unicodeNormalizationForm
8710        end
8711        char = util.casefold(char, form)
8712        prev_space = false
8713      end
8714
8715      table.insert(buffer, char)
8716
8717      ::continue::
8718    end
8719
8720    if prev_space then
8721      table.remove(buffer)
8722    end
8723
8724    local identifier = #buffer == 0 and "section"
8725                    or table.concat(buffer, "")
8726    return identifier
8727  end
```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```
local function create_gfm_auto_identifier(s)
  local buffer = {}
  local prev_space = false
  local letter_found = false
  local normalized_s = s
  if not options.unicodeNormalization
     or options.unicodeNormalizationForm ~= "nfc" then
    normalized_s = util.normalize(normalized_s, "nfc")
  end

  for _, code in utf8.codes(normalized_s) do
    local char = utf8.char(code)

    -- Remove everything up to the first non-space.
    if not letter_found then
      local is_letter = not lpeg.match(
        #parsers.unicode.whitespace,
        char
      )
      if is_letter then
        letter_found = true
      else
        goto continue
      end
    end

    -- Remove all non-alphanumeric characters, except underscores
    -- and hyphens.
    if not lpeg.match(
      ( parsers.underscore
      + parsers.dash
      + parsers.unicode.word
      + #parsers.unicode.whitespace ),
      char
    ) then
      prev_space = false
      goto continue
    end

    -- Replace all spaces and newlines with hyphens.
    if lpeg.match(
      ( parsers.newline
      + #parsers.unicode.whitespace ),
      char
    ) then
      char = "-"
      if prev_space then
```

```
          goto continue
        else
          prev_space = true
        end
      else
        -- Case-fold all alphabetic characters.
        local form = nil
        if options.unicodeNormalization then
          form = options.unicodeNormalizationForm
        end
        char = util.casefold(char, form)
        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
```

#### 3.1.4.19 Formatter of Headings

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)

    -- push empty attributes for implied sections
    while self.attribute_type_levels["heading"] < level - 1 do
      table.insert(buf,
                   self.push_attributes("heading",
                                        nil,
                                        self.secbegin_text,
                                        self.secend_text))
    end

    -- pop attributes for sections that have ended
    while self.attribute_type_levels["heading"] >= level do
```

```
8819      table.insert(buf, self.pop_attributes("heading"))
8820    end
8821
8822    -- construct attributes for the new section
8823    local auto_identifiers = {}
8824    if self.options.autoIdentifiers then
8825      table.insert(auto_identifiers, create_auto_identifier(flat_text))
8826    end
8827    if self.options.gfmAutoIdentifiers then
8828      table.insert(auto_identifiers,
8829                   create_gfm_auto_identifier(flat_text))
8830    end
8831    local normalized_attributes = normalize_attributes(attributes,
8832                                                       auto_identifiers)
8833
8834    -- push attributes for the new section
8835    local start_output = {}
8836    local end_output = {}
8837    table.insert(start_output, self.secbegin_text)
8838    table.insert(end_output, self.secend_text)
8839
8840    table.insert(buf, self.push_attributes("heading",
8841                                           normalized_attributes,
8842                                           start_output,
8843                                           end_output))
8844    assert(self.attribute_type_levels["heading"] == level)
8845
8846    -- render the heading and its attributes
8847    if self.is_writing and #normalized_attributes > 0 then
8848      table.insert(buf,
8849                   "\\markdownRendererHeaderAttributeContextBegin\n")
8850      table.insert(buf, self.attributes(normalized_attributes, false))
8851    end
8852
8853    local cmd
8854    level = level + options.shiftHeadings
8855    if level <= 1 then
8856      cmd = "\\markdownRendererHeadingOne"
8857    elseif level == 2 then
8858      cmd = "\\markdownRendererHeadingTwo"
8859    elseif level == 3 then
8860      cmd = "\\markdownRendererHeadingThree"
8861    elseif level == 4 then
8862      cmd = "\\markdownRendererHeadingFour"
8863    elseif level == 5 then
8864      cmd = "\\markdownRendererHeadingFive"
8865    elseif level >= 6 then
8866      cmd = "\\markdownRendererHeadingSix"
8867    else
```

```
8868        cmd = ""
8869      end
8870      if self.is_writing then
8871        table.insert(buf, {cmd, "{", inlines, "}"})
8872      end
8873
8874      if self.is_writing and #normalized_attributes > 0 then
8875        table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
8876      end
8877
8878      return buf
8879    end
```

#### 3.1.4.20 Managing State and Deferred Writer Calls

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.

```
8880    function self.get_state()
8881      return {
8882        is_writing=self.is_writing,
8883        flatten_inlines=self.flatten_inlines,
8884        active_attributes={table.unpack(self.active_attributes)},
8885      }
8886    end
```

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

```
8887    function self.set_state(s)
8888      local previous_state = self.get_state()
8889      for key, value in pairs(s) do
8890        self[key] = value
8891      end
8892      return previous_state
8893    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`.

```
8894    function self.defer_call(f)
8895      local previous_state = self.get_state()
8896      return function(...)
8897        local state = self.set_state(previous_state)
8898        local return_value = f(...)
8899        self.set_state(state)
8900        return return_value
8901      end
8902    end
8903
8904    return self
8905  end
```

### 3.1.5 Parsers

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

```
8906 parsers                        = {}
```

#### 3.1.5.1 Basic Parsers

```
8907 parsers.percent                = P("%")
8908 parsers.at                     = P("@")
8909 parsers.comma                  = P(",")
8910 parsers.asterisk               = P("*")
8911 parsers.dash                   = P("-")
8912 parsers.plus                   = P("+")
8913 parsers.underscore             = P("_")
8914 parsers.period                 = P(".")
8915 parsers.hash                   = P("#")
8916 parsers.dollar                 = P("$")
8917 parsers.ampersand              = P("&")
8918 parsers.backtick               = P("`")
8919 parsers.less                   = P("<")
8920 parsers.more                   = P(">")
8921 parsers.space                  = P(" ")
8922 parsers.squote                 = P("'")
8923 parsers.dquote                 = P('"')
8924 parsers.lparent                = P("(")
8925 parsers.rparent                = P(")")
8926 parsers.lbracket               = P("[")
8927 parsers.rbracket               = P("]")
8928 parsers.lbrace                 = P("{")
8929 parsers.rbrace                 = P("}")
8930 parsers.circumflex             = P("^")
8931 parsers.slash                  = P("/")
8932 parsers.equal                  = P("=")
8933 parsers.colon                  = P(":")
8934 parsers.semicolon              = P(";")
8935 parsers.exclamation            = P("!")
8936 parsers.pipe                   = P("|")
8937 parsers.tilde                  = P("~")
8938 parsers.backslash              = P("\\")
8939 parsers.tab                    = P("\t")
8940 parsers.newline                = P("\n")
8941
8942 parsers.digit                  = R("09")
8943 parsers.hexdigit               = R("09","af","AF")
8944 parsers.letter                 = R("AZ","az")
8945 parsers.alphanumeric           = R("AZ","az","09")
8946 parsers.keyword                = parsers.letter
8947                                * (parsers.alphanumeric + parsers.dash)^0
8948
```

```
parsers.doubleasterisks        = P("**")
parsers.doubleunderscores      = P("__")
parsers.doubletildes           = P("~~")
parsers.fourspaces             = P("    ")

parsers.any                    = P(1)
parsers.eof                    = P(-1)
parsers.succeed                = P(true)
parsers.fail                   = P(false)

parsers.internal_punctuation   = S(":;,.?")
parsers.ascii_punctuation      = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

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.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

```

### 3.1.5.2 Parsers for Unicode Character Classes and Categories

We define high-level parsers in table `parsers.unicode` based on the low-level parsers in `unicode_data.categories`, defined in Section 3.1.1.5. Unlike the low-level parsers, the high-level parsers are invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
parsers.unicode = {}
parsers.unicode.preceding_punctuation = parsers.fail
parsers.unicode.punctuation = parsers.fail
parsers.unicode.alpha = parsers.fail
parsers.unicode.numeric = parsers.fail
parsers.unicode.word = parsers.fail
parsers.unicode.preceding_whitespace = parsers.fail
parsers.unicode.whitespace = parsers.fail
for n = 1, 4 do
```

For punctuation, accept any characters from Unicode categories P (punctuation) and S (symbol), as mandated by the CommonMark standard[40].

```
  local punctuation_of_length_n
    = unicode_data.categories.P[n]
    + unicode_data.categories.S[n]
   parsers.unicode.preceding_punctuation
    = parsers.unicode.preceding_punctuation
    + B(punctuation_of_length_n)
   parsers.unicode.punctuation
    = parsers.unicode.punctuation
    + punctuation_of_length_n
```

For alphabetical characters, accept any characters from Unicode category L (letter), similar to the character class ‘Unicode.

```
  local alpha_of_length_n = unicode_data.categories.L[n]
  parsers.unicode.alpha
    = parsers.unicode.alpha
    + alpha_of_length_n
```

For numeric characters, accept any characters from Unicode category N (number), similar to the character class ‘Unicode.

```
  local numeric_of_length_n = unicode_data.categories.N[n]
  parsers.unicode.numeric
    = parsers.unicode.numeric
    + numeric_of_length_n
```

For word characters, accept any characters from Unicode categories L (letter), N (number), and Pc (connector punctuation), similar to the character class ‘

```
  local word_of_length_n
    = alpha_of_length_n
    + numeric_of_length_n
    + unicode_data.categories.Pc[n]
  parsers.unicode.word
```

[40] See https://spec.commonmark.org/0.31.2/#unicode-punctuation-character.

```
    = parsers.unicode.word
    + word_of_length_n
```

For space characters, accept any characters from Unicode category Z (separator), as well as the ASCII control characters 9 (horizontal tab) through 13 (carriage return), similar to the character class 'Lua library Selene Unicode.

```
  local whitespace_of_length_n = unicode_data.categories.Z[n]
  if n == 1 then
    whitespace_of_length_n
      = whitespace_of_length_n
      + R("\t\r")
  end
  parsers.unicode.preceding_whitespace
    = parsers.unicode.preceding_whitespace
    + B(whitespace_of_length_n)
  parsers.unicode.whitespace
    = parsers.unicode.whitespace
    + whitespace_of_length_n
end
```

#### 3.1.5.3 Parsers Used for Indentation

```

parsers.leader      = parsers.space^-3

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```
local function has_trail(indent_table)
  return indent_table ~= nil and
    indent_table.trail ~= nil and
    next(indent_table.trail) ~= nil
end

```

Check if indent table `indent_table` has any indents.

```
local function has_indents(indent_table)
  return indent_table ~= nil and
    indent_table.indents ~= nil and
    next(indent_table.indents) ~= nil
end

```

Add a trail `trail_info` to the indent table `indent_table`.

```
local function add_trail(indent_table, trail_info)
  indent_table.trail = trail_info
  return indent_table
end

```

Remove a trail `trail_info` from the indent table `indent_table`.

```
local function remove_trail(indent_table)
```

```
  indent_table.trail = nil
  return indent_table
end

```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
local function update_indent_table(indent_table, new_indent, add)
  indent_table = remove_trail(indent_table)

  if not has_indents(indent_table) then
    indent_table.indents = {}
  end


  if add then
    indent_table.indents[#indent_table.indents + 1] = new_indent
  else
    if indent_table.indents[#indent_table.indents].name
       == 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 = {}
```

```
9286
9287  local blank_starter = left_blank_starter(indent_table)
9288
9289  if current_line_indents == nil then
9290    current_line_indents = {}
9291  end
9292
9293  for index = 1,#indent_table.indents do
9294    local value = indent_table.indents[index]
9295    local pattern = decode_pattern(value.name)
9296
9297    -- match decoded pattern
9298    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
9299    if new_indent_info == nil then
9300      local blankline_end = lpeg.match(
9301        Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
9302      if is_optional or not indent_table.ignore_blockquote_blank
9303         or not blankline_end then
9304        return is_optional, new_index, current_trail,
9305               current_line_indents
9306      end
9307
9308      return traverse_indent(s, tonumber(blankline_end.pos),
9309                             indent_table, is_optional, is_blank,
9310                             current_line_indents)
9311    end
9312
9313    local raw_last_trail = new_indent_info[1]
9314    local delimiter = new_indent_info[2]
9315    local raw_new_trail = new_indent_info[3]
9316    local next_index = new_indent_info[4]
9317
9318    local space_only = delimiter == ""
9319
9320    -- check previous trail
9321    if not space_only and next(current_trail) == nil then
9322      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
9323      current_trail = {is_code=sp.is_code, remainder=sp.remainder,
9324                       total_length=sp.total_length,
9325                       full_remainder=sp.full_remainder}
9326    end
9327
9328    if next(current_trail) ~= nil then
9329      if not space_only and current_trail.is_code then
9330        return is_optional, new_index, current_trail,
9331               current_line_indents
9332      end
9333      if current_trail.internal_remainder ~= nil then
9334        raw_last_trail = current_trail.internal_remainder
```

```
      end
    end

    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}
```

```
9384
9385      current_line_indents[#current_line_indents + 1] = new_indent_info
9386      new_index = next_index
9387    end
9388
9389    return true, new_index, current_trail, current_line_indents
9390  end
9391
```

Check if a code trail is expected.

```
9392  local function check_trail(expect_code, is_code)
9393    return (expect_code and is_code) or (not expect_code and not is_code)
9394  end
9395
```

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`.

```
9396  local check_trail_joined =
9397    function(s, i, indent_table, -- luacheck: ignore s i
9398             spacing, expect_code, omit_remainder)
9399      local is_code
9400      local remainder
9401
9402      if has_trail(indent_table) then
9403        local trail = indent_table.trail
9404        is_code = trail.is_code
9405        if is_code then
9406          remainder = trail.remainder
9407        else
9408          remainder = trail.full_remainder
9409        end
9410      else
9411        local sp = process_starter_spacing(0, spacing, 0, 0)
9412        is_code = sp.is_code
9413        if is_code then
9414          remainder = sp.remainder
9415        else
9416          remainder = sp.full_remainder
9417        end
9418      end
9419
9420      local result = check_trail(expect_code, is_code)
9421      if omit_remainder then
9422        return result
9423      end
9424      return result, remainder
9425    end
9426
```

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)
```

```
9602
9603 parsers.check_minimal_blank_indent
9604   = Cmt( Cb("indent_info") * Cc(false)
9605        * Cc(true)
9606        , check_continuation_indentation)
9607
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
9608
9609 parsers.check_minimal_indent_and_trail =
9610   Cmt( Cb("indent_info")
9611      * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
9612      , check_continuation_indentation_and_trail)
9613
9614 parsers.check_minimal_indent_and_code_trail =
9615   Cmt( Cb("indent_info")
9616      * Cc(false) * Cc(false) * Cc("code") * Cc(false)
9617      , check_continuation_indentation_and_trail)
9618
9619 parsers.check_minimal_blank_indent_and_full_code_trail =
9620   Cmt( Cb("indent_info")
9621      * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
9622      , check_continuation_indentation_and_trail)
9623
9624 parsers.check_minimal_indent_and_any_trail =
9625   Cmt( Cb("indent_info")
9626      * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9627      , check_continuation_indentation_and_trail)
9628
9629 parsers.check_minimal_blank_indent_and_any_trail =
9630   Cmt( Cb("indent_info")
9631      * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9632      , check_continuation_indentation_and_trail)
9633
9634 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
9635   Cmt( Cb("indent_info")
9636      * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
9637      , check_continuation_indentation_and_trail)
9638
9639 parsers.check_optional_indent_and_any_trail =
9640   Cmt( Cb("indent_info")
9641      * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9642      , check_continuation_indentation_and_trail)
9643
9644 parsers.check_optional_blank_indent_and_any_trail =
9645   Cmt( Cb("indent_info")
9646      * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9647      , check_continuation_indentation_and_trail)
9648
```

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")
                       * Cg(Cc(true), "pure_comment")
                       * ((#(parsers.commented_line_letter
                            - parsers.newline)
                          * Cb("backslashes")
                          * Cs(parsers.commented_line_letter
                            - parsers.newline)^1  -- initial
                          * Cg(Cc(""), "backslashes")
                          * Cg(Cc(false), "pure_comment"))
                         + #( parsers.backslash
                            * (parsers.backslash + parsers.newline))
                         * Cg((parsers.backslash  -- even backslash
                              * ( parsers.backslash
                                + #parsers.newline))^1, "backslashes")
                         * Cg(Cc(false), "pure_comment")
                         + (parsers.backslash
                           * (#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")
```

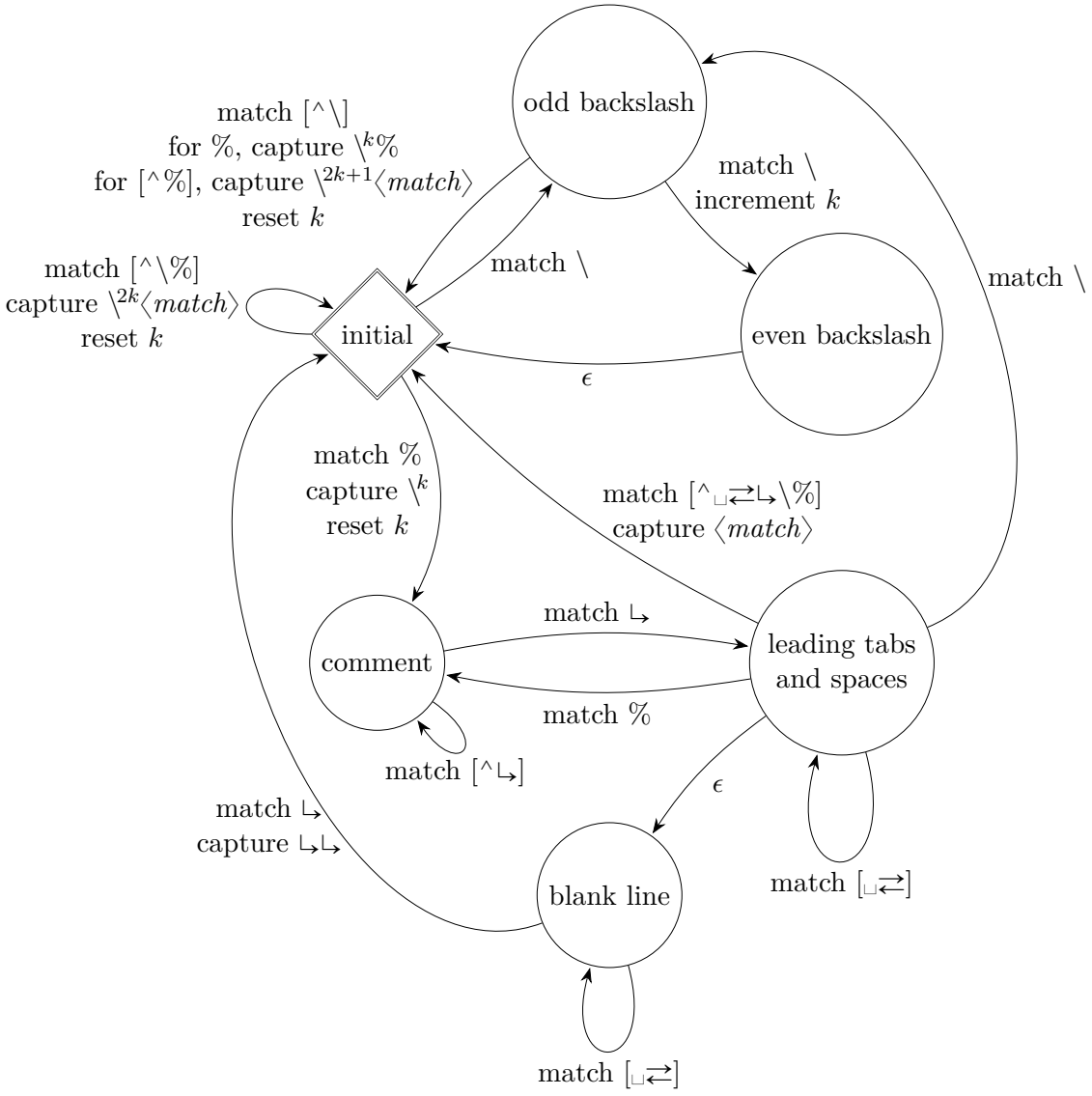


**Figure 8: A pushdown automaton that recognizes TEX comments**

```
                            * Cg(Cc(false), "pure_comment")))^0
                        * (#parsers.percent
                          * Cb("backslashes")
                          / function(backslashes)
                            return string.rep("\\", #backslashes / 2)
                          end
                          * ((Cb("pure_comment")
                             * parsers.percent  -- comment
                             * parsers.line
                             * #parsers.blankline) -- blank line
                            / function(is_pure)
                              return is_pure and "" or "\n"
                            end
                            + 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.unicode.alpha
                           + parsers.unicode.numeric
                           + S("-_:.")
parsers.attribute_raw_char = parsers.unicode.alpha
                           + parsers.unicode.numeric
                           + 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.unicode.alpha
                            * 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
```

#### 3.1.5.4 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
parsers.html_space_entity_char
  = Cmt(parsers.html_entities, function(_, i, entity)
     return entity == " " and i or nil
    end)
```

#### 3.1.5.5 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)

```

#### 3.1.5.6 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

```

#### 3.1.5.7 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(C( parsers.newline
        * (parsers.check_minimal_blank_indent_and_any_trail
          * #parsers.blankline
          + parsers.check_minimal_indent_and_any_trail)
      + parsers.linechar - end_marker)^0)
    * (C(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

parsers.html_inline_attribute_value_specification
  = parsers.html_spnl
  * parsers.equal
  * parsers.html_spnl
  * parsers.html_inline_attribute_value

parsers.html_attribute
  = parsers.html_attribute_spacing
  * parsers.html_attribute_name
  * parsers.html_inline_attribute_value_specification^-1

parsers.html_non_newline_attribute
```

```
  = parsers.spacechar^1
  * parsers.html_attribute_name
  * parsers.html_attribute_value_specification^-1

parsers.nested_breaking_blank = parsers.newline
                              * parsers.check_minimal_blank_indent
                              * parsers.blankline

parsers.html_comment_start = P("<!--")
                           * parsers.optionalspace

parsers.html_comment_end = parsers.optionalspace
                         * P("-->")

parsers.html_comment
  = parsers.html_comment_start
  * parsers.html_until_end(parsers.html_comment_end)

parsers.html_inline_comment = (parsers.html_comment_start / "")
                            * -P(">") * -P("->")
                            * Cs(( parsers.any
                                 - parsers.nested_breaking_blank
                                 - parsers.html_comment_end)^0)
                            * (parsers.html_comment_end / "")

parsers.html_cdatasection_start = P("<![CDATA[")

parsers.html_cdatasection_end = P("]]>")

parsers.html_cdatasection
  = Cs( parsers.html_cdatasection_start
      * parsers.html_until_end(parsers.html_cdatasection_end))
  / remove_trailing_blank_lines

parsers.html_inline_cdatasection
  = 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))
  / remove_trailing_blank_lines

```

```
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))
  / remove_trailing_blank_lines

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
  = Cs( parsers.html_special_block_opening
      * parsers.html_until_end(parsers.html_close_special_tag))
```

```
  / remove_trailing_blank_lines

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(( 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

```

#### 3.1.5.8 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("Space")
                          + 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("Space")
                                + 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 }

-- url for markdown links, allowing nested brackets:
parsers.url         = parsers.less * Cs((parsers.anyescaped
                                         - parsers.newline
                                         - parsers.less
                                         - parsers.more)^0)
                                   * parsers.more
                    + -parsers.less
                    * Cs((parsers.inparens_url + (parsers.anyescaped
                                                  - parsers.spacing
                                                  - parsers.lparent
                                                  - parsers.rparent))^1)

-- quoted text:
parsers.title_s     = parsers.squote
                    * Cs((parsers.html_entities
                         + V("Space")
                         + V("NoSoftLineBreakEndline")
                         + ( parsers.anyescaped
                           - parsers.newline
                           - parsers.squote
                           - parsers.blankline^2))^0)
                    * parsers.squote

parsers.title_d     = parsers.dquote
                    * Cs((parsers.html_entities
```

```
                          + V("Space")
                          + V("NoSoftLineBreakEndline")
                          + ( parsers.anyescaped
                            - parsers.newline
                            - parsers.dquote
                            - parsers.blankline^2))^0)
                    * parsers.dquote

parsers.title_p     = parsers.lparent
                    * Cs((parsers.html_entities
                          + V("Space")
                          + V("NoSoftLineBreakEndline")
                          + ( parsers.anyescaped
                            - parsers.newline
                            - parsers.lparent
                            - parsers.rparent
                            - parsers.blankline^2))^0)
                    * parsers.rparent

parsers.title
  = parsers.title_d + parsers.title_s + parsers.title_p

parsers.optionaltitle
  = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")

```

#### 3.1.5.9 Helpers for Links and Link Reference Definitions

```
-- parse a reference definition:  [foo]: /bar "title"
parsers.define_reference_parser = (parsers.check_trail / "")
                                * parsers.link_label * parsers.colon
                                * parsers.spnlc * parsers.url
                                * ( parsers.spnlc_sep * parsers.title
                                  * parsers.only_blank
                                  + Cc("") * parsers.only_blank)
```

#### 3.1.5.10 Inline Elements

```
parsers.Inline         = V("Inline")

-- parse many p between starter and ender
parsers.between = function(p, starter, ender)
  local ender2 = B(parsers.nonspacechar) * ender
  return ( starter
         * #parsers.nonspacechar
         * Ct(p * (p - ender2)^0)
         * ender2)
end

```

#### 3.1.5.11 Block Elements

```
parsers.lineof = function(c)
    return ( parsers.check_trail_no_rem
           * (P(c) * parsers.optionalspace)^3
           * (parsers.newline + parsers.eof))
end

parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
                             + parsers.lineof(parsers.dash)
                             + parsers.lineof(parsers.underscore)
```

#### 3.1.5.12 Headings

```
-- parse Atx heading start and return level
parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
                      * -parsers.hash / length

-- parse setext header ending and return level
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]
      end
    })
  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.

```
  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.

```
  function self.normalize_tag(tag)
    tag = util.rope_to_string(tag)
    tag = tag:gsub("[ \n\r\t]+", " ")
    tag = tag:gsub("^ ", ""):gsub(" $", "")
    local form = nil
    if options.unicodeNormalization then
      form = options.unicodeNormalizationForm
    end
    tag = util.casefold(tag, form)
    return tag
  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.

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

```
  if options.preserveTabs then
```

```
    self.expandtabs = function(s) return s end
  else
    self.expandtabs = function(s)
                        if s:find("\t") then
                          return iterlines(s, util.expand_tabs_in_line)
                        else
                          return s
                        end
                      end
  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.

```
10556       if toplevel and (options.texComments or options.hybrid) then
10557         str = lpeg.match(Ct(parsers.commented_line^1), str)
10558         str = util.rope_to_string(str)
10559       end
10560       local res = lpeg.match(grammar(), str)
10561       if res == nil then
10562         return writer.error(
10563           format("Parser `%s` failed to process the input text.", name),
10564           format("Here are the first 20 characters of the remaining "
10565               .. "unprocessed text: `%s`.", str:sub(1,20))
10566         )
10567       else
10568         return res
10569       end
10570     end
10571   end
10572
10573   self.create_parser("parse_blocks",
10574                      function()
10575                        return parsers.blocks
10576                      end, true)
10577
10578   self.create_parser("parse_blocks_nested",
10579                      function()
10580                        return parsers.blocks_nested
10581                      end, false)
10582
10583   self.create_parser("parse_inlines",
10584                      function()
10585                        return parsers.inlines
10586                      end, false)
10587
10588   self.create_parser("parse_inlines_no_inline_note",
10589                      function()
10590                        return parsers.inlines_no_inline_note
10591                      end, false)
10592
10593   self.create_parser("parse_inlines_no_html",
10594                      function()
10595                        return parsers.inlines_no_html
10596                      end, false)
10597
10598   self.create_parser("parse_inlines_nbsp",
10599                      function()
10600                        return parsers.inlines_nbsp
10601                      end, false)
```

```
10602
10603   self.create_parser("parse_inlines_no_link_or_emphasis",
10604                      function()
10605                        return parsers.inlines_no_link_or_emphasis
10606                      end, false)
10607
10608   self.create_parser("parse_inlines_identity",
10609                      function()
10610                        return parsers.inlines_identity
10611                      end, false)
10612
10613   self.create_parser("parse_inlines_string",
10614                      function()
10615                        return parsers.inlines_string
10616                      end, false)
10617
10618   self.create_parser("parse_inlines_math",
10619                      function()
10620                        return parsers.inlines_math
10621                      end, false)
```

#### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
10622   parsers.minimally_indented_blankline
10623     = parsers.check_minimal_indent * (parsers.blankline / "")
10624
10625   parsers.minimally_indented_block
10626     = parsers.check_minimal_indent * V("Block")
10627
10628   parsers.minimally_indented_block_or_paragraph
10629     = parsers.check_minimal_indent * V("BlockOrParagraph")
10630
10631   parsers.minimally_indented_paragraph
10632     = parsers.check_minimal_indent * V("Paragraph")
10633
10634   parsers.minimally_indented_plain
10635     = parsers.check_minimal_indent * V("Plain")
10636
10637   parsers.minimally_indented_par_or_plain
10638     = parsers.minimally_indented_paragraph
10639     + parsers.minimally_indented_plain
10640
10641   parsers.minimally_indented_par_or_plain_no_blank
10642     = parsers.minimally_indented_par_or_plain
10643     - parsers.minimally_indented_blankline
10644
10645   parsers.minimally_indented_ref
10646     = parsers.check_minimal_indent * V("Reference")
10647
```

```
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
```

```
10695           + blank^0 / writer.paragraphsep
10696   end
10697
10698   parsers.sep_group_no_output = function(blank)
10699     return  blank^0 * parsers.eof
10700           + blank^0
10701   end
10702
10703   parsers.content_blank = parsers.minimally_indented_blankline
10704
10705   parsers.ref_or_block_separated
10706     = parsers.sep_group_no_output(parsers.content_blank)
10707     * ( parsers.minimally_indented_ref
10708       - parsers.content_blank)
10709     + parsers.block_sep_group(parsers.content_blank)
10710     * ( parsers.minimally_indented_block
10711       - parsers.content_blank)
10712
10713   parsers.loop_body_pair  =
10714     parsers.create_loop_body_pair(
10715       parsers.ref_or_block_separated,
10716       parsers.minimally_indented_par_or_plain_no_blank,
10717       parsers.block_sep_group(parsers.content_blank),
10718       parsers.par_sep_group(parsers.content_blank))
10719
10720   parsers.content_loop  = ( V("Block")
10721                           * parsers.loop_body_pair.block^0
10722                           + (V("Paragraph") + V("Plain"))
10723                           * parsers.ref_or_block_separated
10724                           * parsers.loop_body_pair.block^0
10725                           + (V("Paragraph") + V("Plain"))
10726                           * parsers.loop_body_pair.par^0)
10727                         * parsers.content_blank^0
10728
10729   parsers.indented_content = function()
10730     return  Ct( (V("Reference") + (parsers.blankline / ""))
10731               * parsers.content_blank^0
10732               * parsers.check_minimal_indent
10733               * parsers.content_loop
10734               + (V("Reference") + (parsers.blankline / ""))
10735               * parsers.content_blank^0
10736               + parsers.content_loop)
10737   end
10738
10739   parsers.add_indent = function(pattern, name, breakable)
10740     return  Cg(Cmt( Cb("indent_info")
10741                   * Ct(pattern)
10742                   * ( #parsers.linechar  -- check if starter is blank
10743                     * Cc(false) + Cc(true))
```

```
                    * Cc(name)
                    * Cc(breakable),
                process_starter_indent), "indent_info")
end
```

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

```
if options.hashEnumerators then
  parsers.dig = parsers.digit + parsers.hash
else
  parsers.dig = parsers.digit
end

parsers.enumerator = function(delimiter_type, interrupting)
  local delimiter_range
  local allowed_end
  if interrupting then
    delimiter_range = P("1")
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    delimiter_range = parsers.dig * parsers.dig^-8
    allowed_end = C(parsers.spacechar^1)
                + #(parsers.newline + parsers.eof)
  end

  return parsers.check_trail
           * Ct(C(delimiter_range) * C(delimiter_type))
           * allowed_end
end

parsers.starter = parsers.bullet(parsers.dash)
                + parsers.bullet(parsers.asterisk)
                + parsers.bullet(parsers.plus)
                + parsers.enumerator(parsers.period)
                + parsers.enumerator(parsers.rparent)
```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```
parsers.blockquote_start
  = parsers.check_trail
  * C(parsers.more)
  * C(parsers.spacechar^0)

parsers.blockquote_body
  = parsers.add_indent(parsers.blockquote_start, "bq", true)
  * parsers.indented_content()
  * remove_indent("bq")
```

```
  if not options.breakableBlockquotes then
    parsers.blockquote_body
      = parsers.add_indent(parsers.blockquote_start, "bq", false)
      * parsers.indented_content()
      * remove_indent("bq")
  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.

```
  local function parse_content_part(content_part)
    local rope = util.rope_to_string(content_part)
    local parsed
      = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
    parsed.indent_info = nil
    return parsed
  end

```

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.warning ~= nil then
            if next(content_part) ~= nil then
              content[#content + 1] = parse_content_part(content_part)
            end
            content_part = {}

            content[#content + 1] = value.warning
            value.warning = nil
          end
          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
```

```
10831           else
10832             content_part[#content_part + 1] = value.content
10833           end
10834           value.content = ''
10835           value.is_active = false
10836         end
10837       end
10838
10839       if next(content_part) ~= nil then
10840         content[#content + 1] = parse_content_part(content_part)
10841       end
10842
10843       return content
10844     end
10845
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
10846   local function fill_emph(t, opening_index, closing_index)
10847     local content
10848       = collect_emphasis_content(t, opening_index + 1,
10849                                  closing_index - 1)
10850     t[opening_index + 1].is_active = true
10851     t[opening_index + 1].rendered = writer.emphasis(content)
10852   end
10853
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
10854   local function fill_strong(t, opening_index, closing_index)
10855     local content
10856       = collect_emphasis_content(t, opening_index + 1,
10857                                  closing_index - 1)
10858     t[opening_index + 1].is_active = true
10859     t[opening_index + 1].rendered = writer.strong(content)
10860   end
10861
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
10862   local function breaks_three_rule(opening_delimiter, closing_delimiter)
10863     return ( opening_delimiter.is_closing
10864           or closing_delimiter.is_opening)
10865        and (( opening_delimiter.original_count
10866             + closing_delimiter.original_count) % 3 == 0)
10867        and ( opening_delimiter.original_count % 3 ~= 0
10868           or closing_delimiter.original_count % 3 ~= 0)
10869   end
10870
```

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`.

```
local find_emphasis_opener = function(t, bottom_index, latest_index,
                                      character, closing_delimiter)
  for i = latest_index, bottom_index, -1 do
    local value = t[i]
    if value.is_active and
       value.is_opening and
       value.type == "delimiter" and
       value.element == "emphasis" and
       (value.character == character) and
       (value.current_count > 0) then
      if not breaks_three_rule(value, closing_delimiter) then
        return i
      end
    end
  end
end

```

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
```

```
10963   end
10964
10965   parsers.delimiter_run = function(character)
10966     return  (B(parsers.backslash * character) + -B(character))
10967           * character^1
10968           * -#character
10969   end
10970
10971   parsers.left_flanking_delimiter_run = function(character)
10972     return  (B( parsers.any)
10973               * ( parsers.unicode.preceding_punctuation
10974                 + parsers.unicode.preceding_whitespace)
10975              + -B(parsers.any))
10976             * parsers.delimiter_run(character)
10977             * #parsers.unicode.punctuation
10978             + parsers.delimiter_run(character)
10979             * -#( #parsers.unicode.punctuation
10980                 + #parsers.unicode.whitespace
10981                 + parsers.eof)
10982   end
10983
10984   parsers.right_flanking_delimiter_run = function(character)
10985     return  parsers.unicode.preceding_punctuation
10986           * parsers.delimiter_run(character)
10987           * ( #parsers.unicode.punctuation
10988             + #parsers.unicode.whitespace
10989             + parsers.eof)
10990           + (B(parsers.any)
10991             * -( parsers.unicode.preceding_punctuation
10992                + parsers.unicode.preceding_whitespace))
10993           * parsers.delimiter_run(character)
10994   end
10995
10996   if options.underscores then
10997     parsers.emph_start
10998       = parsers.left_flanking_delimiter_run(parsers.asterisk)
10999       + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
11000         + ( parsers.unicode.preceding_punctuation
11001           * #parsers.right_flanking_delimiter_run(parsers.underscore)))
11002       * parsers.left_flanking_delimiter_run(parsers.underscore)
11003
11004     parsers.emph_end
11005       = parsers.right_flanking_delimiter_run(parsers.asterisk)
11006       + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
11007         + #( parsers.left_flanking_delimiter_run(parsers.underscore)
11008            * #parsers.unicode.punctuation))
11009       * parsers.right_flanking_delimiter_run(parsers.underscore)
11010   else
11011     parsers.emph_start
```

```
11012       = parsers.left_flanking_delimiter_run(parsers.asterisk)
11013
11014     parsers.emph_end
11015       = parsers.right_flanking_delimiter_run(parsers.asterisk)
11016   end
11017
11018   parsers.emph_capturing_open_and_close
11019     = #parsers.emph_start * #parsers.emph_end
11020     * Ct( Cg(Cc("delimiter"), "type")
11021         * Cg(Cc("emphasis"), "element")
11022         * Cg(C(parsers.emph_start), "content")
11023         * Cg(Cc(true), "is_opening")
11024         * Cg(Cc(true), "is_closing"))
11025
11026   parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
11027                                   * Cg(Cc("emphasis"), "element")
11028                                   * Cg(C(parsers.emph_start), "content")
11029                                   * Cg(Cc(true), "is_opening")
11030                                   * Cg(Cc(false), "is_closing"))
11031
11032   parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
11033                                    * Cg(Cc("emphasis"), "element")
11034                                    * Cg(C(parsers.emph_end), "content")
11035                                    * Cg(Cc(false), "is_opening")
11036                                    * Cg(Cc(true), "is_closing"))
11037
11038   parsers.emph_open_or_close  = parsers.emph_capturing_open_and_close
11039                               + parsers.emph_capturing_open
11040                               + parsers.emph_capturing_close
11041
11042   parsers.emph_open = parsers.emph_capturing_open_and_close
11043                     + parsers.emph_capturing_open
11044
11045   parsers.emph_close  = parsers.emph_capturing_open_and_close
11046                       + parsers.emph_capturing_close
11047
```

#### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
11048   -- List of references defined in the document
11049   local references
11050
11051   -- List of note references defined in the document
11052   parsers.rawnotes = {}
11053
```

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.

```
11054   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
    }
    return ""
  else
    local text
      = string.format('Multiply defined link reference "%s"', tag)
    local more = string.format("Look for the text `[%s]: ...`.", tag)
    return writer.warning(text, more)
  end
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
```

```
11101
11102  parsers.title_direct_ref  = parsers.title_s_direct_ref
11103                            + parsers.title_d_direct_ref
11104                            + parsers.title_p_direct_ref
11105
11106  parsers.inline_direct_ref_inside  = parsers.lparent * parsers.spnl
11107                                    * Cg(parsers.url + Cc(""), "url")
11108                                    * parsers.spnl
11109                                    * Cg( parsers.title_direct_ref
11110                                        + Cc(""), "title")
11111                                    * parsers.spnl * parsers.rparent
11112
11113  parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
11114                            * Cg(parsers.url + Cc(""), "url")
11115                            * parsers.spnlc
11116                            * Cg(parsers.title + Cc(""), "title")
11117                            * parsers.spnlc * parsers.rparent
11118
11119  parsers.empty_link  = parsers.lbracket
11120                      * parsers.rbracket
11121
11122  parsers.inline_link = parsers.link_text
11123                      * parsers.inline_direct_ref
11124
11125  parsers.full_link = parsers.link_text
11126                    * parsers.link_label
11127
11128  parsers.shortcut_link = parsers.link_label
11129                        * -(parsers.empty_link + parsers.link_label)
11130
11131  parsers.collapsed_link  = parsers.link_label
11132                          * parsers.empty_link
11133
11134  parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
11135                        * Cg(Cc("inline"), "link_type")
11136                        + #(parsers.exclamation * parsers.full_link)
11137                        * Cg(Cc("full"), "link_type")
11138                        + #( parsers.exclamation
11139                           * parsers.collapsed_link)
11140                        * Cg(Cc("collapsed"), "link_type")
11141                        + #(parsers.exclamation * parsers.shortcut_link)
11142                        * Cg(Cc("shortcut"), "link_type")
11143                        + #(parsers.exclamation * parsers.empty_link)
11144                        * Cg(Cc("empty"), "link_type")
11145
11146  parsers.link_opening  = #parsers.inline_link
11147                        * Cg(Cc("inline"), "link_type")
11148                        + #parsers.full_link
11149                        * 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")

local opening_elements = parsers.fail

if options.inlineNotes then
  opening_elements = opening_elements + inline_note_element
end

opening_elements = opening_elements + image_element

if options.notes then
  opening_elements = opening_elements + note_element
end

```

```
11199 opening_elements = opening_elements + link_element
11200
11201 parsers.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
11202                                  * Cg(Cc(true), "is_opening")
11203                                  * Cg(Cc(false), "is_closing")
11204                                  * opening_elements)
11205
11206 parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
11207                                  * Cg(Cc("link"), "element")
11208                                  * Cg(Cc(false), "is_opening")
11209                                  * Cg(Cc(true), "is_closing")
11210                                  * ( Cg(Cc(true), "is_direct")
11211                                    * Cg( parsers.rbracket
11212                                        * #parsers.inline_direct_ref,
11213                                         "content")
11214                                    + Cg(Cc(false), "is_direct")
11215                                    * Cg(parsers.rbracket, "content")))
11216
11217 parsers.link_image_open_or_close  = parsers.link_image_opening
11218                                   + parsers.link_image_closing
11219
11220 if options.html then
11221   parsers.link_emph_precedence  = parsers.inticks
11222                                 + parsers.autolink
11223                                 + parsers.html_inline_tags
11224 else
11225   parsers.link_emph_precedence  = parsers.inticks
11226                                 + parsers.autolink
11227 end
11228
11229 parsers.link_and_emph_endline = parsers.newline
11230                               * ((parsers.check_minimal_indent
11231                                 * -V("EndlineExceptions")
11232                                 + parsers.check_optional_indent
11233                                 * -V("EndlineExceptions")
11234                                 * -V("ListStarter")) / "")
11235                               * parsers.spacechar^0 / "\n"
11236
11237 parsers.link_and_emph_content
11238   = Ct( Cg(Cc("content"), "type")
11239       * Cg(Cs(( parsers.link_emph_precedence
11240               + parsers.backslash * parsers.linechar
11241               + parsers.link_and_emph_endline
11242               + (parsers.linechar
11243                 - parsers.blankline^2
11244                 - parsers.link_image_open_or_close
11245                 - parsers.emph_open_or_close))^0), "content"))
11246
11247 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,
          self.parser_functions.parse_inlines_string(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
```

```
11371       end
11372
11373       t[opening_index].rendered = rendered
11374       delete_parsed_content_in_range(t, opening_index + 1,
11375                                      closing_index)
11376       empty_content_in_range(t, opening_index, closing_index)
11377       disable_previous_link_openers(t, opening_index)
11378       disable_range(t, opening_index, closing_index)
11379     end
11380
```

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.

```
11381   local resolve_inline_following_content =
11382     function(t, closing_index, match_reference, match_link_attributes)
11383       local content = ""
11384       for i = closing_index + 1, #t do
11385         content = content .. t[i].content
11386       end
11387
11388       local matching_content = parsers.succeed
11389
11390       if match_reference then
11391         matching_content = matching_content
11392                          * parsers.inline_direct_ref_inside
11393       end
11394
11395       if match_link_attributes then
11396         matching_content = matching_content
11397                          * Cg(Ct(parsers.attributes^-1), "attributes")
11398       end
11399
11400       local matched = lpeg.match(Ct( matching_content
11401                                    * Cg(Cp(), "end_position")), content)
11402
11403       local matched_count = matched.end_position - 1
11404       for i = closing_index + 1, #t do
11405         local value = t[i]
11406
11407         local chars_left = matched_count
11408         matched_count = matched_count - #value.content
11409
11410         if matched_count <= 0 then
11411           value.content = value.content:sub(chars_left + 1)
11412           break
11413         end
11414
11415         value.content = ''
11416         value.is_active = false
```

```
11417       end
11418
11419       local attributes = matched.attributes
11420       if attributes == nil or next(attributes) == nil then
11421         attributes = nil
11422       end
11423
11424       return {
11425         url = matched.url or "",
11426         title = matched.title or "",
11427         attributes = attributes
11428       }
11429     end
11430
```

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.

```
11431   local function resolve_inline_link(t, opening_index, closing_index)
11432     local inline_content
11433       = resolve_inline_following_content(t, closing_index, true,
11434                                          t.match_link_attributes)
11435     render_link_or_image(t, opening_index, closing_index,
11436                          closing_index, inline_content)
11437   end
11438
```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
11439   local resolve_note_inline_link =
11440     function(t, opening_index, closing_index)
11441       local inline_content
11442         = resolve_inline_following_content(t, closing_index,
11443                                            false, false)
11444       render_link_or_image(t, opening_index, closing_index,
11445                            closing_index, inline_content)
11446     end
11447
```

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.

```
11448   local function resolve_shortcut_link(t, opening_index, closing_index)
11449     local content
11450       = collect_link_content(t, opening_index + 1, closing_index - 1)
11451     local r = self.lookup_reference(content)
11452
11453     if r then
11454       local inline_content
11455         = resolve_inline_following_content(t, closing_index, false,
11456                                            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)
    else
      local text = string.format('Undefined link reference "%s"',
                                 next_link_content)
      local more = string.format("Look for the text `[...][%s]`.",
                                 next_link_content)
      t[opening_index].warning = writer.warning(text, more)
    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)
  else
    local text = string.format('Undefined link reference "%s"',
                               content)
    local more = string.format("Look for the text `[%s][]`.",
                               content)
    t[opening_index].warning = writer.warning(text, more)
  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
```

```
11543         goto continue
11544       end
11545
11546       local opening_delimiter = t[opener_position]
11547       opening_delimiter.removed = true
11548
11549       local link_type = opening_delimiter.link_type
11550
11551       if (link_type == "inline") then
11552         resolve_inline_link(t, opener_position, i)
11553       end
11554       if (link_type == "shortcut") then
11555         resolve_shortcut_link(t, opener_position, i)
11556       end
11557       if (link_type == "full") then
11558         resolve_full_link(t, opener_position, i)
11559       end
11560       if (link_type == "collapsed") then
11561         resolve_collapsed_link(t, opener_position, i)
11562       end
11563       if (link_type == "note_inline") then
11564         resolve_note_inline_link(t, opener_position, i)
11565       end
11566       if (link_type == "raw_note") then
11567         resolve_raw_note_link(t, opener_position, i)
11568       end
11569
11570       ::continue::
11571     end
11572
11573     t[#t].content = t[#t].content:gsub("%s*$","")
11574
11575     process_emphasis(t, 1, #t)
11576     local final_result = collect_emphasis_content(t, 1, #t)
11577     return final_result
11578   end
11579
11580   function self.defer_link_and_emphasis_processing(delimiter_table)
11581     return writer.defer_call(function()
11582       return process_links_and_emphasis(delimiter_table)
11583     end)
11584   end
11585
```

#### 3.1.6.8 Inline Elements (local)

```
11586   parsers.Str      = ( parsers.normalchar
11587                        * (parsers.normalchar + parsers.at)^0)
11588                      / writer.string
11589
```

```
11590 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
11591                  / writer.string
11592
11593 parsers.Ellipsis = P("...") / writer.ellipsis
11594
11595 parsers.Smart    = parsers.Ellipsis
11596
11597 parsers.Code     = parsers.inticks / writer.code
11598
11599 if options.blankBeforeBlockquote then
11600   parsers.bqstart = parsers.fail
11601 else
11602   parsers.bqstart = parsers.blockquote_start
11603 end
11604
11605 if options.blankBeforeHeading then
11606   parsers.headerstart = parsers.fail
11607 else
11608   parsers.headerstart = parsers.atx_heading
11609 end
11610
11611 if options.blankBeforeList then
11612   parsers.interrupting_bullets = parsers.fail
11613   parsers.interrupting_enumerators = parsers.fail
11614 else
11615   parsers.interrupting_bullets
11616     = parsers.bullet(parsers.dash, true)
11617     + parsers.bullet(parsers.asterisk, true)
11618     + parsers.bullet(parsers.plus, true)
11619
11620   parsers.interrupting_enumerators
11621     = parsers.enumerator(parsers.period, true)
11622     + parsers.enumerator(parsers.rparent, true)
11623 end
11624
11625 if options.html then
11626   parsers.html_interrupting
11627     = parsers.check_trail
11628     * ( parsers.html_incomplete_open_tag
11629       + parsers.html_incomplete_close_tag
11630       + parsers.html_incomplete_open_special_tag
11631       + parsers.html_comment_start
11632       + parsers.html_cdatasection_start
11633       + parsers.html_declaration_start
11634       + parsers.html_instruction_start
11635       - parsers.html_close_special_tag
11636       - parsers.html_empty_special_tag)
11637 else
11638   parsers.html_interrupting = parsers.fail
```

```
11639 end
11640
11641 if options.blankBeforeHtmlBlock then
11642   parsers.html_interrupting = parsers.fail
11643 end
11644
11645 parsers.ListStarter = parsers.starter
11646
11647 parsers.EndlineExceptions
11648                    = parsers.blankline -- paragraph break
11649                    + parsers.eof       -- end of document
11650                    + parsers.bqstart
11651                    + parsers.thematic_break_lines
11652                    + parsers.interrupting_bullets
11653                    + parsers.interrupting_enumerators
11654                    + parsers.headerstart
11655                    + parsers.html_interrupting
11656
11657 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
11658
11659 parsers.endline = parsers.newline
11660                 * (parsers.check_minimal_indent
11661                   * -V("EndlineExceptions")
11662                   + parsers.check_optional_indent
11663                   * -V("EndlineExceptions")
11664                   * -V("ListStarter")) / function(_) return end
11665                 * parsers.spacechar^0
11666
11667 parsers.Endline = parsers.endline
11668                 / writer.soft_line_break
11669
11670 parsers.EndlineNoSub = parsers.endline
11671
11672 parsers.NoSoftLineBreakEndline
11673                       = parsers.newline
11674                       * (parsers.check_minimal_indent
11675                         * -V("NoSoftLineBreakEndlineExceptions")
11676                         + parsers.check_optional_indent
11677                         * -V("NoSoftLineBreakEndlineExceptions")
11678                         * -V("ListStarter"))
11679                       * parsers.spacechar^0
11680                       / writer.space
11681
11682 parsers.EndlineBreak = parsers.backslash * parsers.endline
11683                                          / writer.hard_line_break
11684
11685 parsers.OptionalIndent
11686                    = parsers.spacechar^1 / writer.space
11687
```

```
  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.spacechar
                       + parsers.html_space_entity_char)^1
                     * parsers.endline / writer.soft_line_break

  parsers.NonbreakingEndline
                       = parsers.endline
                       / writer.nbsp

  parsers.NonbreakingSpace
                   = parsers.spacechar^2 * parsers.endline
                                         / writer.nbsp
                   + parsers.spacechar^1
                   * parsers.endline^-1 * parsers.eof / ""
                   + parsers.spacechar^1 * parsers.endline
                                         * parsers.optionalspace
                                         / writer.nbsp
                   + parsers.spacechar^1 * parsers.optionalspace
                                         / writer.nbsp

```

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.

```
function self.auto_link_url(url, attributes)
  return writer.link(writer.escape(url),
                     url, nil, attributes)
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.

```
function self.auto_link_email(email, attributes)
  return writer.link(writer.escape(email),
                     "mailto:"..email,
                     nil, attributes)
end

  parsers.AutoLinkUrl = parsers.auto_link_url
                      / self.auto_link_url

```

```
11730  parsers.AutoLinkEmail
11731                      = parsers.auto_link_email
11732                      / self.auto_link_email
11733
11734  parsers.AutoLinkRelativeReference
11735                      = parsers.auto_link_relative_reference
11736                      / self.auto_link_url
11737
11738  parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
11739                      / self.defer_link_and_emphasis_processing
11740
11741  parsers.EscapedChar = parsers.backslash
11742                      * C(parsers.escapable) / writer.string
11743
11744  if options.htmlOutput == "commonmark" then
11745    parsers.InlineHtml
11746      = Cs(parsers.html_inline_comment)
11747      / self.parser_functions.parse_inlines_no_html
11748      / writer.inline_html_comment
11749      + Cs(parsers.html_any_empty_inline_tag)
11750      / writer.inline_html_empty_tag
11751      + Cs(parsers.html_inline_instruction)
11752      / writer.inline_html_processing_instruction
11753      + Cs(parsers.html_inline_cdatasection)
11754      / writer.inline_html_cdata_section
11755      + Cs(parsers.html_inline_declaration)
11756      / writer.inline_html_declaration
11757      + Cs(parsers.html_any_open_inline_tag)
11758      / writer.inline_html_open_tag
11759      + Cs(parsers.html_any_close_tag)
11760      / writer.inline_html_close_tag
11761  else
11762    parsers.InlineHtml = Cs(parsers.html_inline_comment)
11763                       / self.parser_functions.parse_inlines_no_html
11764                       / writer.inline_html_comment
11765                       + Cs( parsers.html_any_empty_inline_tag
11766                           + parsers.html_inline_instruction
11767                           + parsers.html_inline_cdatasection
11768                           + parsers.html_inline_declaration
11769                           + parsers.html_any_open_inline_tag
11770                           + parsers.html_any_close_tag)
11771                       / writer.inline_html_tag
11772  end
11773
11774  parsers.HtmlEntity = parsers.html_entities / writer.string
```

#### 3.1.6.9 Block Elements (local)

```
11775  if options.htmlOutput == "commonmark" then
11776    parsers.DisplayHtml
```

```
      = parsers.check_trail
      * ( parsers.html_comment
        / self.parser_functions.parse_blocks_nested
        / writer.block_html_comment)
      + Cs(parsers.check_trail * parsers.html_special_block)
      / writer.block_html_cdata_element
      + Cs(parsers.check_trail * parsers.html_block)
      / writer.block_html_pcdata_element
      + Cs(parsers.check_trail * parsers.html_any_block)
      / writer.block_html_standalone_tag
      + Cs(parsers.check_trail * parsers.html_instruction)
      / writer.block_html_processing_instruction
      + Cs(parsers.check_trail * parsers.html_cdatasection)
      / writer.block_html_cdata_section
      + Cs(parsers.check_trail * parsers.html_declaration)
      / writer.block_html_declaration
  else
    parsers.DisplayHtml = Cs( parsers.check_trail
                            * ( parsers.html_comment
                              + parsers.html_special_block
                              + parsers.html_block
                              + parsers.html_any_block
                              + parsers.html_instruction
                              + parsers.html_cdatasection
                              + parsers.html_declaration))
                        / writer.block_html_element
  end

  parsers.indented_non_blank_line = parsers.indentedline
                                  - parsers.blankline

  parsers.Verbatim
    = Cs( parsers.check_code_trail
        * (parsers.line - parsers.blankline)
        * (( parsers.check_minimal_blank_indent_and_full_code_trail
           * parsers.blankline)^0
          * ( (parsers.check_minimal_indent / "")
            * parsers.check_code_trail
            * (parsers.line - parsers.blankline))^1)^0)
    / self.expandtabs / writer.verbatim

  parsers.Blockquote   = parsers.blockquote_body
                       / writer.blockquote

  parsers.ThematicBreak = parsers.thematic_break_lines
                        / writer.thematic_break

  parsers.Reference    = parsers.define_reference_parser
                       / self.register_link
```

```
11826
11827  parsers.Paragraph   = parsers.freeze_trail
11828                      * (Ct((parsers.Inline)^1)
11829                      * (parsers.newline + parsers.eof)
11830                      * parsers.unfreeze_trail
11831                      / writer.paragraph)
11832
11833  parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
11834                       / writer.plain
```

#### 3.1.6.10 Lists (local)

```
11835
11836  if options.taskLists then
11837    parsers.tickbox = ( parsers.ticked_box
11838                      + parsers.halfticked_box
11839                      + parsers.unticked_box
11840                      ) / writer.tickbox
11841  else
11842     parsers.tickbox = parsers.fail
11843  end
11844
11845  parsers.list_blank = parsers.conditionally_indented_blankline
11846
11847  parsers.ref_or_block_list_separated
11848    = parsers.sep_group_no_output(parsers.list_blank)
11849    * parsers.minimally_indented_ref
11850    + parsers.block_sep_group(parsers.list_blank)
11851    * parsers.minimally_indented_block
11852
11853  parsers.ref_or_block_non_separated
11854    = parsers.minimally_indented_ref
11855    + (parsers.succeed / writer.interblocksep)
11856    * parsers.minimally_indented_block
11857    - parsers.minimally_indented_blankline
11858
11859  parsers.tight_list_loop_body_pair =
11860    parsers.create_loop_body_pair(
11861      parsers.ref_or_block_non_separated,
11862      parsers.minimally_indented_par_or_plain_no_blank,
11863      (parsers.succeed / writer.interblocksep),
11864      (parsers.succeed / writer.paragraphsep))
11865
11866  parsers.loose_list_loop_body_pair =
11867    parsers.create_loop_body_pair(
11868      parsers.ref_or_block_list_separated,
11869      parsers.minimally_indented_par_or_plain,
11870      parsers.block_sep_group(parsers.list_blank),
11871      parsers.par_sep_group(parsers.list_blank))
11872
```

```
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 / ""))
        * parsers.check_minimal_indent
        * parsers.loose_list_content_loop
        + (V("Reference") + (parsers.blankline / ""))
        + (parsers.tickbox^-1 / writer.escape)
        * parsers.loose_list_content_loop))

parsers.TightListItem = function(starter)
```

```
11922     return  -parsers.ThematicBreak
11923             * parsers.add_indent(starter, "li")
11924             * parsers.indented_content_tight
11925   end
11926
11927   parsers.LooseListItem = function(starter)
11928     return  -parsers.ThematicBreak
11929             * parsers.add_indent(starter, "li")
11930             * parsers.indented_content_loose
11931             * remove_indent("li")
11932   end
11933
11934   parsers.BulletListOfType = function(bullet_type)
11935     local bullet = parsers.bullet(bullet_type)
11936     return  ( Ct( parsers.TightListItem(bullet)
11937                 * ( (parsers.check_minimal_indent / "")
11938                   * parsers.TightListItem(bullet)
11939                   )^0
11940               )
11941               * Cc(true)
11942               * -#( (parsers.list_blank^0 / "")
11943                   * parsers.check_minimal_indent
11944                   * (bullet - parsers.ThematicBreak)
11945               )
11946               + Ct( parsers.LooseListItem(bullet)
11947                   * ( (parsers.list_blank^0 / "")
11948                     * (parsers.check_minimal_indent / "")
11949                     * parsers.LooseListItem(bullet)
11950                     )^0
11951               )
11952               * Cc(false)
11953             ) / writer.bulletlist
11954   end
11955
11956   parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11957                      + parsers.BulletListOfType(parsers.asterisk)
11958                      + parsers.BulletListOfType(parsers.plus)
11959
11960   local function ordered_list(items,tight,starter)
11961     local startnum = starter[2][1]
11962     if options.startNumber then
11963       startnum = tonumber(startnum) or 1  -- fallback for '#'
11964       if startnum ~= nil then
11965         startnum = math.floor(startnum)
11966       end
11967     else
11968       startnum = nil
11969     end
11970     return writer.orderedlist(items,tight,startnum)
```

```
11971   end
11972
11973   parsers.OrderedListOfType = function(delimiter_type)
11974     local enumerator = parsers.enumerator(delimiter_type)
11975     return  Cg(enumerator, "listtype")
11976           * (Ct( parsers.TightListItem(Cb("listtype"))
11977                * ( (parsers.check_minimal_indent / "")
11978                  * parsers.TightListItem(enumerator))^0)
11979           * Cc(true)
11980           * -#((parsers.list_blank^0 / "")
11981               * parsers.check_minimal_indent * enumerator)
11982           + Ct( parsers.LooseListItem(Cb("listtype"))
11983               * ((parsers.list_blank^0 / "")
11984                 * (parsers.check_minimal_indent / "")
11985                 * parsers.LooseListItem(enumerator))^0)
11986           * Cc(false)
11987           ) * Ct(Cb("listtype")) / ordered_list
11988   end
11989
11990   parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11991                       + parsers.OrderedListOfType(parsers.rparent)
```

#### 3.1.6.11 Blank (local)

```
11992   parsers.Blank        = parsers.blankline / ""
11993                        + V("Reference")
```

#### 3.1.6.12 Headings (local)

```
11994   function parsers.parse_heading_text(s)
11995     local inlines = self.parser_functions.parse_inlines(s)
11996     local flatten_inlines = self.writer.flatten_inlines
11997     self.writer.flatten_inlines = true
11998     local flat_text = self.parser_functions.parse_inlines(s)
11999     flat_text = util.rope_to_string(flat_text)
12000     self.writer.flatten_inlines = flatten_inlines
12001     return {flat_text, inlines}
12002   end
12003
12004   -- parse atx header
12005   parsers.AtxHeading = parsers.check_trail_no_rem
12006                      * Cg(parsers.heading_start, "level")
12007                      * ((C( parsers.optionalspace
12008                           * parsers.hash^0
12009                           * parsers.optionalspace
12010                           * parsers.newline)
12011                         + parsers.spacechar^1
12012                         * C(parsers.line))
12013                        / strip_atx_end
12014                        / parsers.parse_heading_text)
```

```
                    * Cb("level")
                    / writer.heading

parsers.heading_line  = parsers.linechar^1
                      - parsers.thematic_break_lines

parsers.heading_text = parsers.heading_line
                     * ( (V("Endline") / "\n")
                       * ( parsers.heading_line
                         - parsers.heading_level))^0
                     * parsers.newline^-1

parsers.SetextHeading = parsers.freeze_trail
                      * parsers.check_trail_no_rem
                      * #( parsers.heading_text
                         * parsers.check_minimal_indent
                         * parsers.check_trail
                         * parsers.heading_level)
                      * Cs(parsers.heading_text)
                      / parsers.parse_heading_text
                      * parsers.check_minimal_indent_and_trail
                      * parsers.heading_level
                      * parsers.newline
                      * parsers.unfreeze_trail
                      / writer.heading

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.

```
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.

```
  local walkable_syntax = (function(global_walkable_syntax)
    local local_walkable_syntax = {}
    for lhs, rule in pairs(global_walkable_syntax) do
      local_walkable_syntax[lhs] = util.table_copy(rule)
    end
    return local_walkable_syntax
  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.

```
12050     local current_extension_name = nil
12051     self.insert_pattern = function(selector, pattern, pattern_name)
12052       assert(pattern_name == nil or type(pattern_name) == "string")
12053       local _, _, lhs, pos, rhs
12054         = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
12055       assert(lhs ~= nil,
12056         [[Expected selector in form ]]
12057         .. [["LHS (before|after|instead of) RHS", not "]]
12058         .. selector .. [["]])
12059       assert(walkable_syntax[lhs] ~= nil,
12060         [[Rule ]] .. lhs
12061         .. [[ -> ... does not exist in markdown grammar]])
12062       assert(pos == "before" or pos == "after" or pos == "instead of",
12063         [[Expected positional specifier "before", "after", ]]
12064         .. [[or "instead of", not "]]
12065         .. pos .. [["]])
12066       local rule = walkable_syntax[lhs]
12067       local index = nil
12068       for current_index, current_rhs in ipairs(rule) do
12069         if type(current_rhs) == "string" and current_rhs == rhs then
12070           index = current_index
12071           if pos == "after" then
12072             index = index + 1
12073           end
12074           break
12075         end
12076       end
12077       assert(index ~= nil,
12078         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
12079           .. [[ does not exist in markdown grammar]])
12080       local accountable_pattern
12081       if current_extension_name then
12082         accountable_pattern
12083           = {pattern, current_extension_name, pattern_name}
12084       else
12085         assert(type(pattern) == "string",
12086           [[reader->insert_pattern() was called outside ]]
12087           .. [[an extension with ]]
12088           .. [[a PEG pattern instead of a rule name]])
12089         accountable_pattern = pattern
12090       end
12091       if pos == "instead of" then
12092         rule[index] = accountable_pattern
12093       else
12094         table.insert(rule, index, accountable_pattern)
12095       end
12096       -- TODO: Remove all occurrences of `pattern` after `index`
12097       --       to improve speed?
12098     end
```

```
12099     if options.htmlOverLinks then
12100       self.insert_pattern("Inline before AutoLinkUrl", "InlineHtml")
12101     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.

```
12102     local syntax =
12103       { "Blocks",
12104
12105         Blocks = V("InitializeState")
12106                * V("ExpectedJekyllData")
12107                * 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.

```
12108                * ( V("Block")
12109                  * ( V("Blank")^0 * parsers.eof
12110                    + ( V("Blank")^2 / writer.paragraphsep
12111                      + V("Blank")^0 / writer.interblocksep
12112                      )
12113                    )
12114                  + ( V("Paragraph") + V("Plain") )
12115                  * ( V("Blank")^0 * parsers.eof
12116                    + ( V("Blank")^2 / writer.paragraphsep
12117                      + V("Blank")^0 / writer.interblocksep
12118                      )
12119                    )
12120                  * V("Block")
12121                  * ( V("Blank")^0 * parsers.eof
12122                    + ( V("Blank")^2 / writer.paragraphsep
12123                      + V("Blank")^0 / writer.interblocksep
12124                      )
12125                    )
12126                  + ( V("Paragraph") + V("Plain") )
12127                  * ( V("Blank")^0 * parsers.eof
12128                    + V("Blank")^0 / writer.paragraphsep
12129                    )
12130                  )^0,
12131
12132         ExpectedJekyllData = parsers.succeed,
12133
12134         Blank              = parsers.Blank,
12135         Reference          = parsers.Reference,
12136
12137         Blockquote         = parsers.Blockquote,
12138         Verbatim           = parsers.Verbatim,
12139         ThematicBreak      = parsers.ThematicBreak,
12140         BulletList         = parsers.BulletList,
12141         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,
      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
```

```
12186           = previous_accountable_pattern[2]
12187           .. ", " .. current_extension_name
12188       else
12189         previous_pattern = nil
12190         extension_name = current_extension_name
12191       end
12192       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
```

```
12193       if type(get_pattern) == "function" then
12194         pattern = get_pattern(previous_pattern)
12195       else
12196         assert(previous_pattern == nil,
12197                [[Rule ]] .. rule_name ..
12198                [[ has already been updated by ]] .. extension_name)
12199         pattern = get_pattern
12200       end
12201       local accountable_pattern = { pattern, extension_name, rule_name }
12202       walkable_syntax[rule_name] = { accountable_pattern }
12203     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.

```
12204     local special_characters = {}
12205     self.add_special_character = function(c)
12206       table.insert(special_characters, c)
12207       syntax.SpecialChar = S(table.concat(special_characters, ""))
12208     end
12209
12210     self.add_special_character("*")
12211     self.add_special_character("[")
12212     self.add_special_character("]")
12213     self.add_special_character("<")
12214     self.add_special_character("!")
12215     self.add_special_character("\\")
```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```
12216     self.initialize_named_group = function(name, value)
12217       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
```

```
12262             output_line = output_line .. ","
12263           end
12264           table.insert(output_lines, output_line)
12265         end
12266         local output_line = "    ]"
12267         if lhs_index < #sorted_lhs then
12268           output_line = output_line .. ","
12269         end
12270         table.insert(output_lines, output_line)
12271       end
12272       table.insert(output_lines, "}")
12273
12274       local output = table.concat(output_lines, "\n")
12275       local output_filename = options.debugExtensionsFileName
12276       local output_file = assert(io.open(output_filename, "w"),
12277         [[Could not open file "]] .. output_filename
12278         .. [[" for writing]])
12279       assert(output_file:write(output))
12280       assert(output_file:close())
12281     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`.

```
12282     for lhs, rule in pairs(walkable_syntax) do
12283       syntax[lhs] = parsers.fail
12284       for _, rhs in ipairs(rule) do
12285         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.

```
12286         if type(rhs) == "string" then
12287           pattern = V(rhs)
12288         else
12289           pattern = rhs[1]
12290           if type(pattern) == "string" then
12291             pattern = V(pattern)
12292           end
12293         end
12294         syntax[lhs] = syntax[lhs] + pattern
12295       end
12296     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.

```
12297     if options.underscores then
12298       self.add_special_character("_")
```

```
12299     end
12300
12301     if not options.codeSpans then
12302       syntax.Code = parsers.fail
12303     else
12304       self.add_special_character("`")
12305     end
12306
12307     if not options.html then
12308       syntax.DisplayHtml = parsers.fail
12309       syntax.InlineHtml = parsers.fail
12310       syntax.HtmlEntity  = parsers.fail
12311     else
12312       self.add_special_character("&")
12313     end
12314
12315     if options.preserveTabs then
12316       options.stripIndent = false
12317     end
12318
12319     if not options.smartEllipses then
12320       syntax.Smart = parsers.fail
12321     else
12322       self.add_special_character(".")
12323     end
12324
12325     if not options.relativeReferences then
12326       syntax.AutoLinkRelativeReference = parsers.fail
12327     end
12328
12329     if options.contentLevel == "inline" then
12330       syntax[1] = "Inlines"
12331       syntax.Inlines = V("InitializeState")
12332                        * parsers.Inline^0
12333                        * ( parsers.spacing^0
12334                          * parsers.eof / "")
12335       syntax.Space = parsers.Space + parsers.blankline / writer.space
12336     end
12337
12338     local blocks_nested_t = util.table_copy(syntax)
12339     blocks_nested_t.ExpectedJekyllData = parsers.succeed
12340     parsers.blocks_nested = Ct(blocks_nested_t)
12341
12342     parsers.blocks = Ct(syntax)
12343
12344     local inlines_t = util.table_copy(syntax)
12345     inlines_t[1] = "Inlines"
12346     inlines_t.Inlines = V("InitializeState")
12347                       * 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)

    local inlines_identity_t = util.table_copy(inlines_t)
    inlines_identity_t.Str = parsers.any / function(s) return s end
    parsers.inlines_identity = Ct(inlines_identity_t)

    local inlines_string_t = util.table_copy(inlines_t)
    inlines_string_t.Str = parsers.any / writer.string
    parsers.inlines_string = Ct(inlines_string_t)

    local inlines_math_t = util.table_copy(inlines_t)
    inlines_math_t.Str = parsers.any / writer.math
    parsers.inlines_math = Ct(inlines_math_t)

    local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
    inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
    inlines_no_link_or_emphasis_t.EndlineExceptions
      = parsers.EndlineExceptions - parsers.eof
    parsers.inlines_no_link_or_emphasis
      = Ct(inlines_no_link_or_emphasis_t)
```

Return a function that converts markdown string `input` into a plain TEX output and returns it..

```
    return function(input)
```

Normalize the input.

```
      if options.unicodeNormalization then
        local form = options.unicodeNormalizationForm
        input = util.normalize(input, form)
      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.

```
12390       input = input:gsub("\r\n?", "\n")
12391       if input:sub(-1) ~= "\n" then
12392         input = input .. "\n"
12393       end
```

Clear the table of references.

```
12394       references = {}
12395       local document = self.parser_functions.parse_blocks(input)
12396       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`.

```
12397       local undosep_start, undosep_end
12398       local potential_secend_start, secend_start
12399       local potential_sep_start, sep_start
12400       while true do
12401         -- find a `writer->undosep`
12402         undosep_start, undosep_end
12403           = output:find(writer.undosep_text, 1, true)
12404         if undosep_start == nil then break end
12405         -- skip any preceding section ends
12406         secend_start = undosep_start
12407         while true do
12408           potential_secend_start = secend_start - #writer.secend_text
12409           if potential_secend_start < 1
12410              or output:sub(potential_secend_start,
12411                            secend_start - 1) ~= writer.secend_text
12412              then
12413             break
12414           end
12415           secend_start = potential_secend_start
12416         end
12417         -- find an immediately preceding
12418         -- block element / paragraph separator
12419         sep_start = secend_start
12420         potential_sep_start = sep_start - #writer.interblocksep_text
12421         if potential_sep_start >= 1
12422            and output:sub(potential_sep_start,
12423                           sep_start - 1) == writer.interblocksep_text
12424            then
12425           sep_start = potential_sep_start
12426         else
12427           potential_sep_start = sep_start - #writer.paragraphsep_text
12428           if potential_sep_start >= 1
12429              and output:sub(potential_sep_start,
12430                             sep_start - 1) == writer.paragraphsep_text
12431              then
12432             sep_start = potential_sep_start
12433           end
```

```
        end
        -- remove `writer->undosep` and immediately preceding
        -- block element / paragraph separator
        output = output:sub(1, sep_start - 1)
              .. output:sub(secend_start, undosep_start - 1)
              .. output:sub(undosep_end + 1)
      end
      return output
    end
  end
  return self
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 Acronyms

The `extensions.acronyms` function recognizes acronyms, initialisms, and other all-caps sequences in markdown text.

```
M.extensions.acronyms = function(acronyms)
  assert(#acronyms > 0)
  return {
    name = "built-in acronyms syntax extension",
    extend_writer = function(self)
```

Define `writer->acronym` as a function that will transform an input acronym, initialism, or another all-caps sequence to the output format.

```
      function self.acronym(s)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererAcronym{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local options = self.options
      local writer = self.writer
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of all registered acronyms, initialisms, and other all-caps sequences.

```
      local prefix_tree = {_type = "intermediate"}
      for _, acronym in ipairs(acronyms) do
        local node = prefix_tree
```

Normalize each acronym, initialism, or another all-caps sequence.

```
12463         if options.unicodeNormalization then
12464           local form = options.unicodeNormalizationForm
12465           acronym = util.normalize(acronym, form)
12466         end
12467         for i = 1, #acronym do
12468           local acronym_byte = acronym:sub(i, i)
12469           if i < #acronym then
12470             if node[acronym_byte] == nil then
12471               node[acronym_byte] = {_type = "intermediate"}
12472             end
12473             node = node[acronym_byte]
12474           else
12475             table.insert(node, {acronym_byte, _type = "leaf"})
12476           end
12477         end
12478       end
```

Next, we will construct a parser out of the prefix tree.

```
12479       local subparsers = {}
12480       depth_first_search(prefix_tree, "", function(node, path)
12481         if node._type == "leaf" then
12482           local acronym_byte = table.unpack(node)
12483           if subparsers[path] == nil then
12484             subparsers[path] = parsers.fail
12485           end
12486           subparsers[path] = subparsers[path] + P(acronym_byte)
12487         end
12488       end, function(_, path)
12489         if #path > 0 then
12490           local byte = path:sub(#path, #path)
12491           local parent_path = path:sub(1, #path-1)
12492           local prefix = P(byte)
12493           local suffix = prefix * subparsers[path]
12494           if subparsers[parent_path] == nil then
12495             subparsers[parent_path] = parsers.fail
12496           end
12497           subparsers[parent_path] = subparsers[parent_path] + suffix
12498         end
12499       end)
12500       assert(subparsers[""] ~= nil)
```

Only match acronyms at word boundaries.

```
12501       local Acronym = ( parsers.unicode.punctuation
12502                       - V("SpecialChar")
12503                       + parsers.unicode.whitespace
12504                       - V("EndlineExceptions"))^0
12505                     / writer.string
12506                     * (
12507                       subparsers[""]
```

```
                      * #(-parsers.normalchar
                         + parsers.unicode.punctuation
                         + parsers.unicode.whitespace
                         + parsers.eof)
                      / writer.acronym)
      self.insert_pattern("Inline before Str", Acronym, "Acronym")
    end
  }
end
%
%#### Bracketed Spans
%
% The \luamdef{extensions.bracketed_spans} function implements the Pandoc
% bracketed span syntax extension.
%
% \end{markdown}
%  \begin{macrocode}
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),
                "\\markdownRendererBracketedSpan{",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(util.find_files(language_map)) 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
          base = curr
        else
          setmetatable(prev, { __index = curr })
        end
        prev = curr
      end
      ::continue::
    end
```

```
12767     return base or {}
12768   end)()
12769
12770   return {
12771     name = "built-in content_blocks syntax extension",
12772     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.

```
12773       function self.contentblock(src,suf,type,tit)
12774         if not self.is_writing then return "" end
12775         src = src.."."..suf
12776         suf = suf:lower()
12777         if type == "onlineimage" then
12778           return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
12779                   "{",self.string(src),"}",
12780                   "{",self.uri(src),"}",
12781                   "{",self.string(tit or ""),"}"}
12782         elseif languages_json[suf] then
12783           return {"\\markdownRendererContentBlockCode{",suf,"}",
12784                   "{",self.string(languages_json[suf]),"}",
12785                   "{",self.string(src),"}",
12786                   "{",self.uri(src),"}",
12787                   "{",self.string(tit or ""),"}"}
12788         else
12789           return {"\\markdownRendererContentBlock{",suf,"}",
12790                   "{",self.string(src),"}",
12791                   "{",self.uri(src),"}",
12792                   "{",self.string(tit or ""),"}"}
12793         end
12794       end
12795     end, extend_reader = function(self)
12796       local parsers = self.parsers
12797       local writer = self.writer
12798
12799       local contentblock_tail
12800                   = parsers.optionaltitle
12801                   * (parsers.newline + parsers.eof)
12802
12803       -- case insensitive online image suffix:
12804       local onlineimagesuffix
12805                   = (function(...)
12806                       local parser = nil
12807                       for _, suffix in ipairs({...}) do
12808                         local pattern=nil
12809                         for i=1,#suffix do
12810                           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
```

```
12898           return {"\\markdownRendererDlBeginTight\n", buffer,
12899             "\n\\markdownRendererDlEndTight"}
12900         else
12901           return {"\\markdownRendererDlBegin\n", buffer,
12902             "\n\\markdownRendererDlEnd"}
12903         end
12904       end
12905     end, extend_reader = function(self)
12906       local parsers = self.parsers
12907       local writer = self.writer
12908
12909       local defstartchar = S("~:")
12910
12911       local defstart
12912         = parsers.check_trail_length(0) * defstartchar
12913         * #parsers.spacing
12914         * (parsers.tab + parsers.space^-3)
12915         + parsers.check_trail_length(1)
12916         * defstartchar * #parsers.spacing
12917         * (parsers.tab + parsers.space^-2)
12918         + parsers.check_trail_length(2)
12919         * defstartchar * #parsers.spacing
12920         * (parsers.tab + parsers.space^-1)
12921         + parsers.check_trail_length(3)
12922         * defstartchar * #parsers.spacing
12923
12924       local indented_line
12925         = (parsers.check_minimal_indent / "")
12926         * parsers.check_code_trail * parsers.line
12927
12928       local blank
12929         = parsers.check_minimal_blank_indent_and_any_trail
12930         * parsers.optionalspace * parsers.newline
12931
12932       local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
12933
12934       local indented_blocks = function(bl)
12935         return Cs( bl
12936               * (blank^1 * (parsers.check_minimal_indent / "")
12937                 * parsers.check_code_trail * -parsers.blankline * bl)^0
12938               * (blank^1 + parsers.eof))
12939       end
12940
12941       local function definition_list_item(term, defs, _)
12942         return { term = self.parser_functions.parse_inlines(term),
12943                  definitions = defs }
12944       end
12945
12946       local DefinitionListItemLoose
```

```
12947         = C(parsers.line) * blank^0
12948         * Ct((parsers.check_minimal_indent * (defstart
12949             * indented_blocks(dlchunk)
12950             / self.parser_functions.parse_blocks_nested))^1)
12951         * Cc(false) / definition_list_item
12952
12953       local DefinitionListItemTight
12954         = C(parsers.line)
12955         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12956             / self.parser_functions.parse_blocks_nested))^1)
12957         * Cc(true) / definition_list_item
12958
12959       local DefinitionList
12960         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
12961           + Ct(DefinitionListItemTight^1)
12962           * (blank^0
12963             * -DefinitionListItemLoose * Cc(true))
12964           ) / writer.definitionlist
12965
12966       self.insert_pattern("Block after Heading",
12967                           DefinitionList, "DefinitionList")
12968     end
12969   }
12970 end
```

#### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```
12971 M.extensions.fancy_lists = function()
12972   return {
12973     name = "built-in fancy_lists syntax extension",
12974     extend_writer = function(self)
12975       local options = self.options
12976
```

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.

```
12977       function self.fancylist(items,tight,startnum,numstyle,numdelim)
12978         if not self.is_writing then return "" end
12979         local buffer = {}
12980         local num = startnum
12981         for _,item in ipairs(items) do
12982           if item ~= "" then
12983             buffer[#buffer + 1] = self.fancyitem(item,num)
12984           end
12985           if num ~= nil and item ~= "" then
12986             num = num + 1
12987           end
12988         end
12989         local contents = util.intersperse(buffer,"\n")
12990         if tight and options.tightLists then
12991           return {"\\markdownRendererFancyOlBeginTight{",
12992                   numstyle,"}{",numdelim,"}",contents,
12993                   "\n\\markdownRendererFancyOlEndTight "}
12994         else
12995           return {"\\markdownRendererFancyOlBegin{",
12996                   numstyle,"}{",numdelim,"}",contents,
12997                   "\n\\markdownRendererFancyOlEnd "}
12998         end
12999       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.

```
13000       function self.fancyitem(s,num)
13001         if num ~= nil then
13002           return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
13003                   "\\markdownRendererFancyOlItemEnd "}
13004         else
13005           return {"\\markdownRendererFancyOlItem ",s,
13006                   "\\markdownRendererFancyOlItemEnd "}
13007         end
13008       end
13009     end, extend_reader = function(self)
13010       local parsers = self.parsers
13011       local options = self.options
13012       local writer = self.writer
13013
13014       local function combine_markers_and_delims(markers, delims)
```

```
13015         local markers_table = {}
13016         for _,marker in ipairs(markers) do
13017           local start_marker
13018           local continuation_marker
13019           if type(marker) == "table" then
13020             start_marker = marker[1]
13021             continuation_marker = marker[2]
13022           else
13023             start_marker = marker
13024             continuation_marker = marker
13025           end
13026           for _,delim in ipairs(delims) do
13027             table.insert(markers_table,
13028                          {start_marker, continuation_marker, delim})
13029           end
13030         end
13031         return markers_table
13032       end
13033
13034       local function join_table_with_func(func, markers_table)
13035         local pattern = func(table.unpack(markers_table[1]))
13036         for i = 2, #markers_table do
13037           pattern = pattern + func(table.unpack(markers_table[i]))
13038         end
13039         return pattern
13040       end
13041
13042       local lowercase_letter_marker = R("az")
13043       local uppercase_letter_marker = R("AZ")
13044
13045       local roman_marker = function(chars)
13046         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
13047         local l, x, v, i
13048           = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
13049         return  m^-3
13050                 * (c*m + c*d + d^-1 * c^-3)
13051                 * (x*c + x*l + l^-1 * x^-3)
13052                 * (i*x + i*v + v^-1 * i^-3)
13053       end
13054
13055       local lowercase_roman_marker
13056         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
13057       local uppercase_roman_marker
13058         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
13059
13060       local lowercase_opening_roman_marker  = P("i")
13061       local uppercase_opening_roman_marker  = P("I")
13062
13063       local digit_marker = parsers.dig * parsers.dig^-8
```

```
13064
13065       local markers = {
13066         {lowercase_opening_roman_marker, lowercase_roman_marker},
13067         {uppercase_opening_roman_marker, uppercase_roman_marker},
13068         lowercase_letter_marker,
13069         uppercase_letter_marker,
13070         lowercase_roman_marker,
13071         uppercase_roman_marker,
13072         digit_marker
13073       }
13074
13075       local delims = {
13076         parsers.period,
13077         parsers.rparent
13078       }
13079
13080       local markers_table = combine_markers_and_delims(markers, delims)
13081
13082       local function enumerator(start_marker, _,
13083                                 delimiter_type, interrupting)
13084         local delimiter_range
13085         local allowed_end
13086         if interrupting then
13087           delimiter_range = P("1")
13088           allowed_end = C(parsers.spacechar^1) * #parsers.linechar
13089         else
13090           delimiter_range = start_marker
13091           allowed_end = C(parsers.spacechar^1)
13092                       + #(parsers.newline + parsers.eof)
13093         end
13094
13095         return parsers.check_trail
13096                 * Ct(C(delimiter_range) * C(delimiter_type))
13097                 * allowed_end
13098       end
13099
13100       local starter = join_table_with_func(enumerator, markers_table)
13101
13102       local TightListItem = function(starter)
13103         return  parsers.add_indent(starter, "li")
13104                 * parsers.indented_content_tight
13105       end
13106
13107       local LooseListItem = function(starter)
13108         return  parsers.add_indent(starter, "li")
13109                 * parsers.indented_content_loose
13110                 * remove_indent("li")
13111       end
13112
```

```
13113       local function roman2number(roman)
13114         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
13115                          ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
13116         local numeral = 0
13117
13118         local i = 1
13119         local len = string.len(roman)
13120         while i < len do
13121           local z1, z2 = romans[ string.sub(roman, i, i) ],
13122                          romans[ string.sub(roman, i+1, i+1) ]
13123           if z1 < z2 then
13124               numeral = numeral + (z2 - z1)
13125               i = i + 2
13126           else
13127               numeral = numeral + z1
13128               i = i + 1
13129           end
13130         end
13131         if i <= len then
13132           numeral = numeral + romans[ string.sub(roman,i,i) ]
13133         end
13134         return numeral
13135       end
13136
13137       local function sniffstyle(numstr, delimend)
13138         local numdelim
13139         if delimend == ")" then
13140           numdelim = "OneParen"
13141         elseif delimend == "." then
13142           numdelim = "Period"
13143         else
13144           numdelim = "Default"
13145         end
13146
13147         local num
13148         num = numstr:match("^([I])$")
13149         if num then
13150           return roman2number(num), "UpperRoman", numdelim
13151         end
13152         num = numstr:match("^([i])$")
13153         if num then
13154           return roman2number(string.upper(num)), "LowerRoman", numdelim
13155         end
13156         num = numstr:match("^([A-Z])$")
13157         if num then
13158           return string.byte(num) - string.byte("A") + 1,
13159                  "UpperAlpha", numdelim
13160         end
13161         num = numstr:match("^([a-z])$")
```

```
13162         if num then
13163           return string.byte(num) - string.byte("a") + 1,
13164                  "LowerAlpha", numdelim
13165         end
13166         num = numstr:match("^([IVXLCDM]+)")
13167         if num then
13168           return roman2number(num), "UpperRoman", numdelim
13169         end
13170         num = numstr:match("^([ivxlcdm]+)")
13171         if num then
13172           return roman2number(string.upper(num)), "LowerRoman", numdelim
13173         end
13174         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
13175       end
13176
13177       local function fancylist(items,tight,start)
13178         local startnum, numstyle, numdelim
13179           = sniffstyle(start[2][1], start[2][2])
13180         return writer.fancylist(items,tight,
13181                                 options.startNumber and startnum or 1,
13182                                 numstyle or "Decimal",
13183                                 numdelim or "Default")
13184       end
13185
13186       local FancyListOfType
13187         = function(start_marker, continuation_marker, delimiter_type)
13188           local enumerator_start
13189             = enumerator(start_marker, continuation_marker,
13190                          delimiter_type)
13191           local enumerator_cont
13192             = enumerator(continuation_marker, continuation_marker,
13193                          delimiter_type)
13194           return Cg(enumerator_start, "listtype")
13195                * (Ct( TightListItem(Cb("listtype"))
13196                    * ((parsers.check_minimal_indent / "")
13197                    * TightListItem(enumerator_cont))^0)
13198                * Cc(true)
13199                * -#((parsers.conditionally_indented_blankline^0 / "")
13200                    * parsers.check_minimal_indent * enumerator_cont)
13201                + Ct( LooseListItem(Cb("listtype"))
13202                    * ((parsers.conditionally_indented_blankline^0 / "")
13203                      * (parsers.check_minimal_indent / "")
13204                      * LooseListItem(enumerator_cont))^0)
13205                * Cc(false)
13206                ) * Ct(Cb("listtype")) / fancylist
13207         end
13208
13209       local FancyList
13210         = join_table_with_func(FancyListOfType, markers_table)
```

```
13211
13212       local ListStarter = starter
13213
13214       self.update_rule("OrderedList", FancyList)
13215       self.update_rule("ListStarter", ListStarter)
13216     end
13217   }
13218 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.

```
13219 M.extensions.fenced_code = function(blank_before_code_fence,
13220                                       allow_attributes,
13221                                       allow_raw_blocks)
13222   return {
13223     name = "built-in fenced_code syntax extension",
13224     extend_writer = function(self)
13225       local options = self.options
13226
```

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.

```
13227       function self.fencedCode(s, i, attr)
13228         if not self.is_writing then return "" end
13229         s = s:gsub("\n$", "")
13230         local buf = {}
13231         if attr ~= nil then
13232           table.insert(buf,
13233             {"\\markdownRendererFencedCodeAttributeContextBegin",
13234              self.attributes(attr)})
13235         end
13236         local name = util.cache_verbatim(options.cacheDir, s)
13237         table.insert(buf,
13238           {"\\markdownRendererInputFencedCode{",
13239            name,"}{",self.string(i),"}{",self.infostring(i),"}"})
13240         if attr ~= nil then
13241           table.insert(buf,
13242             "\\markdownRendererFencedCodeAttributeContextEnd{}")
13243         end
13244         return buf
13245       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
    local writer = self.writer

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

    local function strip_enclosing_whitespaces(str)
      return str:gsub("^%s*(.-)%s*$", "%1")
    end

    local tilde_infostring = Cs(Cs((V("HtmlEntity")
                                  + parsers.anyescaped
                                  - parsers.newline)^0)
                               / strip_enclosing_whitespaces)

    local backtick_infostring
      = Cs( Cs((V("HtmlEntity")
          + ( -#(parsers.backslash * parsers.backtick)
            * parsers.anyescaped)
            - parsers.newline
            - parsers.backtick)^0)
      / strip_enclosing_whitespaces)

    local fenceindent

    local function has_trail(indent_table)
      return indent_table ~= nil and
        indent_table.trail ~= nil and
        next(indent_table.trail) ~= nil
    end

    local function has_indents(indent_table)
      return indent_table ~= nil and
        indent_table.indents ~= nil and
        next(indent_table.indents) ~= nil
```

```
13293       end
13294
13295       local function get_last_indent_name(indent_table)
13296         if has_indents(indent_table) then
13297           return indent_table.indents[#indent_table.indents].name
13298         end
13299       end
13300
13301       local count_fenced_start_indent =
13302         function(_, _, indent_table, trail)
13303           local last_indent_name = get_last_indent_name(indent_table)
13304           fenceindent = 0
13305           if last_indent_name ~= "li" then
13306             fenceindent = #trail
13307           end
13308           return true
13309         end
13310
13311       local fencehead = function(char, infostring)
13312         return Cmt( Cb("indent_info")
13313                   * parsers.check_trail, count_fenced_start_indent)
13314               * Cg(char^3, "fencelength")
13315               * parsers.optionalspace
13316               * infostring
13317               * (parsers.newline + parsers.eof)
13318       end
13319
13320       local fencetail = function(char)
13321         return parsers.check_trail_no_rem
13322               * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
13323               * parsers.optionalspace * (parsers.newline + parsers.eof)
13324               + parsers.eof
13325       end
13326
13327       local process_fenced_line =
13328         function(s, i, -- luacheck: ignore s i
13329                  indent_table, line_content, is_blank)
13330           local remainder = ""
13331           if has_trail(indent_table) then
13332             remainder = indent_table.trail.internal_remainder
13333           end
13334
13335           if is_blank
13336              and get_last_indent_name(indent_table) == "li" then
13337             remainder = ""
13338           end
13339
13340           local str = remainder .. line_content
13341           local index = 1
```

```
13342           local remaining = fenceindent
13343
13344           while true do
13345             local c = str:sub(index, index)
13346             if c == " " and remaining > 0 then
13347               remaining = remaining - 1
13348               index = index + 1
13349             elseif c == "\t" and remaining > 3 then
13350               remaining = remaining - 4
13351               index = index + 1
13352             else
13353               break
13354             end
13355           end
13356
13357           return true, str:sub(index)
13358         end
13359
13360       local fencedline = function(char)
13361         return Cmt( Cb("indent_info")
13362                   * C(parsers.line - fencetail(char))
13363                   * Cc(false), process_fenced_line)
13364       end
13365
13366       local blankfencedline
13367         = Cmt( Cb("indent_info")
13368              * C(parsers.blankline)
13369              * Cc(true), process_fenced_line)
13370
13371       local TildeFencedCode
13372         = fencehead(parsers.tilde, tilde_infostring)
13373         * Cs(( (parsers.check_minimal_blank_indent / "")
13374              * blankfencedline
13375              + ( parsers.check_minimal_indent / "")
13376                * fencedline(parsers.tilde))^0)
13377         * ( (parsers.check_minimal_indent / "")
13378           * fencetail(parsers.tilde) + parsers.succeed)
13379
13380       local BacktickFencedCode
13381              = fencehead(parsers.backtick, backtick_infostring)
13382              * Cs(( (parsers.check_minimal_blank_indent / "")
13383                   * blankfencedline
13384                   + (parsers.check_minimal_indent / "")
13385                   * fencedline(parsers.backtick))^0)
13386              * ( (parsers.check_minimal_indent / "")
13387                * fencetail(parsers.backtick) + parsers.succeed)
13388
13389       local infostring_with_attributes
13390                         = Ct(C((parsers.linechar
```

```
                              - ( parsers.optionalspace
                                * parsers.attributes))^0)
                            * parsers.optionalspace
                            * Ct(parsers.attributes))

    local FencedCode
      = ((TildeFencedCode + BacktickFencedCode)
      / function(infostring, code)
        local expanded_code = self.expandtabs(code)

        if allow_raw_blocks then
          local raw_attr = lpeg.match(parsers.raw_attribute,
                                      infostring)
          if raw_attr then
            return writer.rawBlock(expanded_code, raw_attr)
          end
        end

        local attr = nil
        if allow_attributes then
          local match = lpeg.match(infostring_with_attributes,
                                   infostring)
          if match then
            infostring, attr = table.unpack(match)
          end
        end
        return writer.fencedCode(expanded_code, infostring, attr)
      end)

    self.insert_pattern("Block after Verbatim",
                        FencedCode, "FencedCode")

    local fencestart
    if blank_before_code_fence then
      fencestart = parsers.fail
    else
      fencestart = fencehead(parsers.backtick, backtick_infostring)
                 + fencehead(parsers.tilde, tilde_infostring)
    end

    self.update_rule("EndlineExceptions", function(previous_pattern)
      if previous_pattern == nil then
        previous_pattern = parsers.EndlineExceptions
      end
      return previous_pattern + fencestart
    end)

    self.add_special_character("`")
    self.add_special_character("~")
```

```
    end
  }
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
                              = Ct(parsers.attributes)
                              + (
                                C( parsers.attribute_classname
                                 - parsers.colon)^1
                                / function (infostring)
                                  return {"." .. infostring}
                                end
                              )

      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 =
```

```
13518           function(s, i, -- luacheck: ignore s i
13519                    fenced_div_indent_table, fenced_div_level)
13520             fenced_div_level = tonumber(fenced_div_level)
13521             fenced_div_indent_table[fenced_div_level] = nil
13522             return true, tostring(fenced_div_level - 1)
13523           end
13524
13525         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
13526                       * Cb("fenced_div_level"), pop_indent_table)
13527                  , "fenced_div_level")
13528       end
13529
13530
13531       local non_fenced_div_block
13532         = parsers.check_minimal_indent * V("Block")
13533         - parsers.check_minimal_indent_and_trail * fenced_div_end
13534
13535       local non_fenced_div_paragraph
13536         = parsers.check_minimal_indent * V("Paragraph")
13537         - parsers.check_minimal_indent_and_trail * fenced_div_end
13538
13539       local blank = parsers.minimally_indented_blank
13540
13541       local block_separated = parsers.block_sep_group(blank)
13542                             * non_fenced_div_block
13543
13544       local loop_body_pair
13545         = parsers.create_loop_body_pair(block_separated,
13546                                         non_fenced_div_paragraph,
13547                                         parsers.block_sep_group(blank),
13548                                         parsers.par_sep_group(blank))
13549
13550       local content_loop  = ( non_fenced_div_block
13551                             * loop_body_pair.block^0
13552                             + non_fenced_div_paragraph
13553                             * block_separated
13554                             * loop_body_pair.block^0
13555                             + non_fenced_div_paragraph
13556                             * loop_body_pair.par^0)
13557                           * blank^0
13558
13559       local FencedDiv = fenced_div_begin
13560                       / writer.div_begin
13561                       * increment_div_level()
13562                       * parsers.skipblanklines
13563                       * Ct(content_loop)
13564                       * parsers.minimally_indented_blank^0
13565                       * parsers.check_minimal_indent_and_trail
13566                       * 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
```

```
13657
13658       local heading_text
13659         = heading_line
13660         * ( (V("Endline") / "\n")
13661           * (heading_line - parsers.heading_level))^0
13662         * parsers.newline^-1
13663
13664       local SetextHeading
13665         = parsers.freeze_trail * parsers.check_trail_no_rem
13666         * #(heading_text
13667            * (parsers.attributes
13668              * parsers.optionalspace
13669              * parsers.newline)^-1
13670            * parsers.check_minimal_indent
13671            * parsers.check_trail
13672            * parsers.heading_level)
13673         * Cs(heading_text) / strip_trailing_spaces
13674         / parsers.parse_heading_text
13675         * Cg(Ct((parsers.attributes
13676               * parsers.optionalspace
13677               * parsers.newline)^-1), "attributes")
13678         * parsers.check_minimal_indent_and_trail * parsers.heading_level
13679         * Cb("attributes")
13680         * parsers.newline
13681         * parsers.unfreeze_trail
13682         / writer.heading
13683
13684       local Heading = AtxHeading + SetextHeading
13685       self.update_rule("Heading", Heading)
13686     end
13687   }
13688 end
```

#### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```
13689 M.extensions.inline_code_attributes = function()
13690   return {
13691     name = "built-in inline_code_attributes syntax extension",
13692     extend_writer = function()
13693     end, extend_reader = function(self)
13694       local writer = self.writer
13695
13696       local CodeWithAttributes = parsers.inticks
13697                                * Ct(parsers.attributes)
13698                                / writer.code
13699
13700       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

      local LineBlock
        = Ct((Cs(( (parsers.pipe * parsers.space) / ""
                 * ((parsers.space)/entities.char_entity("nbsp"))^0
                 * parsers.linechar^0 * (parsers.newline/""))
                 * (-parsers.pipe
                   * (parsers.space^1/" ")
                   * parsers.linechar^1
                   * (parsers.newline/"")
                   )^0)
             / self.parser_functions.parse_inlines)^1)
        / writer.lineblock

      self.insert_pattern("Block after Blockquote",
                          LineBlock, "LineBlock")
    end
  }
end
```

#### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```
M.extensions.mark = function()
  return {
    name = "built-in mark syntax extension",
    extend_writer = function(self)
```

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

```
      function self.mark(s)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererMark{", s, "}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local doubleequals = P("==")

      local Mark
        = parsers.between(V("Inline"), doubleequals, doubleequals)
        / function (inlines) return writer.mark(inlines) end

      self.add_special_character("=")
      self.insert_pattern("Inline before LinkAndEmph",
                          Mark, "Mark")
    end
  }
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.

```
      function self.note(s)
        if self.flatten_inlines then return "" end
        return {"\\markdownRendererNote{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local rawnotes = parsers.rawnotes

      if inline_notes then
        local InlineNote
          = parsers.circumflex
          * ( parsers.link_label
            / self.parser_functions.parse_inlines_no_inline_note)
          / writer.note

        self.insert_pattern("Inline after LinkAndEmph",
                            InlineNote, "InlineNote")
      end
      if notes then
        local function strip_first_char(s)
          return s:sub(2)
        end

        local RawNoteRef
```

```
                    = #(parsers.lbracket * parsers.circumflex)
                    * parsers.link_label / strip_first_char

      -- like indirect_link
      local function lookup_note(ref)
        return writer.defer_call(function()
          local found = rawnotes[self.normalize_tag(ref)]
          if found then
            return writer.note(
              self.parser_functions.parse_blocks_nested(found))
          else
            local text = string.format(
              'Undefined note reference "%s"', ref)
            local more = string.format(
              "Look for the text `[^%s]`.", ref)
            return {writer.warning(text, more), "[",
              self.parser_functions.parse_inlines("^" .. ref), "]"}
          end
        end)
      end

      local function register_note(ref,rawnote)
        local normalized_tag = self.normalize_tag(ref)
        if rawnotes[normalized_tag] == nil then
          rawnotes[normalized_tag] = rawnote
          return ""
        else
          local text
            = string.format('Multiply defined note reference "%s"',
                            ref)
          local more
            = string.format("Look for the text `[^%s]: ...`.", ref)
          return writer.warning(text, more)
        end
      end

      local NoteRef = RawNoteRef / lookup_note

      local optionally_indented_line
        = parsers.check_optional_indent_and_any_trail * parsers.line

      local blank
        = parsers.check_optional_blank_indent_and_any_trail
        * parsers.optionalspace * parsers.newline

      local chunk
        = Cs(parsers.line
        * (optionally_indented_line - blank)^0)

```

```
        local indented_blocks = function(bl)
          return Cs( bl
                * ( blank^1 * (parsers.check_optional_indent / "")
                  * parsers.check_code_trail
                  * -parsers.blankline * bl)^0)
        end

        local NoteBlock
                    = parsers.check_trail_no_rem
                    * RawNoteRef * parsers.colon
                    * parsers.spnlc * indented_blocks(chunk)
                    / register_note

        self.update_rule("Reference", function(previous_pattern)
          if previous_pattern == nil then
            previous_pattern = parsers.Reference
          end
          return NoteBlock + previous_pattern
        end)

        self.insert_pattern("Inline before LinkAndEmph",
                            NoteRef, "NoteRef")
      end

      self.add_special_character("^")
    end
  }
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.

```
M.extensions.pipe_tables = function(table_captions, table_attributes)

  local function make_pipe_table_rectangular(rows)
    local num_columns = #rows[2]
    local rectangular_rows = {}
    for i = 1, #rows do
      local row = rows[i]
      local rectangular_row = {}
      for j = 1, num_columns do
        rectangular_row[j] = row[j] or ""
      end
      table.insert(rectangular_rows, rectangular_row)
    end
```

```
13957     return rectangular_rows
13958   end
13959
13960   local function pipe_table_row(allow_empty_first_column
13961                                 , nonempty_column
13962                                 , column_separator
13963                                 , column)
13964     local row_beginning
13965     if allow_empty_first_column then
13966       row_beginning = -- empty first column
13967                       #(parsers.spacechar^4
13968                         * column_separator)
13969                     * parsers.optionalspace
13970                     * column
13971                     * parsers.optionalspace
13972                     -- non-empty first column
13973                     + parsers.nonindentspace
13974                     * nonempty_column^-1
13975                     * parsers.optionalspace
13976     else
13977       row_beginning = parsers.nonindentspace
13978                     * nonempty_column^-1
13979                     * parsers.optionalspace
13980     end
13981
13982     return Ct(row_beginning
13983               * (-- single column with no leading pipes
13984                  #(column_separator
13985                    * parsers.optionalspace
13986                    * parsers.newline)
13987                  * column_separator
13988                  * parsers.optionalspace
13989                  -- single column with leading pipes or
13990                  -- more than a single column
13991                  + (column_separator
13992                    * parsers.optionalspace
13993                    * column
13994                    * parsers.optionalspace)^1
13995                  * (column_separator
13996                    * parsers.optionalspace)^-1))
13997   end
13998
13999   return {
14000     name = "built-in pipe_tables syntax extension",
14001     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.

```
14002       function self.table(rows, caption, attributes)
14003         if not self.is_writing then return "" end
```

```
14004         local buffer = {}
14005         if attributes ~= nil then
14006           table.insert(buffer,
14007                        "\\markdownRendererTableAttributeContextBegin\n")
14008           table.insert(buffer, self.attributes(attributes))
14009         end
14010         table.insert(buffer,
14011                      {"\\markdownRendererTable{",
14012                       caption or "", "}{", #rows - 1, "}{",
14013                       #rows[1], "}"})
14014         local temp = rows[2] -- put alignments on the first row
14015         rows[2] = rows[1]
14016         rows[1] = temp
14017         for i, row in ipairs(rows) do
14018           table.insert(buffer, "{")
14019           for _, column in ipairs(row) do
14020             if i > 1 then -- do not use braces for alignments
14021               table.insert(buffer, "{")
14022             end
14023             table.insert(buffer, column)
14024             if i > 1 then
14025               table.insert(buffer, "}")
14026             end
14027           end
14028           table.insert(buffer, "}")
14029         end
14030         if attributes ~= nil then
14031           table.insert(buffer,
14032                        "\\markdownRendererTableAttributeContextEnd{}")
14033         end
14034         return buffer
14035       end
14036     end, extend_reader = function(self)
14037       local parsers = self.parsers
14038       local writer = self.writer
14039
14040       local table_hline_separator = parsers.pipe + parsers.plus
14041
14042       local table_hline_column = (parsers.dash
14043                                   - #(parsers.dash
14044                                      * (parsers.spacechar
14045                                        + table_hline_separator
14046                                        + parsers.newline)))^1
14047                                 * (parsers.colon * Cc("r")
14048                                   + parsers.dash * Cc("d"))
14049                                 + parsers.colon
14050                                 * (parsers.dash
14051                                   - #(parsers.dash
14052                                      * (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

    local table_row
      = pipe_table_row(true
                      , (C((parsers.linechar - parsers.pipe)^1)
                        / strip_trailing_spaces
                        / self.parser_functions.parse_inlines)
                      , parsers.pipe
                      , (C((parsers.linechar - parsers.pipe)^0)
                        / strip_trailing_spaces
                        / self.parser_functions.parse_inlines))

    local table_caption
    if table_captions then
      table_caption = #table_caption_beginning
                    * table_caption_beginning
      if table_attributes then
        table_caption = table_caption
                      * (C(((( parsers.linechar
                             - (parsers.attributes
                               * parsers.optionalspace
                               * parsers.newline
                               * -#( parsers.optionalspace
                                   * parsers.linechar)))
                            + ( parsers.newline
                              * #( parsers.optionalspace
                                 * parsers.linechar)
                              * C(parsers.optionalspace)
                              / writer.space))
                           * (parsers.linechar
```

```
                                - parsers.lbrace)^0)^1)
                              / strip_trailing_spaces
                              / self.parser_functions.parse_inlines)
                          * (parsers.newline
                            + ( Ct(parsers.attributes)
                              * parsers.optionalspace
                              * parsers.newline))
        else
          table_caption = table_caption
                          * C(( parsers.linechar
                              + ( parsers.newline
                                * #( parsers.optionalspace
                                   * parsers.linechar)
                                * C(parsers.optionalspace)
                                / writer.space))^1)
                          / self.parser_functions.parse_inlines
                          * parsers.newline
        end
      else
        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
  }
end
```

#### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

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

```
      function self.strike_through(s)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererStrikeThrough{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local StrikeThrough = (
        parsers.between(parsers.Inline, parsers.doubletildes,
                        parsers.doubletildes)
      ) / writer.strike_through

      self.insert_pattern("Inline after LinkAndEmph",
                          StrikeThrough, "StrikeThrough")

```

```
      self.add_special_character("~")
    end
  }
end
```

#### 3.1.7.17 Subscripts

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

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

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

```
      function self.subscript(s, parsed)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererSubscript{",parsed,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Subscript = (
        parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
      ) / function(s)
            return writer.subscript(
              s, self.parser_functions.parse_inlines_identity(s[1])
            )
          end

      self.insert_pattern("Inline after LinkAndEmph",
                          Subscript, "Subscript")

      self.add_special_character("~")
    end
  }
end
```

#### 3.1.7.18 Superscripts

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

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

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

```
14219       function self.superscript(s, parsed)
14220         if self.flatten_inlines then return s end
14221         return {"\\markdownRendererSuperscript{",parsed,"}"}
14222       end
14223     end, extend_reader = function(self)
14224       local parsers = self.parsers
14225       local writer = self.writer
14226
14227       local Superscript = (
14228         parsers.between(parsers.Str, parsers.circumflex,
14229                         parsers.circumflex)
14230       ) / function(s)
14231             return writer.superscript(
14232               s, self.parser_functions.parse_inlines_identity(s[1])
14233             )
14234           end
14235
14236       self.insert_pattern("Inline after LinkAndEmph",
14237                           Superscript, "Superscript")
14238
14239       self.add_special_character("^")
14240     end
14241   }
14242 end
```

#### 3.1.7.19 TEX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
14243 M.extensions.tex_math = function(tex_math_dollars,
14244                                   tex_math_single_backslash,
14245                                   tex_math_double_backslash)
14246   return {
14247     name = "built-in tex_math syntax extension",
14248     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.

```
14249       function self.display_math(s, parsed)
14250         if self.flatten_inlines then return s end
14251         return {"\\markdownRendererDisplayMath{",parsed,"}"}
14252       end
```

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

```
14253       function self.inline_math(s, parsed)
14254         if self.flatten_inlines then return s end
14255         return {"\\markdownRendererInlineMath{",parsed,"}"}
14256       end
```

```
14257     end, extend_reader = function(self)
14258       local parsers = self.parsers
14259       local writer = self.writer
14260
14261       local function between(p, starter, ender)
14262         return (starter * Cs(p * (p - ender)^0) * ender)
14263       end
14264
14265       local function strip_preceding_whitespaces(str)
14266         return str:gsub("^%s*(.-)$", "%1")
14267       end
14268
14269       local allowed_before_closing
14270         = B( parsers.backslash * parsers.any
14271            + parsers.any * (parsers.any - parsers.backslash))
14272
14273       local allowed_before_closing_no_space
14274         = B( parsers.backslash * parsers.any
14275            + parsers.any * (parsers.nonspacechar - parsers.backslash))
14276
```

The following patterns implement the Pandoc dollar math syntax extension.

```
14277       local dollar_math_content
14278         = (parsers.newline * (parsers.check_optional_indent / "")
14279         + parsers.backslash^-1
14280         * parsers.linechar)
14281         - parsers.blankline^2
14282         - parsers.dollar
14283
14284       local inline_math_opening_dollars = parsers.dollar
14285                                         * #(parsers.nonspacechar)
14286
14287       local inline_math_closing_dollars
14288         = allowed_before_closing_no_space
14289         * parsers.dollar
14290         * -#(parsers.digit)
14291
14292       local inline_math_dollars = between(Cs( dollar_math_content),
14293                                           inline_math_opening_dollars,
14294                                           inline_math_closing_dollars)
14295
14296       local display_math_opening_dollars  = parsers.dollar
14297                                           * parsers.dollar
14298
14299       local display_math_closing_dollars  = parsers.dollar
14300                                           * parsers.dollar
14301
14302       local display_math_dollars = between(Cs( dollar_math_content),
14303                                            display_math_opening_dollars,
14304                                            display_math_closing_dollars)
```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```
local backslash_math_content
  = (parsers.newline * (parsers.check_optional_indent / "")
  + parsers.linechar)
  - parsers.blankline^2
```

The following patterns implement the Pandoc double backslash math syntax extension.

```
local inline_math_opening_double  = parsers.backslash
                                  * parsers.backslash
                                  * parsers.lparent

local inline_math_closing_double  = allowed_before_closing
                                  * parsers.spacechar^0
                                  * parsers.backslash
                                  * parsers.backslash
                                  * parsers.rparent

local inline_math_double  = between(Cs( backslash_math_content),
                                    inline_math_opening_double,
                                    inline_math_closing_double)
                          / strip_preceding_whitespaces

local display_math_opening_double = parsers.backslash
                                  * parsers.backslash
                                  * parsers.lbracket

local display_math_closing_double = allowed_before_closing
                                  * parsers.spacechar^0
                                  * parsers.backslash
                                  * parsers.backslash
                                  * parsers.rbracket

local display_math_double = between(Cs( backslash_math_content),
                                    display_math_opening_double,
                                    display_math_closing_double)
                          / strip_preceding_whitespaces
```

The following patterns implement the Pandoc single backslash math syntax extension.

```
local inline_math_opening_single  = parsers.backslash
                                  * parsers.lparent

local inline_math_closing_single  = allowed_before_closing
                                  * parsers.spacechar^0
                                  * parsers.backslash
                                  * parsers.rparent

local inline_math_single  = between(Cs( backslash_math_content),
                                    inline_math_opening_single,
                                    inline_math_closing_single)
```

```
                              / strip_preceding_whitespaces

    local display_math_opening_single = parsers.backslash
                                      * parsers.lbracket

    local display_math_closing_single = allowed_before_closing
                                      * parsers.spacechar^0
                                      * parsers.backslash
                                      * parsers.rbracket

    local display_math_single = between(Cs( backslash_math_content),
                                        display_math_opening_single,
                                        display_math_closing_single)
                              / strip_preceding_whitespaces

    local display_math = parsers.fail

    local inline_math = parsers.fail

    if tex_math_dollars then
      display_math = display_math + display_math_dollars
      inline_math = inline_math + inline_math_dollars
    end

    if tex_math_double_backslash then
      display_math = display_math + display_math_double
      inline_math = inline_math + inline_math_double
    end

    if tex_math_single_backslash then
      display_math = display_math + display_math_single
      inline_math = inline_math + inline_math_single
    end

    local TexMath = display_math
                  / function(s)
                      return writer.display_math(
                        s, self.parser_functions.parse_inlines_math(s)
                      )
                    end
                  + inline_math
                  / function(s)
                      return writer.inline_math(
                        s, self.parser_functions.parse_inlines_math(s)
                      )
                    end

    self.insert_pattern("Inline after LinkAndEmph",
                        TexMath, "TexMath")
```

```
14398
14399       if tex_math_dollars then
14400         self.add_special_character("$")
14401       end
14402
14403       if tex_math_single_backslash or tex_math_double_backslash then
14404         self.add_special_character("\\")
14405         self.add_special_character("[")
14406         self.add_special_character("]")
14407         self.add_special_character(")")
14408         self.add_special_character("(")
14409       end
14410     end
14411   }
14412 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.

```
14413 M.extensions.jekyll_data = function(expect_jekyll_data,
14414                                     ensure_jekyll_data)
14415   return {
14416     name = "built-in jekyll_data syntax extension",
14417     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.

```
14418       function self.jekyllData(d, t, p)
14419         if not self.is_writing then return "" end
14420
14421         local buf = {}
14422
14423         local keys = {}
14424         for k, _ in pairs(d) do
14425           table.insert(keys, k)
14426         end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
14427         table.sort(keys, function(first, second)
```

```
14428           if type(first) ~= type(second) then
14429             return type(first) < type(second)
14430           else
14431             return first < second
14432           end
14433         end)
14434
14435         if not p then
14436           table.insert(buf, "\\markdownRendererJekyllDataBegin")
14437         end
14438
14439         local is_sequence = false
14440         if #d > 0 and #d == #keys then
14441           for i=1, #d do
14442             if d[i] == nil then
14443               goto not_a_sequence
14444             end
14445           end
14446           is_sequence = true
14447         end
14448         ::not_a_sequence::
14449
14450         if is_sequence then
14451           table.insert(buf,
14452             "\\markdownRendererJekyllDataSequenceBegin{")
14453           table.insert(buf, self.identifier(p or "null"))
14454           table.insert(buf, "}{")
14455           table.insert(buf, #keys)
14456           table.insert(buf, "}")
14457         else
14458           table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
14459           table.insert(buf, self.identifier(p or "null"))
14460           table.insert(buf, "}{")
14461           table.insert(buf, #keys)
14462           table.insert(buf, "}")
14463         end
14464
14465         for _, k in ipairs(keys) do
14466           local v = d[k]
14467           local typ = type(v)
14468           k = tostring(k or "null")
14469           if typ == "table" and next(v) ~= nil then
14470             table.insert(
14471               buf,
14472               self.jekyllData(v, t, k)
14473             )
14474           else
14475             k = self.identifier(k)
14476             v = tostring(v)
```

```
14477             if typ == "boolean" then
14478               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
14479               table.insert(buf, k)
14480               table.insert(buf, "}{")
14481               table.insert(buf, v)
14482               table.insert(buf, "}")
14483             elseif typ == "number" then
14484               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
14485               table.insert(buf, k)
14486               table.insert(buf, "}{")
14487               table.insert(buf, v)
14488               table.insert(buf, "}")
14489             elseif typ == "string" then
14490               table.insert(buf,
14491                 "\\markdownRendererJekyllDataProgrammaticString{")
14492               table.insert(buf, k)
14493               table.insert(buf, "}{")
14494               table.insert(buf, self.identifier(v))
14495               table.insert(buf, "}")
14496               table.insert(buf,
14497                 "\\markdownRendererJekyllDataTypographicString{")
14498               table.insert(buf, k)
14499               table.insert(buf, "}{")
14500               table.insert(buf, t(v))
14501               table.insert(buf, "}")
14502             elseif typ == "table" then
14503               table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
14504               table.insert(buf, k)
14505               table.insert(buf, "}")
14506             else
14507               local error = self.error(format(
14508                 "Unexpected type %s for value of "
14509                 .. "YAML key %s.", typ, k))
14510               table.insert(buf, error)
14511             end
14512           end
14513         end
14514
14515         if is_sequence then
14516           table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
14517         else
14518           table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
14519         end
14520
14521         if not p then
14522           table.insert(buf, "\\markdownRendererJekyllDataEnd")
14523         end
14524
14525         return buf
```

```
14526       end
14527     end, extend_reader = function(self)
14528       local parsers = self.parsers
14529       local writer = self.writer
14530
14531       local JekyllData
14532         = Cmt( C((parsers.line - P("---") - P("..."))^0)
14533              , function(s, i, text) -- luacheck: ignore s i
14534                  local data
14535                  local ran_ok, _ = pcall(function()
14536                    local tinyyaml = require("tinyyaml")
14537                    data = tinyyaml.parse(text, {timestamps=false})
14538                  end)
14539                  if ran_ok and data ~= nil then
14540                    return true, writer.jekyllData(data, function(s)
14541                      return self.parser_functions.parse_blocks_nested(s)
14542                    end, nil)
14543                  else
14544                    return false
14545                  end
14546                end
14547              )
14548
14549       local UnexpectedJekyllData
14550         = P("---")
14551         * parsers.blankline / 0
14552         -- if followed by blank, it's thematic break
14553         * #(-parsers.blankline)
14554         * JekyllData
14555         * (P("---") + P("..."))
14556
14557       local ExpectedJekyllData
14558         = ( P("---")
14559           * parsers.blankline / 0
14560           -- if followed by blank, it's thematic break
14561           * #(-parsers.blankline)
14562           )^-1
14563         * JekyllData
14564         * (P("---") + P("..."))^-1
14565
14566       if ensure_jekyll_data then
14567         ExpectedJekyllData = ExpectedJekyllData
14568                            * parsers.eof
14569       else
14570         ExpectedJekyllData = ( ExpectedJekyllData
14571                              * (V("Blank")^0 / writer.interblocksep)
14572                              )^-1
14573       end
14574
```

```
      self.insert_pattern("Block before Blockquote",
                          UnexpectedJekyllData, "UnexpectedJekyllData")
      if expect_jekyll_data then
        self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
      end
    end
  }
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 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` and `experimentalOptions` tables.

```
  options = options or {}
  setmetatable(options, { __index = function (_, key)
    return defaultOptions[key] end })
  if options.experimental then
    setmetatable(options, { __index = function (_, key)
      return experimentalOptions[key] end })
  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, include_flat_output)
    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 raw_output, flat_output
    if options.eagerCache or options.finalizeCache then
      local salt = util.salt(options)
      local name, result = util.cache(options.cacheDir, input, salt,
                                      convert, ".md.tex")
      raw_output = [[\input{]] .. name .. [[}\relax]]
      flat_output = function()
        if result == nil then
          local input_file = assert(io.open(name, "r"),
            [[Could not open file "]] .. name .. [[" for reading]])
          result = assert(input_file:read("*a"))
          assert(input_file:close())
        end
        return result
      end
```

Otherwise, return the result of the conversion directly.

```
    else
      raw_output = convert(input)
      flat_output = function()
        return raw_output
      end
    end
```

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

```
    if options.finalizeCache then
      local file, mode
      if options.frozenCacheCounter > 0 then
        mode = "a"
      else
        mode = "w"
      end
      file = assert(io.open(options.frozenCacheFileName, mode),
        [[Could not open file "]] .. options.frozenCacheFileName
        .. [[" for writing]])
      assert(file:write(
        [[\expandafter\global\expandafter\def\csname ]]
        .. [[markdownFrozenCache]] .. options.frozenCacheCounter
        .. [[\endcsname{]] .. raw_output .. [[}]] .. "\n"))
      assert(file:close())
    end
```

Besides the canonical output of the conversion, which may contain cached files behind `\input`, also return a function that always produces a flat output regardless of caching as the second return value.

```
    if include_flat_output then
```

```
      return raw_output, flat_output
    else
      return raw_output
    end
  end
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.

```
function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```
  options = options or {}
  setmetatable(options, { __index = function (_, key)
    return defaultOptions[key] end })
  if options.experimental then
    setmetatable(options, { __index = function (_, key)
      return experimentalOptions[key] end })
  end
```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
  if options.singletonCache and singletonCache.convert then
    for k, v in pairs(defaultOptions) do
      if type(v) == "table" then
        for i = 1, math.max(#singletonCache.options[k], #options[k]) do
          if singletonCache.options[k][i] ~= options[k][i] then
            goto miss
          end
        end
```

The `cacheDir` option is disregarded.

```
      elseif k ~= "cacheDir"
         and singletonCache.options[k] ~= options[k] then
        goto miss
      end
    end
    return singletonCache.convert
  end
  ::miss::
```

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

```
  local extensions = {}

  if #options.acronyms > 0 then
    local acronyms_extension = M.extensions.acronyms(options.acronyms)
    table.insert(extensions, acronyms_extension)
  end

  if options.bracketedSpans then
    local bracketed_spans_extension = M.extensions.bracketed_spans()
```

```
14681     table.insert(extensions, bracketed_spans_extension)
14682   end
14683
14684   if options.contentBlocks then
14685     local content_blocks_extension = M.extensions.content_blocks(
14686       options.contentBlocksLanguageMap)
14687     table.insert(extensions, content_blocks_extension)
14688   end
14689
14690   if options.definitionLists then
14691     local definition_lists_extension = M.extensions.definition_lists(
14692       options.tightLists)
14693     table.insert(extensions, definition_lists_extension)
14694   end
14695
14696   if options.fencedCode then
14697     local fenced_code_extension = M.extensions.fenced_code(
14698       options.blankBeforeCodeFence,
14699       options.fencedCodeAttributes,
14700       options.rawAttribute)
14701     table.insert(extensions, fenced_code_extension)
14702   end
14703
14704   if options.fencedDivs then
14705     local fenced_div_extension = M.extensions.fenced_divs(
14706       options.blankBeforeDivFence)
14707     table.insert(extensions, fenced_div_extension)
14708   end
14709
14710   if options.headerAttributes then
14711     local header_attributes_extension = M.extensions.header_attributes()
14712     table.insert(extensions, header_attributes_extension)
14713   end
14714
14715   if options.inlineCodeAttributes then
14716     local inline_code_attributes_extension =
14717       M.extensions.inline_code_attributes()
14718     table.insert(extensions, inline_code_attributes_extension)
14719   end
14720
14721   if options.jekyllData then
14722     local jekyll_data_extension = M.extensions.jekyll_data(
14723       options.expectJekyllData, options.ensureJekyllData)
14724     table.insert(extensions, jekyll_data_extension)
14725   end
14726
14727   if options.linkAttributes then
14728     local link_attributes_extension =
14729       M.extensions.link_attributes()
```

```
14730     table.insert(extensions, link_attributes_extension)
14731   end
14732
14733   if options.lineBlocks then
14734     local line_block_extension = M.extensions.line_blocks()
14735     table.insert(extensions, line_block_extension)
14736   end
14737
14738   if options.mark then
14739     local mark_extension = M.extensions.mark()
14740     table.insert(extensions, mark_extension)
14741   end
14742
14743   if options.pipeTables then
14744     local pipe_tables_extension = M.extensions.pipe_tables(
14745       options.tableCaptions, options.tableAttributes)
14746     table.insert(extensions, pipe_tables_extension)
14747   end
14748
14749   if options.rawAttribute then
14750     local raw_inline_extension = M.extensions.raw_inline()
14751     table.insert(extensions, raw_inline_extension)
14752   end
14753
14754   if options.strikeThrough then
14755     local strike_through_extension = M.extensions.strike_through()
14756     table.insert(extensions, strike_through_extension)
14757   end
14758
14759   if options.subscripts then
14760     local subscript_extension = M.extensions.subscripts()
14761     table.insert(extensions, subscript_extension)
14762   end
14763
14764   if options.superscripts then
14765     local superscript_extension = M.extensions.superscripts()
14766     table.insert(extensions, superscript_extension)
14767   end
14768
14769   if options.texMathDollars or
14770      options.texMathSingleBackslash or
14771      options.texMathDoubleBackslash then
14772     local tex_math_extension = M.extensions.tex_math(
14773       options.texMathDollars,
14774       options.texMathSingleBackslash,
14775       options.texMathDoubleBackslash)
14776     table.insert(extensions, tex_math_extension)
14777   end
14778
```

```
14779   if options.notes or options.inlineNotes then
14780     local notes_extension = M.extensions.notes(
14781       options.notes, options.inlineNotes)
14782     table.insert(extensions, notes_extension)
14783   end
14784
14785   if options.citations then
14786     local citations_extension
14787       = M.extensions.citations(options.citationNbsps)
14788     table.insert(extensions, citations_extension)
14789   end
14790
14791   if options.fancyLists then
14792     local fancy_lists_extension = M.extensions.fancy_lists()
14793     table.insert(extensions, fancy_lists_extension)
14794   end
```

Apply user-defined syntax extensions based on `options.extensions`.

```
14795   for _, user_extension_filename in ipairs(options.extensions) do
14796     local user_extension = (function(filename)
```

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

```
14797       local pathname = assert(util.find_file(filename),
14798         [[Could not locate user-defined syntax extension "]]
14799         .. filename)
14800       local input_file = assert(io.open(pathname, "r"),
14801         [[Could not open user-defined syntax extension "]]
14802         .. pathname .. [[" for reading]])
14803       local input = assert(input_file:read("*a"))
14804       assert(input_file:close())
14805       local user_extension, err = load([[
14806         local sandbox = {}
14807         setmetatable(sandbox, {__index = _G})
14808         _ENV = sandbox
14809       ]] .. input)()
14810       assert(user_extension,
14811         [[Failed to compile user-defined syntax extension "]]
14812         .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
14813       assert(user_extension.api_version ~= nil,
14814         [[User-defined syntax extension "]] .. pathname
14815         .. [[" does not specify mandatory field "api_version"]])
14816       assert(type(user_extension.api_version) == "number",
14817         [[User-defined syntax extension "]] .. pathname
14818         .. [[" specifies field "api_version" of type "]]
14819         .. type(user_extension.api_version)
14820         .. [[" but "number" was expected]])
14821       assert(user_extension.api_version > 0
14822          and user_extension.api_version
14823           <= metadata.user_extension_api_version,
```

```
14824         [[User-defined syntax extension "]] .. pathname
14825         .. [[" uses syntax extension API version "]]
14826         .. user_extension.api_version .. [[ but markdown.lua ]]
14827         .. metadata.version .. [[ uses API version ]]
14828         .. metadata.user_extension_api_version
14829         .. [[, which is incompatible]])
14830
14831       assert(user_extension.grammar_version ~= nil,
14832         [[User-defined syntax extension "]] .. pathname
14833         .. [[" does not specify mandatory field "grammar_version"]])
14834       assert(type(user_extension.grammar_version) == "number",
14835         [[User-defined syntax extension "]] .. pathname
14836         .. [[" specifies field "grammar_version" of type "]]
14837         .. type(user_extension.grammar_version)
14838         .. [[" but "number" was expected]])
14839       assert(user_extension.grammar_version == metadata.grammar_version,
14840         [[User-defined syntax extension "]] .. pathname
14841         .. [[" uses grammar version "]]
14842         .. user_extension.grammar_version
14843         .. [[ but markdown.lua ]] .. metadata.version
14844         .. [[ uses grammar version ]] .. metadata.grammar_version
14845         .. [[, which is incompatible]])
14846
14847       assert(user_extension.finalize_grammar ~= nil,
14848         [[User-defined syntax extension "]] .. pathname
14849         .. [[" does not specify mandatory "finalize_grammar" field]])
14850       assert(type(user_extension.finalize_grammar) == "function",
14851         [[User-defined syntax extension "]] .. pathname
14852         .. [[" specifies field "finalize_grammar" of type "]]
14853         .. type(user_extension.finalize_grammar)
14854         .. [[" 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.)

```
14855       local extension = {
14856         name = [[user-defined "]] .. pathname .. [[" syntax extension]],
14857         extend_reader = user_extension.finalize_grammar,
14858         extend_writer = function() end,
14859       }
14860       return extension
14861     end)(user_extension_filename)
14862     table.insert(extensions, user_extension)
14863   end
```

Produce a conversion function from markdown to plain TEX.

```
14864   local writer = M.writer.new(options)
14865   local reader = M.reader.new(writer, options)
14866   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 raw_output, flat_output = convert(input, true)
local output
if flat_output == nil then
  if options.eagerCache then
    warn("markdown.lua has not produced flat output, so I am using " ..
         "backwards-compatible raw output instead. This may cause " ..
         'the conversion result to be hidden behind "\\input".')
  end
  output = raw_output
else
  output = flat_output()
end

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_protected:Npn
      \markdownInfo #1
      {
        \msg_info:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownWarning
  {
    \cs_new_protected:Npn
      \markdownWarning #1
      {
        \msg_warning:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownError
  {
    \cs_new_protected: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_protected: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_protected: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 }
          {
            \msg_warning:nnnn
              { markdown }
              { pin-theme-version }
              { witiko/diagrams }
              { v2 }
            \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~and~v2 }
      }
  }
```

```
\cs_generate_variant:Nn
  \msg_error:nnnnn
  { nnnen }
\msg_new:nnn
  { markdown }
  { pin-theme-version }
  {
    Write~"#1@#2"~to~pin~version~"#2"~of~the~theme~"#1".
    This~will~keep~your~documents~from~suddenly~breaking
    when~we~have~released~future~versions~of~the~theme
    with~backwards-incompatible~syntax~and~behavior.
  }
\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, the desired format, and the command to produce the diagram.

```
    \tl_new:N
      \l_@@_diagrams_caption_tl
    \tl_new:N
      \l_@@_diagrams_format_tl
    \tl_set:Nn
      \l_@@_diagrams_format_tl
      { pdf }
    \tl_new:N
      \l_@@_diagrams_command_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_case:nnF
                  { ##1 }
                  {
                    { caption }
                      {
                        \tl_set:Nn
                          \l_@@_diagrams_caption_tl
                          { ##2 }
                      }
                    { format }
                      {
                        \tl_set:Nn
                          \l_@@_diagrams_format_tl
                          { ##2 }
                      }
                    { command }
                      {
                        \tl_set:Nn
                          \l_@@_diagrams_command_tl
                          { ##2 }
                      }
                  }
                  {
                    \@@_diagrams_previous_key_value:nn
                      { ##1 }
                      { ##2 }
                  }
              },
            },
          }
      },
      fencedCodeAttributeContextEnd = {
        \markdownRendererImageAttributeContextEnd
        \group_end:
      },
    },
  }
\cs_new:Nn
  \@@_diagrams_render_diagram:nnnnn
```

```
      {
        \@@_if_option:nF
          { frozenCache }
          {
```

If possible, determine the version of the command used to produce the diagram and extract the major version number to `\l_markdown_diagrams_command_version_tl`. Otherwise, clear the variable.

```
            \tl_clear:N
              \l_markdown_diagrams_command_version_tl
            \tl_if_empty:nF
              { #3 }
              {
                \sys_get_shell:xnN
                  { #3 }
                  { }
                  \l_tmpa_tl
                \regex_extract_once:nVNT
                  { ^ \s* \d+ }
                  \l_tmpa_tl
                  \l_tmpa_seq
                  {
                    \tl_set:Nx
                      \l_markdown_diagrams_command_version_tl
                      {
                        \seq_item:Nn
                          \l_tmpa_seq
                          { 1 }
                      }
                  }
              }
            \tl_set:Nn
              \l_tmpa_tl
              {
                if~!~test~-e~#2.source~
                ||~!~diff~#1~#2.source;
                then~
              }
            \tl_put_right:Nx
              \l_tmpa_tl
              { (#4); }
            \tl_put_right:Nn
              \l_tmpa_tl
              {
                  cp~#1~#2.source;
                fi
              }
            \sys_shell_now:V
              \l_tmpa_tl
          }
```

```
      \exp_args:NNnV
        \exp_last_unbraced:No
        \markdownRendererImage
          {
            { #5 }
            { #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.

The token list `\l_markdown_diagrams_command_version_tl` should be used to store and pass around the version of the command used to produce the diagram, if necessary.

```
\prop_new:N
  \g_markdown_diagrams_infostrings_prop
\tl_new:N
  \l_markdown_diagrams_command_version_tl
```

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
      {
        \tl_set:Nn
          \l_tmpb_tl
          { dot~-T }
        \tl_put_right:NV
          \l_tmpb_tl
          \l_@@_diagrams_format_tl
        \tl_put_right:Nn
          \l_tmpb_tl
          { ~-o~#1. }
        \tl_put_right:NV
          \l_tmpb_tl
          \l_@@_diagrams_format_tl
        \tl_put_right:Nn
          \l_tmpb_tl
          { ~#1 }
```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```
        \str_if_eq:VnT
          \l_@@_diagrams_format_tl
          { svg }
          {
            \tl_put_right:Nn
              \l_tmpb_tl
              { ;~inkscape~#1. }
            \tl_put_right:NV
              \l_tmpb_tl
              \l_@@_diagrams_format_tl
            \tl_put_right:Nn
              \l_tmpb_tl
              { ~--export-area-drawing~--export-dpi=300~-o~#1.pdf }
          }
        \@@_diagrams_render_diagram:nnnVn
          { #1 }
          { #1.pdf }
          { }
          \l_tmpb_tl
          { Graphviz~image }
      }
    \cs_generate_variant:Nn
      \@@_diagrams_render_diagram:nnnnn
      { nnnVn }
    \@@_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`. The exact command can be configured by redefining or appending to the `\g_@@_diagrams_mmdc_command_tl` token list, by redefining the `\mmdcCommand` macro, or using the HTML attribute `command` on the fenced code.

Unlike the token list, the macro can be redefined even before loading the Markdown package. Unlike the token list and the macro, the HTML attribute is local to a single fenced code.

```
  \tl_new:N
    \g_@@_diagrams_mmdc_command_tl
  \tl_gset:Nn
    \g_@@_diagrams_mmdc_command_tl
    {
      \tl_if_empty:NTF
        \l_@@_diagrams_command_tl
        { mmdc }
        {
          \tl_use:N
            \l_@@_diagrams_command_tl
        }
    }
  \cs_if_free:NT
    \mmdcCommand
    {
      \cs_new:Npn
        \mmdcCommand
        {
          \tl_use:N
            \g_@@_diagrams_mmdc_command_tl
        }
    }
  \cs_set:Nn
    \@@_diagrams_infostrings_current:n
    {
      \tl_set:Nx
        \l_tmpb_tl
        { \mmdcCommand }
      \tl_put_right:Nn
        \l_tmpb_tl
        { ~ }
      \@@_diagrams_render_diagram:nnxxn
        { #1 }
        { #1.pdf }
```

```
          {
            \tl_use:N
              \l_tmpb_tl
            --version
          }
          {
            \tl_use:N
              \l_tmpb_tl
```

Do not pass the removed `--pdfFit` option to `mmdc` $\geqslant$ 10.0.0, where its previous semantics are now the default.

```
            \tl_if_empty:NF
              \l_markdown_diagrams_command_version_tl
              {
                \int_compare:VNnT
                  \l_markdown_diagrams_command_version_tl
                  <
                  { 10 }
                  { --pdfFit~ }
              }
            -i~#1~--o~#1.pdf
          }
          { Mermaid~image }
      }
    \cs_generate_variant:Nn
      \@@_diagrams_render_diagram:nnnnn
      { nnxxn }
    \@@_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
      {
```

Use the output format provided by the user.

```
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \regex_replace_once:NxN
```

```
          \c_@@_diagrams_filename_suffix_regex
          {
            .
            \tl_use:N
              \l_@@_diagrams_format_tl
          }
          \l_tmpa_tl
        \tl_set:Nn
          \l_tmpb_tl
          { plantuml~-t }
        \tl_put_right:NV
          \l_tmpb_tl
          \l__markdown_diagrams_format_tl
        \tl_put_right:Nn
          \l_tmpb_tl
          { ~#1 }
```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```
        \str_if_eq:VnT
          \l_@@_diagrams_format_tl
          { svg }
          {
            \tl_put_right:Nn
              \l_tmpb_tl
              { ;~inkscape~ }
            \tl_put_right:NV
              \l_tmpb_tl
              \l_tmpa_tl
            \tl_put_right:Nn
              \l_tmpb_tl
              { ~--export-area-drawing~--export-dpi=300~-o~ }
            \tl_set:Nn
              \l_tmpa_tl
              { #1 }
            \regex_replace_once:NnN
              \c_@@_diagrams_filename_suffix_regex
              { .pdf }
              \l_tmpa_tl
            \tl_put_right:NV
              \l_tmpb_tl
              \l_tmpa_tl
          }
        \@@_diagrams_render_diagram:nVnVn
          { #1 }
          \l_tmpa_tl
          { }
          \l_tmpb_tl
          { PlantUML~image }
      }
    \cs_generate_variant:Nn
```

```
      \@@_diagrams_render_diagram:nnnnn
      { nVnVn }
    \cs_generate_variant:Nn
      \regex_replace_once:NnN
      { NxN }
    \cs_generate_variant:Nn
      \sys_get_shell:nnN
      { xnN }
    \cs_generate_variant:Nn
      \sys_shell_now:n
      { V }
    \prg_generate_conditional_variant:Nnn
      \int_compare:nNn
      { VNn }
      { T }
    \@@_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_protected:Npn
  \markdownRendererHeaderAttributeContextBeginPrototype
  {
    \group_begin:
    \color_group_begin:
  }
\cs_gset_protected: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_protected: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}}%
\def\markdownRendererBracketedSpanPrototype#1{#1}%
```

#### 3.2.3.1 CommonMark Raw html Renderers

The CommonMark raw HTML renderer prototypes (see Section 2.2.5.13) fall back onto the renderer for generic inline HTML tags.

```
\def\markdownRendererInlineHtmlProcessingInstructionPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
\def\markdownRendererInlineHtmlDeclarationPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
\def\markdownRendererInlineHtmlCdataSectionPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
\def\markdownRendererInlineHtmlOpenTagPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
\def\markdownRendererInlineHtmlCloseTagPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
\def\markdownRendererInlineHtmlEmptyTagPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
```

### 3.2.3.2 CommonMark Block html Renderers

Most CommonMark block HTML renderer prototypes (see Section 2.2.5.12) fall back onto the renderer prototype for generic block HTML elements.

```
\def\markdownRendererInputBlockHtmlCdataElementPrototype{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\def\markdownRendererInputBlockHtmlProcessingInstructionPrototype{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\def\markdownRendererInputBlockHtmlDeclarationPrototype{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\def\markdownRendererInputBlockHtmlCdataSectionPrototype{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\def\markdownRendererInputBlockHtmlPcdataElementPrototype{%
  \markdownRendererInputBlockHtmlElementPrototype}%
```

The block comment HTML renderer (type 2) falls back on the renderer prototype for inline HTML elements.

```
\def\markdownRendererBlockHtmlCommentPrototype{%
  \markdownRendererInlineHtmlCommentPrototype}%
```

The standalone HTML tag renderer (type 7) falls back on the renderer prototype for generic inline HTML tags.

```
\def\markdownRendererBlockHtmlStandaloneTagPrototype{%
  \markdownRendererInlineHtmlTagPrototype}%
```

### 3.2.3.3 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
```

#### 3.2.3.4 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_protected: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_protected: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_protected: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_protected: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_protected: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_protected: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.

```
\cs_gset_protected:Npn
  \markdownRendererJekyllDataSequenceBeginPrototype #1#2
  {
    \@@_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_sequence_tl
  }
\cs_gset_protected:Npn
  \markdownRendererJekyllDataMappingBeginPrototype #1#2
  {
    \@@_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_mapping_tl
  }
\cs_gset_protected:Npn
  \markdownRendererJekyllDataSequenceEndPrototype
  {
    \@@_jekyll_data_pop:
  }
\cs_gset_protected:Npn
  \markdownRendererJekyllDataMappingEndPrototype
  {
    \@@_jekyll_data_pop:
  }
\cs_gset_protected:Npn
  \markdownRendererJekyllDataBooleanPrototype #1#2
  {
    \@@_jekyll_data_set_keyvals_known:nn
      { #1 }
      { #2 }
  }
\def\markdownRendererJekyllDataEmptyPrototype#1{}
```

```
\cs_gset_protected:Npn
  \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{}
\cs_gset_protected:Npn
  \markdownRendererJekyllDataTypographicStringPrototype #1#2
  {
    \@@_jekyll_data_set_keyvals_known:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

#### 3.2.3.5 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 .value_required:n = { false },
        #1 .default:n = { },
      }
  }
\seq_new:N
  \l_@@_jekyll_data_current_position_seq
\tl_new:N
  \l_@@_jekyll_data_current_position_tl
\cs_new_protected: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 }
      }
```

```
16197           },
```

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`.

```
16198           jekyllDataEmpty = {
16199             \tl_set:Nx
16200               \l_@@_jekyll_data_current_position_tl
16201               {
16202                 \seq_use:Nn
16203                   \l_@@_jekyll_data_current_position_seq
16204                   { / }
16205               }
16206             \keys_if_exist:VnTF
16207               \l_@@_jekyll_data_current_position_tl
16208               { ##1 / empty }
16209               {
16210                   \keys_set:xn
16211                   {
16212                     \tl_use:N
16213                       \l_@@_jekyll_data_current_position_tl
16214                     / ##1
16215                   }
16216                   { empty }
16217               }
16218               {
16219                 \keys_set:Vn
16220                   \l_@@_jekyll_data_current_position_tl
16221                   { ##1 }
16222               }
16223           },
```

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.

```
16224           jekyllDataTypographicString = {
16225             \tl_set:Nx
16226               \l_@@_jekyll_data_current_position_tl
16227               {
16228                 \seq_use:Nn
16229                   \l_@@_jekyll_data_current_position_seq
16230                   { / }
16231               }
16232             \keys_if_exist:VnTF
16233               \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_protected:Nn
  \@@_keys_set:nn
  {
    \keys_set:nn
      { }
      { { #1 } = { #2 } }
  }
\cs_new_protected: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_protected: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_protected: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
```

```
16372             \l_tmpa_tl
16373             \c_@@_option_type_counter_tl
16374         }
16375           {
16376             \@@_get_option_value:nN
16377               { #1 }
16378               \l_tmpa_tl
16379             \tl_gput_right:Nx
16380               \g_@@_formatted_lua_options_tl
16381               { #1~=~ \l_tmpa_tl ,~ }
16382           }
16383         {
16384           \str_if_eq_p:VV
16385             \l_tmpa_tl
16386             \c_@@_option_type_clist_tl
16387         }
16388           {
16389             \@@_get_option_value:nN
16390               { #1 }
16391               \l_tmpa_tl
16392             \tl_gput_right:Nx
16393               \g_@@_formatted_lua_options_tl
16394               { #1~=~\c_left_brace_str }
16395             \clist_map_inline:Vn
16396               \l_tmpa_tl
16397               {
16398                 \@@_lua_escape:xN
16399                   { ##1 }
16400                   \l_tmpb_tl
16401                 \tl_gput_right:Nn
16402                   \g_@@_formatted_lua_options_tl
16403                   { " }
16404                 \tl_gput_right:NV
16405                   \g_@@_formatted_lua_options_tl
16406                   \l_tmpb_tl
16407                 \tl_gput_right:Nn
16408                   \g_@@_formatted_lua_options_tl
16409                   { " ,~ }
16410               }
16411             \tl_gput_right:Nx
16412               \g_@@_formatted_lua_options_tl
16413               { \c_right_brace_str ,~ }
16414           }
16415       }
16416       {
16417         \@@_get_option_value:nN
16418           { #1 }
16419           \l_tmpa_tl
16420         \@@_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_protected:Nn
      \@@_lua_escape:nN
      {
        \tl_set:Nx
          #2
          {
            \lua_escape:n
              { #1 }
          }
      }
  }
  {
    \regex_const:Nn
      \c_@@_lua_escape_regex
      { [\\"'] }
    \cs_new_protected: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_protected: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.

```
    local~lfs = require("lfs")
    local~options = \markdownLuaOptions
    if~not~lfs.isdir(options.cacheDir) then~
      assert(lfs.mkdir(options.cacheDir))
    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.

```
    local~md = require("markdown")
    local~convert = md.new(options)
  }
```

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.

```
\cs_new:Npn
  \markdownConvert
  {
    local~filename = \markdownInputFilename
    local~file = assert(io.open(filename, "r"),
      [[Could~not~open~file~"]] .. filename .. [["~for~reading]])
    local~input = assert(file:read("*a"))
    assert(file:close())
    print(convert(input))
  }
\ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TEX.

```
\def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
  if options.cacheDir then
    lfs.rmdir(options.cacheDir)
  end
}%
```

### 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.

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

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

```
\begingroup
  \catcode`\^^I=12%
  \gdef\markdownReadAndConvertTab{^^I}%
\endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the LATEX $2_\varepsilon$ `\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%
```

```
16531   |catcode`|%=12@
16532   |gdef|markdownReadAndConvert#1#2{@
16533     |begingroup@
```

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

```
16534     |markdownIfOption{frozenCache}{}{@
16535       |immediate|openout|markdownOutputFileStream@
16536         |markdownOptionInputTempFileName|relax@
16537       |markdownInfo{@
16538         Buffering block-level markdown input into the temporary @
16539         input file "|markdownOptionInputTempFileName" and scanning @
16540         for the closing token sequence "#1"}@
16541     }@
```

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.

```
16542     |def|do##1{|catcode`##1=12}|dospecials@
16543     |catcode`| =12@
16544     |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.

```
16545     |def|markdownReadAndConvertStripPercentSign##1{@
16546       |markdownIfOption{stripPercentSigns}{@
16547         |if##1%@
16548           |expandafter|expandafter|expandafter@
16549             |markdownReadAndConvertProcessLine@
16550         |else@
16551           |expandafter|expandafter|expandafter@
16552             |markdownReadAndConvertProcessLine@
16553             |expandafter|expandafter|expandafter##1@
16554         |fi@
16555       }{@
16556         |expandafter@
16557           |markdownReadAndConvertProcessLine@
16558           |expandafter##1@
16559       }@
16560     }@
```

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.

```
16561     |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_protected: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_protected: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_after:Nw
      \@@_inline_read_markdown_text:
  }
\cs_new_protected:Nn
  \@@_inline_read_markdown_text:
  {
```

Allow both `{`⟨*markdown text*⟩`}` but also any other delimiter such as `|`⟨*markdown text*⟩`|`.

```
    \bool_if:nTF
      {
        \token_if_group_begin_p:N
          \l_peek_token ||
```

```
        \token_if_eq_charcode_p:NN
          \l_peek_token
          \c_group_begin_token
      }
      {
        \regex_set:Nn
          \l_tmpa_regex
          { ( (.*?) ) }
      }
      {
        \tl_set:Nx
          \l_tmpa_tl
          {
            \text_purify:n
              { \l_peek_token }
          }
        \regex_set:Nn
          \l_tmpa_regex
          {
            \u { l_tmpa_tl }
            (.*?)
            \u { l_tmpa_tl }
          }
      }
    \peek_regex_replace_once:NnF
      \l_tmpa_regex
      {
```

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_protected: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 [20, Section 9]. As a consequence, we can directly reuse the existing plain TeX implementation.

```
\def\markdownVersionSpace{ }%
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
```

```
16830   \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).

```
16831 \ExplSyntaxOn
16832 \cs_gset_eq:NN
16833   \markinlinePlainTeX
16834   \markinline
16835 \cs_gset_protected:Npn
16836   \markinline
16837   {
16838     \peek_regex_replace_once:nn
16839       { ( \[ (.*?) \] ) ? }
16840       {
```

Apply the options locally.

```
16841         \c { group_begin: }
16842         \c { @@_setup:n }
16843           \cB { \2 \cE }
16844         \c { tl_put_right:Nn }
16845           \c { g_@@_after_markinline_tl }
16846           \cB { \c { group_end: } \cE }
16847         \c { markinlinePlainTeX }
16848       }
16849   }
16850 \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).

```
16851 \let\markdownInputPlainTeX\markdownInput
16852 \renewcommand\markdownInput[2][]{%
16853   \begingroup
16854     \markdownSetup{#1}%
16855     \markdownInputPlainTeX{#2}%
16856   \endgroup}%
16857 \renewcommand\yamlInput[2][]{%
16858   \begingroup
16859     \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
16860     \markdownInputPlainTeX{#2}%
16861   \endgroup}%
```

The `markdown`, `markdown*`, and `yaml` LATEX environments are implemented using the `\markdownReadAndConvert` macro.

```
16862 \ExplSyntaxOn
```

```
\renewenvironment
  { markdown }
  {
```

In our implementation of the `markdown` LaTeX environment, we want to distinguish between the following two cases:

```
\begin{markdown} [smartEllipses]   \begin{markdown}
% This is an optional argument ^     [smartEllipses]
% ...                              % ^ This is link
\end{markdown}                     \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 11 (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}
```

```
16875         \c { exp_args:NV }
16876           \c { markdownReadAndConvert@ }
16877           \c { @currenvir }
16878       }
16879       {
16880         \group_end:
16881         \exp_args:NV
16882           \markdownReadAndConvert@
16883           \@currenvir
16884       }
16885   }
16886   { \markdownEnd }
16887 \renewenvironment
16888   { markdown* }
16889   [ 1 ]
16890   {
16891     \@@_if_option:nTF
16892       { experimental }
16893       {
16894         \msg_error:nn
16895           { markdown }
16896           { latex-markdown-star-deprecated }
16897       }
16898       {
16899         \msg_warning:nn
16900           { markdown }
16901           { latex-markdown-star-deprecated }
16902       }
16903     \@@_setup:n
16904       { #1 }
16905     \markdownReadAndConvert@
16906       { markdown* }
16907   }
16908   { \markdownEnd }
16909 \renewenvironment
16910   { yaml }
16911   {
16912     \group_begin:
16913     \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  % noqa: w402
  \@@_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_protected: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:nn
            { markdown }
            { latex-theme-after-preamble }
        }
```

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
```

The witiko/glossaries theme first checks that the version is either v1 or latest, issuing a warning for the latter.

```
\prop_gput:Nnn
  \g_@@_latex_built_in_themes_prop
  { witiko / glossaries }
  {
    \str_case:enF
      { \markdownThemeVersion }
      {
        { latest }
          {
            \msg_warning:nnnn
              { markdown }
              { pin-theme-version }
              { witiko/glossaries }
              { v1 }
          }
        { v1 }
          { }
      }
      {
        \msg_error:nnnen
          { markdown }
          { unknown-theme-version }
          { witiko/glossaries }
          { \markdownThemeVersion }
          { v1 }
      }
  }
\ExplSyntaxOff
```

In Section 3.3.3, we'll continue the implementation of the theme by defining the snippet witiko/glossaries/import-acronyms. 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 3.3.5 for the actual definitions.

### 3.3.3 Snippets

This section defines the built-in LaTeX snippet witiko/glossaries/import-acronyms.

```
\ExplSyntaxOn
```

```
\prop_get:NnN
  \g_@@_latex_built_in_themes_prop
  { witiko / glossaries }
  \l_tmpa_tl
\tl_put_right:Nn
  \l_tmpa_tl
  {
    \cs_new:Nn
      \@@_register_acronym:n
        {
          \@@_setup:n
            {
```

When we register an acronym with the `acronyms` option, enclose it in an extra set of braces in case the acronym contained a comma.

```
              acronyms += {{ #1 }},
            }
        }
    \cs_generate_variant:Nn
      \@@_register_acronym:n
      { V }
    \prop_new:N
      \g_@@_acronyms_to_labels_prop
    \tl_new:N
      \l_@@_acronym_short_tl
```

When we import an acronym from the glossaries package, we receive a ⟨*label*⟩, which doesn't correspond to an acronym. Therefore, we register the short form of the corresponding acronym instead and keep a mapping from these short forms back to the labels when we need to display an acronym in the markdown text.

```
    \cs_new:Nn
      \@@_register_acronym_label:n
      {
        \tl_set:Nx
          \l_@@_acronym_short_tl
          { \glsentryshort { #1 } }
```

Be idempotent and only import each acronym once.

```
        \prop_if_in:NVF
          \g_@@_acronyms_to_labels_prop
          \l_@@_acronym_short_tl
          {
            \@@_register_acronym:V
              \l_@@_acronym_short_tl
            \prop_gput:NVn
              \g_@@_acronyms_to_labels_prop
              \l_@@_acronym_short_tl
              { #1 }
          }
      }
```

```
    \cs_generate_variant:Nn
      \@@_register_acronym_label:n
      { V }
```

When we need to diplay an acronym in the markdown text, first consult our mapping to see if it has been imported from the glossaries package.

```
    \tl_new:N
      \l_@@_acronym_label_tl
    \cs_new:Nn
      \@@_print_acronym:n
      {
        \prop_get:NnNTF
          \g_@@_acronyms_to_labels_prop
          { #1 }
          \l_@@_acronym_label_tl
          {
```

If it has, retrieve the ⟨*label*⟩ and display the acronym using the glossaries package, linking it to the corresponding glossary entry.

```
            \group_begin:
              \cs_set_eq:NN
                \acronymfont
                \markdownRendererAcronymPrototype
              \exp_args:NV
                \acrshort
                \l_@@_acronym_label_tl
            \group_end:
          }
          {
```

Otherwise, just format the acronym using the current renderer prototype.

```
            \markdownRendererAcronymPrototype
              { #1 }
          }
      }
      \cs_generate_variant:Nn
        \@@_print_acronym:n
        { x }
```

In the snippet `witiko/glossaries/import-acronyms`, first import all acronyms from the glossaries package and register them with the `acronyms` option.

```
    \markdownSetupSnippet
      { import-acronyms }
      {
        code = {
          \forglsentries
            [ acronym ]
            \l_@@_acronym_label_tl
            {
              \@@_register_acronym_label:V
                \l_@@_acronym_label_tl
```

```
            }
```

Then, redefine the corresponding token renderers to format acronyms that appear in the markdown text.

```
            \@@_setup:n
              {
                renderers = {
                  acronym = {
```

Fold nested acronyms.

```
                    \group_begin:
                      \markdownSetup {
                        unprotectedRenderers = {
                          acronym = { ##1 },
                        }
                      }
                      \@@_print_acronym:x
                        { #1 }
                    \group_end:
                  }
                }
              }
          }
      }
  }
\prop_gput:NnV
  \g_@@_latex_built_in_themes_prop
  { witiko / glossaries }
  \l_tmpa_tl
\ExplSyntaxOff
```

### 3.3.4 Options

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

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

### 3.3.5 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}
```

#### 3.3.5.1 Acronyms

If the `acronyms` option is non-empty, render acronyms, initialisms, and other all-caps sequences using small caps.

```
\ExplSyntaxOn
```

```
\prop_new_linked:N
  \g_@@_recorded_acronyms_prop
\bool_new:N
  \g_@@_record_acronyms_bool
\bool_gset_false:N
  \g_@@_record_acronyms_bool
\cs_new_protected:Nn
  \@@_record_acronym:n
  {
```

For explicitly listed acronyms using the HTML class `acronym` that only contain simple text and not control sequences after cleaning up any potential nested acronyms, record them in the auxiliary file, so that they are applied throughout the document in the next compilation run.

```
    \bool_if:NT
      \g_@@_record_acronyms_bool
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \regex_replace_all:nnN
          { \c { markdownRendererAcronym } | \cB\{ | \cE\} }
          { }
          \l_tmpa_tl
        \prop_if_in:NVF
          \g_@@_recorded_acronyms_prop
          \l_tmpa_tl
          {
            \regex_if_match:nVF
              { \cC. }
              \l_tmpa_tl
              {
                \prop_gput:NVn
                  \g_@@_recorded_acronyms_prop
                  \l_tmpa_tl
                  { true }
                \tl_gput_right:Nx
                  \markdownOptionAcronyms
                  { , { \tl_use:N \l_tmpa_tl } }
                \tl_set:Nn
                  \l_tmpb_tl
                  {
                    \ExplSyntaxOn
                      \tl_gput_right:Nn
                      \markdownOptionAcronyms
                  }
                \tl_gput_right:Nx
                  \l_tmpb_tl
                  { { , { \tl_use:N \l_tmpa_tl } } }
```

```
                \tl_gput_right:Nx
                  \l_tmpb_tl
                  { \ExplSyntaxOff }
                \iow_now:NV
                  \@auxout
                  \l_tmpb_tl
              }
          }
      }
  }
\int_new:N
  \g_@@_heading_nesting_depth_int
\markdownSetup {
  rendererPrototypes = {
    acronym = {
```

Prevent acronyms in headings.

```
      \int_if_zero:nTF
        { \g_@@_heading_nesting_depth_int }
        {
```

Prevent nested acronyms.

```
          \group_begin:
            \markdownSetup {
              unprotectedRenderers = {
                acronym = { ##1 },
              }
            }
            \@@_record_acronym:n
              { #1 }
            \tl_set:Nn
              \l_tmpa_tl
              { #1 }
```

Render all characters that are neither digits nor ASCII lower case characters in small caps. Ideally, we would also be able to exclude Unicode lower-case characters but that doesn't seem possible with expl3 at the moment [21, Chapter 34 (The l3unicode module)].

```
            \regex_replace_all:nnN
              { [^\d a-z]+ }
              {
                \c{ textsc } \cB\{
                  \c{ MakeLowercase } \cB\{ \0 \cE\}
                \cE\}
              }
              \l_tmpa_tl
```

Furthermore, if the current typeface supports it, render all digits using old-style numerals.

```
            \cs_if_exist:NT
              \oldstylenums
              {
                \regex_replace_all:nnN
```

```
                    { \d+ }
                    { \c{ oldstylenums } \cB\{ \0 \cE\} }
                    \l_tmpa_tl
                }
```

If the package microtype is loaded, letter-space the small caps.

```
              \@ifpackageloaded
                { microtype }
                {
                  \textls
                    {
                      \tl_use:N
                        \l_tmpa_tl
                    }
                }
                {
                  \tl_use:N
                    \l_tmpa_tl
                }
            \group_end:
          }
```

Prevent acronyms in headings.

```
          { #1 }
      },
    },
}
```

If the package glossaries is loaded with the option `acronym`, use the built-in snippet LaTeX `witiko/glossaries/import-acronyms`.

```
\@ifpackageloaded
  { glossaries }
  {
    \cs_if_exist:NT
      \@gls@do@acronymsdef
      {
        \markdownSetup {
           import = /witiko/glossaries@v1,
          snippet = /witiko/glossaries/import-acronyms,
        }
      }
  }
  { }
```

#### 3.3.5.2 Lists

If either the `tightLists` or the `fancyLists` 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 the command `\DocumentMetadata` have been used, use the package enumitem. Otherwise, use the package paralist.

```
\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_new:N
  \g_@@_paralist_bool
\@ifpackageloaded
  { paralist }
  {
    \bool_gset_true:N
      \g_@@_paralist_bool
```

```
  }
  {
    \bool_gset_false:N
      \g_@@_paralist_bool
  }
\bool_new:N
  \g_@@_enumitem_bool
\@ifpackageloaded
  { enumitem }
  {
    \bool_gset_true:N
      \g_@@_enumitem_bool
  }
  {
    \bool_gset_false:N
      \g_@@_enumitem_bool
  }
\bool_if:nT
  {
    \g_@@_tight_or_fancy_lists_bool &&
    ! \g_@@_beamer_paralist_or_enumitem_bool
  }
  {
    \str_if_eq:enTF
      { \markdownThemeVersion }
      { experimental }
      {
        \RequirePackage
          { enumitem }
        \bool_gset_true:N
          \g_@@_enumitem_bool
      }
      {
        \IfDocumentMetadataTF
          {
```

Since LaTeX 2$_\varepsilon$ 2026-06-01, the enumitem package can't be loaded and its interfaces are emulated when tagging is used. [4] Therefore, there is no need to (or point in) trying to load the package and we can just act as though it has already been loaded.

```
            \bool_gset_true:N
              \g_@@_enumitem_bool
          }
          {
            \RequirePackage
              { paralist }
            \bool_gset_true:N
              \g__markdown_paralist_bool
          }
      }
  }
```

```
\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
\bool_if:NTF \g_@@_enumitem_bool {
  \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.

```
}
{ \bool_if:NTF \g_@@_paralist_bool {
  \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,[41] we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```
  codeSpan = {%
    \ifmmode
      \text{#1}%
    \else
      \texttt{#1}%
    \fi
  }}}
```

#### 3.3.5.3 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 }
            {
```

[41] 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.

If we are using TEX4ht[42], we will pass HTML elements to the output HTML document unchanged.

```
            \cs_if_exist:NTF
              \HCode
              {
                \if_mode_vertical:
                  \IgnorePar
                \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.

[42]See https://tug.org/tex4ht/.

```
\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 = -2
                    \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}}}}
```

#### 3.3.5.4 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
\cs_new_protected:Npn
  \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}}}}
}%
```

Prevent acronyms in headings.

```
\ExplSyntaxOn
\markdownSetup {
  rendererPrototypes = {
    heading* ^= {
      \int_gincr:N
        \g_@@_heading_nesting_depth_int
    },
    heading* $= {
      \int_gdecr:N
        \g_@@_heading_nesting_depth_int
    }
  }
}
\ExplSyntaxOff
```

#### 3.3.5.5 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
}
```

#### 3.3.5.6 html elements

If the `html` option is enabled and we are using TEX4ht[43], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

#### 3.3.5.7 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{%
```

[43] See https://tug.org/tex4ht/.

```
\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}%
    }}}}
```

#### 3.3.5.8 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 }
      }
  }
\cs_new_protected:Npn
  \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}}}
```

#### 3.3.5.9 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}
```

#### 3.3.5.10 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
    },
  }}
}{}
```

#### 3.3.5.11 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 .value_required:n = { true },
    author .code:n = {
      \author
        { #1 }
    },
    date .value_required:n = { true },
    date .code:n = {
      \date
        { #1 }
    },
    title .value_required:n = { true },
    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, temporarily resetting the category codes of the percent sign and the hash sign back to comment and parameter, respectively.

```
      \char_set_catcode_comment:N \%
```

```
      \char_set_catcode_parameter:N \#
      \AddToHook
        { begindocument / end }
        { \maketitle }
      \char_set_catcode_other:N \%
      \char_set_catcode_other:N \#
    },
  }
```

#### 3.3.5.12 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 }
              },
            }
          }
      }
  }
```

#### 3.3.5.13 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 }
              },
            }
          }
      }
  }
```

#### 3.3.5.14 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
```

#### 3.3.5.15 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[44], we will pass HTML elements to the output HTML document unchanged.

```
            \cs_if_exist:NTF
              \HCode
              {
```

[44] See https://tug.org/tex4ht/.

```
                \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 }
          {
```

If we are using TEX4ht[45], 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
              }
```

[45] See https://tug.org/tex4ht/.

```
            {
              \@@_luaxml_print_html:n
                { #1 }
            }
        }
        {
          \@@_plain_tex_default_input_raw_block:nn
            { #1 }
            { #2 }
        }
  }
```

#### 3.3.5.16 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 $= {
      \markdownSetup {
        rendererPrototypes = {
          attributeIdentifier = {
            \seq_put_right:Nn
              \l_@@_bracketed_span_identifiers_seq
              { ##1 }
          },
        },
      }
    },
    bracketedSpanAttributeContextEnd ^= {
      \seq_map_inline:Nn
        \l_@@_bracketed_span_identifiers_seq
        { \label { ##1 } }
    },
  },
}
```

We will also allow acronyms to be manually marked up using the HTML class `acronym`.

```
\markdownSetup {
  rendererPrototypes = {
    bracketedSpanAttributeContextBegin += {
      \markdownSetup {
        rendererPrototypes = {
          attributeClassName = {
            \str_if_eq:nnT
              { ##1 }
```

```
                { acronym }
                {
                  \markdownSetup {
                    rendererPrototypes = {
                      bracketedSpan = {
                        \bool_set_true:N
                          \g__markdown_record_acronyms_bool
                        \markdownRendererAcronym
                          { ####1 }
                      }
                    }
                  }
                }
            },
          },
        }
      },
    },
}
\ExplSyntaxOff
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
```

### 3.3.6 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. [22, sec. 31]. According to Eijkhout [23, 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_protected:Nn
  \@@_load_theme:nnn
  {
```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_them` 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] Hans Hagen. *ConTEXt Lua Documents*. July 8, 2023. URL: https://www.pragma-ade.nl/general/manuals/cld-mkiv.pdf (visited on 09/22/2025).

[2] LuaTEX development team. *LuaTEX reference manual*. Version 1.21. Feb. 1, 2025. URL: http://mirrors.ctan.org/systems/doc/luatex/luatex.pdf (visited on 05/12/2025).

[3] LATEX Project. *l3kernel. LATEX3 programming conventions*. Dec. 25, 2024. URL: https://ctan.org/pkg/l3kernel (visited on 01/06/2025).

[4] LATEX Project. *LATEX News. Issue 43*. LATEX release 2026-06-01. June 2026. URL: https://www.latex-project.org/news/latex2e-news/ltnews43.pdf (visited on 11/17/2026).

[5] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[6] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[7] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[8] 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.

[9] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[10] 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).

[11] 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).

[12] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: https://github.com/Witiko/markdown/discussions/514 (visited on 10/21/2024).

[13] 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).

[14] Vít Starý Novotný. *In l3keys, `unknown` keys don't support `.append_method:n`*. Nov. 24, 2026. URL: https://github.com/latex3/latex3/issues/1997 (visited on 11/26/2026).

[15] 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).

[16] 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).

[17] 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).

[18] Unicode Consortium. *The Unicode Standard. Version 16.0 – Core Specification*. Sept. 10, 2024. URL: https://www.unicode.org/versions/Unicode16.0.0/UnicodeStandard-16.0.pdf (visited on 05/07/2025).

[19] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[20] 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).

[21] LaTeX Project. *The LaTeX3 Interfaces*. Jan. 19, 2026. URL: https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf (visited on 02/18/2026).

[22] Donald Ervin Knuth. *TeX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

[23] 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

`acronyms` 22, 90, 171, 467–469
`autoIdentifiers` *22*, 34, 91, 111

`blankBeforeBlockquote` *23*
`blankBeforeCodeFence` *23*

blankBeforeDivFence *24*
blankBeforeHeading *24*
blankBeforeHtmlBlock *24*
blankBeforeList *24*
bracketedSpans *25*, 94, 502
breakableBlockquotes *25*

cacheDir 4, 17, 20, 60, 169, 199, 405, 428, 448
citationNbsps *25*
citations *26*, 97
codeSpans *26*
contentBlocks 21, *27*, 37
contentBlocksLanguageMap 21
contentLevel *27*

debugExtensions 9, 21, *27*, 349
debugExtensionsFileName 21, 27
defaultOptions *9*, 53, 403, 405
definitionLists *28*, 105
depth_first_search *177*

eagerCache *17*, 403
ensureJekyllData *28*, 29, 30, 39
entities.char_entity *251*
entities.dec_entity *250*
entities.hex_entity *250*
entities.hex_entity_with_x_char *251*
escape_minimal *255*
escape_programmatic_text *255*
escape_typographic_text *255*
expandtabs *310*
expectJekyllData 28, 29, *29*, 30, 39
experimental 4, *18*, 36, 473
experimentalOptions *9*, 403, 405
extensions 30, *176*, 354
extensions.acronyms 178, *354*
extensions.citations *357*
extensions.content_blocks *361*
extensions.definition_lists *364*
extensions.fancy_lists *366*
extensions.fenced_code *372*
extensions.fenced_divs *377*
extensions.header_attributes *381*
extensions.inline_code_attributes *382*
extensions.jekyll_data *399*

extensions.line_blocks *383*
extensions.link_attributes *384*
extensions.mark *384*
extensions.notes *386*
extensions.pipe_table *388*
extensions.raw_inline *392*
extensions.strike_through *393*
extensions.subscripts *394*
extensions.superscripts *394*
extensions.tex_math *395*

fancyLists *31*, 120–125, 472
fencedCode *32*, 42, 98, 110, 126, 416, 419
fencedCodeAttributes *32*, 91, 110, 419
fencedDivs *33*, 43, 110
finalizeCache 18, 22, *33*, 34, 60, 168, 403, 404
frozenCache 22, 33, 60, 80, 168, 416, 428
frozenCacheCounter 34, 404, 456, 457
frozenCacheFileName 22, 33, 60, 404

\g_markdown_diagrams_infostrings_prop *422*
gfmAutoIdentifiers 23, *34*, 91, 111

hashEnumerators *34*
headerAttributes *35*, 43, 91, 111
html 35, *35*, 36, 92, 100, 102, 486
htmlOutput *35*, 92, 100, 102
htmlOverLinks 18, *36*
hybrid 36, *36*, 42, 47, 49, 63, 80, 127, 169, 255, 312, 456

inlineCodeAttributes *38*, 91, 99
inlineNotes *38*
\inputmarkdown *172*, 173, 174, 504
inputTempFileName 61, 63, 450, 451, 453, 454
\inputyaml 29, 30, 39, *172*, 174, 504
iterlines *310*

jekyllData 3, 28–30, *38*, 39, 136–139, 141

\l_markdown_diagrams_command_version_tl 421, *422*
languages_json 361, *361*
lineBlocks *39*, 116
linkAttributes *39*, 91, 114, 117, 330, 498

mark *40*, 118, 497
\markdown 164, *164*, 165, 166, 460

markdown *163*, 164, 458, 459
markdown* *163*, 164, 168, 458
\markdownBegin *54*, 55–57, 162, 164, 165, 172, 173
\markdownCleanup *449*
\markdownConvert *448*
\markdownEnd *54*, 55–57, 162, 164–166, 172, 173
\markdownError 161, *161*, 162
\markdownEscape *54*, 58, 457
\markdownIfOption *59*
\markdownIfSnippetExists *84*
\markdownInfo *161*, 162
\markdownInput *54*, 57, 163, 166, 168, 173, 455, 458
\markdownInputFilename *448*
\markdownInputFileStream *449*
\markdownInputPlainTeX *458*
\markdownLoadPlainTeXTheme *169*, 176, 415
\markdownLuaExecute *451*, 455
\markdownLuaOptions *445*, 448
\markdownMakeOther *162*, 503
\markdownOptionExperimental *64*
\markdownOptionFinalizeCache *60*
\markdownOptionFrozenCache *60*
\markdownOptionInputTempFileName *60*
\markdownOptionNoDefaults *62*
\markdownOptionOutputDir 61, *61*, 64, 65
\markdownOptionPlain *62*
\markdownOptionStripPercentSigns *63*
\markdownOutputFileStream *449*
\markdownPrepare *448*
\markdownPrepareInputFilename *448*
\markdownPrepareLuaOptions *445*
\markdownReadAndConvert *162*, 449, 458–460, 504
\markdownReadAndConvertProcessLine *450*, 451
\markdownReadAndConvertStripPercentSigns *450*
\markdownReadAndConvertTab *449*
\markdownRendererAcronym *90*
\markdownRendererAttributeClassName *91*
\markdownRendererAttributeIdentifier *91*
\markdownRendererAttributeKeyValue *91*
\markdownRendererBlockHtmlComment *100*
\markdownRendererBlockHtmlStandaloneTag *100*
\markdownRendererBlockQuoteBegin *93*
\markdownRendererBlockQuoteEnd *93*
\markdownRendererBracketedSpan *94*

\markdownRendererBracketedSpanAttributeContextBegin *94*
\markdownRendererBracketedSpanAttributeContextEnd *94*
\markdownRendererCite 97, *97*
\markdownRendererCodeSpan *99*
\markdownRendererCodeSpanAttributeContextBegin *99*
\markdownRendererCodeSpanAttributeContextEnd *99*
\markdownRendererContentBlock 104, *104*
\markdownRendererContentBlockCode *105*
\markdownRendererContentBlockOnlineImage *104*
\markdownRendererDisplayMath *133*
\markdownRendererDlBegin *105*
\markdownRendererDlBeginTight *106*
\markdownRendererDlDefinitionBegin *107*
\markdownRendererDlDefinitionEnd *107*
\markdownRendererDlEnd *108*
\markdownRendererDlEndTight *108*
\markdownRendererDlItem *106*
\markdownRendererDlItemEnd *107*
\markdownRendererDocumentBegin *119*
\markdownRendererDocumentEnd *119*
\markdownRendererEllipsis 44, *108*
\markdownRendererEmphasis *109*, 145
\markdownRendererError *135*
\markdownRendererFancyOlBegin 121, *121*
\markdownRendererFancyOlBeginTight *121*
\markdownRendererFancyOlEnd *125*
\markdownRendererFancyOlEndTight *125*
\markdownRendererFancyOlItem *123*
\markdownRendererFancyOlItemEnd *123*
\markdownRendererFancyOlItemWithNumber *124*
\markdownRendererFencedCodeAttributeContextBegin *110*
\markdownRendererFencedCodeAttributeContextEnd *110*
\markdownRendererFencedDivAttributeContextBegin *110*
\markdownRendererFencedDivAttributeContextEnd *110*
\markdownRendererHalfTickedBox *134*
\markdownRendererHardLineBreak *117*
\markdownRendererHeaderAttributeContextBegin *111*
\markdownRendererHeaderAttributeContextEnd *111*
\markdownRendererHeadingFive *113*
\markdownRendererHeadingFour *113*
\markdownRendererHeadingOne *112*
\markdownRendererHeadingSix *113*
\markdownRendererHeadingThree *112*
\markdownRendererHeadingTwo *112*

\markdownRendererImage *114*
\markdownRendererImageAttributeContextBegin *114*
\markdownRendererImageAttributeContextEnd *114*
\markdownRendererInlineHtmlCdataSection *102*
\markdownRendererInlineHtmlCloseTag *102*
\markdownRendererInlineHtmlComment *92*
\markdownRendererInlineHtmlDeclaration *102*
\markdownRendererInlineHtmlEmptyTag *102*
\markdownRendererInlineHtmlOpenTag *102*
\markdownRendererInlineHtmlProcessingInstruction *102*
\markdownRendererInlineHtmlTag *92*, 102
\markdownRendererInlineMath *133*
\markdownRendererInputBlockHtmlCdataElement *100*
\markdownRendererInputBlockHtmlCdataSection *100*
\markdownRendererInputBlockHtmlDeclaration *100*
\markdownRendererInputBlockHtmlElement *92*, 100
\markdownRendererInputBlockHtmlPcdataElement *100*
\markdownRendererInputBlockHtmlProcessingInstruction *100*
\markdownRendererInputFencedCode *98*
\markdownRendererInputRawBlock *126*
\markdownRendererInputRawInline *126*
\markdownRendererInputVerbatim *98*
\markdownRendererInterblockSeparator *115*
\markdownRendererJekyllDataBegin *136*
\markdownRendererJekyllDataBoolean *138*
\markdownRendererJekyllDataEmpty *141*
\markdownRendererJekyllDataEnd *136*
\markdownRendererJekyllDataMappingBegin *136*
\markdownRendererJekyllDataMappingEnd *137*
\markdownRendererJekyllDataNumber *138*
\markdownRendererJekyllDataProgrammaticString 139, *139*, 140
\markdownRendererJekyllDataSequenceBegin *137*
\markdownRendererJekyllDataSequenceEnd *138*
\markdownRendererJekyllDataString 140, *140*, 144
\markdownRendererJekyllDataStringPrototype *156*
\markdownRendererJekyllDataTypographicString 139, *139*, 140, 399
\markdownRendererLineBlockBegin *116*
\markdownRendererLineBlockEnd *116*
\markdownRendererLink *117*, 145
\markdownRendererLinkAttributeContextBegin *117*
\markdownRendererLinkAttributeContextEnd *117*
\markdownRendererMark *118*
\markdownRendererNbsp *119*
\markdownRendererNote *120*

\markdownRendererOlBegin *120*
\markdownRendererOlBeginTight *120*
\markdownRendererOlEnd *124*
\markdownRendererOlEndTight *124*
\markdownRendererOlItem 44, *122*
\markdownRendererOlItemEnd *122*
\markdownRendererOlItemWithNumber 44, *122*
\markdownRendererParagraphSeparator *115*
\markdownRendererReplacementCharacter *127*
\markdownRendererSectionBegin *126*
\markdownRendererSectionEnd *126*
\markdownRendererSoftLineBreak *116*
\markdownRendererStrikeThrough *130*
\markdownRendererStrongEmphasis *109*
\markdownRendererSubscript *131*
\markdownRendererSuperscript *131*
\markdownRendererTable *132*
\markdownRendererTableAttributeContextBegin *131*
\markdownRendererTableAttributeContextEnd *131*
\markdownRendererTextCite *97*
\markdownRendererThematicBreak *133*
\markdownRendererTickedBox *134*
\markdownRendererUlBegin *95*
\markdownRendererUlBeginTight *95*
\markdownRendererUlEnd *96*
\markdownRendererUlEndTight *97*
\markdownRendererUlItem *95*
\markdownRendererUlItemEnd *96*
\markdownRendererUntickedBox *134*
\markdownRendererWarning *135*
\markdownSetup 59, *59*, 63, 167, 168, 174, 459, 469
\markdownSetupSnippet 83, *83*
\markdownThemeVersion 73, *73*, 74
\markdownWarning *161*, 162
\markinline *54*, 56, 163, 166, 453, 458
\markinlinePlainTeX *458*
\mmdcCommand *424*

new *8*, 19, 403, 405
notes *40*, 120

parsers *274*, 310
parsers.commented_line *291*
parsers.unicode *276*

pipeTables 6, *41*, 46, 132
preserveTabs *41*, 45, 310

rawAttribute 37, 42, *42*, 126
read_decompositions *179*
reader *8*, 31, 176, 274, 309, 354
reader->add_special_character *8*, 9, 31, 348
reader->auto_link_email *337*
reader->auto_link_url *337*
reader->create_parser *311*
reader->finalize_grammar *344*, 410
reader->initialize_named_group *348*
reader->insert_pattern *8*, 9, 31, 344, 350
reader->lookup_note_reference *323*
reader->lookup_reference *323*
reader->normalize_tag *310*
reader->options *310*
reader->parser_functions *311*
reader->parser_functions.name *311*
reader->parsers 310, *310*
reader->register_link *322*
reader->update_rule 344, *347*, 350
reader->writer *310*
reader.new 309, *309*, 410
relativeReferences 36, *42*

serialize_byte_parser *178*
serialize_byte_range_parser *178*
serialize_replacement *178*
\setupmarkdown *174*, 175
\setupyaml *175*
shiftHeadings 6, 43
singletonCache *19*
slice 6, 43, 252, 266, 267
smartEllipses *44*, 108, 169
\startmarkdown 172, *172*, 504
startNumber *44*, 122–124
\startyaml 28, 30, 39, 172, *172*, 173, 504
\stopmarkdown 172, *172*, 504
\stopyaml 29, 30, 39, 172, *172*, 173, 504
strikeThrough *44*, 130, 498
stripIndent *45*, 311
stripPercentSigns 449, 450
subscripts *45*, 131

superscripts *45*, 131
syntax *346*, 350

tableAttributes *46*, 131, 494
tableCaptions 6, 46, *46*, 131
taskLists *47*, 134, 485
texComments *47*, 312
texMathDollars 37, *48*, 133
texMathDoubleBackslash 37, *48*, 133
texMathSingleBackslash 37, *48*, 133
tightLists *49*, 95, 97, 106, 108, 120, 121, 124, 125, 472

underscores *49*
unicode_data.casefold_mapping *189*, 199
unicode_data.categories *191*, 276
unicode_data.ccc *193*, 200
unicode_data.composition_mapping *185*, 204
unicode_data.decomposition_mapping *180*, 181, 202
unicodeNormalization *19*, 20
unicodeNormalizationForm 19, *20*
util.cache 195, *195*
util.cache_verbatim *195*
util.canonically_order *200*
util.casefold *199*
util.compose *204*
util.decompose *202*
util.encode_json_string *195*
util.err *194*
util.escaper *198*
util.expand_tabs_in_line *195*
util.find_file *206*
util.find_files *206*
util.flatten *196*
util.intersperse *197*
util.map *197*
util.normalize *206*
util.pathname *198*
util.rope_last *197*
util.rope_to_string *197*
util.salt *199*
util.table_copy *195*
util.walk *196*, 197
util.warning *199*

walkable_syntax *8*, 21, 27, 344, 347–350

writer 176, *176*, 251, 354
writer->acronym *354*
writer->active_attributes 266, *266*, 267
writer->attribute_type_levels *266*
writer->attributes *263*
writer->block_html_cdata_element *259*
writer->block_html_cdata_section *260*
writer->block_html_comment *260*
writer->block_html_declaration *260*
writer->block_html_element *259*
writer->block_html_pcdata_element *260*
writer->block_html_processing_instruction *260*
writer->block_html_standalone_tag *261*
writer->blockquote *262*
writer->bulletitem *258*
writer->bulletlist *258*
writer->citations *357*
writer->code *256*
writer->contentblock *362*
writer->defer_call 273, *273*
writer->definitionlist *364*
writer->display_math *395*
writer->div_begin *377*
writer->div_end *377*
writer->document *263*
writer->ellipsis *254*
writer->emphasis *262*
writer->error *256*
writer->escape *255*
writer->escaped_chars *254*, 255
writer->escaped_minimal_strings *254*, 255
writer->escaped_strings *254*
writer->escaped_uri_chars *254*, 255
writer->fancyitem *367*
writer->fancylist *366*
writer->fencedCode *372*
writer->flatten_inlines 252, *252*
writer->get_state *273*
writer->hard_line_break *254*
writer->heading *271*
writer->identifier *255*
writer->image *257*
writer->infostring *255*
writer->inline_html_cdata_section *261*

writer->inline_html_close_tag *261*
writer->inline_html_comment *259*
writer->inline_html_declaration *261*
writer->inline_html_empty_tag *261*
writer->inline_html_open_tag *261*
writer->inline_html_processing_instruction *261*
writer->inline_html_tag *259*
writer->inline_math *395*
writer->interblocksep *253*
writer->is_writing 252, *252*
writer->jekyllData *399*
writer->lineblock *383*
writer->link *257*
writer->mark *384*
writer->math *255*
writer->nbsp *253*
writer->note *386*
writer->options *252*
writer->ordereditem *259*
writer->orderedlist *258*
writer->paragraph *253*
writer->paragraphsep *253*
writer->plain *253*
writer->pop_attributes 266, *266*, 267
writer->push_attributes 266, *266*, 267
writer->rawBlock *373*
writer->rawInline *393*
writer->set_state *273*
writer->slice_begin *252*
writer->slice_end *252*
writer->soft_line_break *254*
writer->space *253*
writer->span *356*
writer->strike_through *393*
writer->string *255*
writer->strong *262*
writer->subscript *394*
writer->superscript *395*
writer->table *389*
writer->thematic_break *254*
writer->tickbox *262*
writer->undosep *253*, 353
writer->uri *255*
writer->verbatim *263*

writer->warning 199, *256*
writer.new 251, 252, *252*, 410

\yaml *166*
yaml 28, 30, 39, *163*, 165, 166, 458
\yamlBegin 28, 30, 39, *54*, 55, 56, 162, 165, 172
\yamlEnd 28, 30, 39, *54*, 55, 56, 162, 165, 166, 172
\yamlInput 28, 30, 39, *54*, 57, 163, 166, 174, 458
\yamlSetup *59*