A Markdown Interpreter for TEX

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

Version 3.7.0-0-g98dece19
2024-08-30

Contents

List of Figures

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 of the package. The technical documentation contains only a limited

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

number of tutorials and code examples. You can find more of these in the user manual.[3]

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

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

1.1 Requirements

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

1.1.1 Lua Requirements

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

LPeg $\geqslant$ 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg $\geqslant$ 0.10 is included in LuaTEX $\geqslant$ 0.72.0 (TEX Live $\geqslant$ 2013).

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

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

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

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaTEX (TEX Live $\geqslant$ 2008).

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

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

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

```
(function()
```

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

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

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

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

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

api7/lua-tinyyaml A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jekyllData` option is enabled. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

```
# hard lua-tinyyaml  # TODO: Uncomment after TeX Live 2022 deprecation.
```

1.1.2 Plain TEX Requirements

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

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

```
hard l3kernel
\unprotect
```

```
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic
\fi
```

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

```
hard lt3luabridge
```

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

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

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

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

Unless you convert markdown documents to TEX manually using the Lua command-line interface (see Section 2.1.7), the plain TEX part of the package will require that either the LuaTEX `\directlua` primitive or the shell access file stream 18 is available in your TEX engine. If only the shell access file stream is available in your TEX engine (as is the case with pdfTEX and XƎTEX), 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 2ε format is loaded,

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

a TEX engine that extends ε-TEX, and all the plain TEX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or LATEX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
soft url
```

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

```
soft graphics
```

paralist A package that provides the `compactitem`, `compactenum`, and `compactdesc` macros for the typesetting of tight bulleted lists, ordered lists, and definition lists as well as the rendering of fancy lists.

```
soft paralist
```

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` LaTeX theme (see Section 2.3.3).

```
soft latex
soft epstopdf-pkg  # required by `latex`
```

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`
```

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

```
soft gobble
```

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

```
soft amsmath
soft amsfonts
```

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` LaTeX theme, see Section 2.3.3.

```
soft catchfile
```

grffile A package that extends the name processing of the graphics package to support a larger range of file names in 2006 ⩽ TEX Live ⩽ 2019. Since TEX Live ⩾ 2020, the functionality of the package has been integrated in the LATEX 2_ε kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` LATEX themes, see Section 2.3.3.

```
49 soft grffile
```

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.8, and also in the default renderer prototype for identifier attributes.

```
50 soft etoolbox
```

soulutf8 A package that is used in the default renderer prototype for strike-throughs and marked text.

```
51 soft soul
```

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.

```
52 soft ltxcmds
```

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

```
53 soft verse
```

1.1.4 ConTEXt Prerequisites

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

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

1.2 Feedback

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

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

1.3 Acknowledgements

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

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

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

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

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

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

```
54 local M = {metadata = metadata}
```

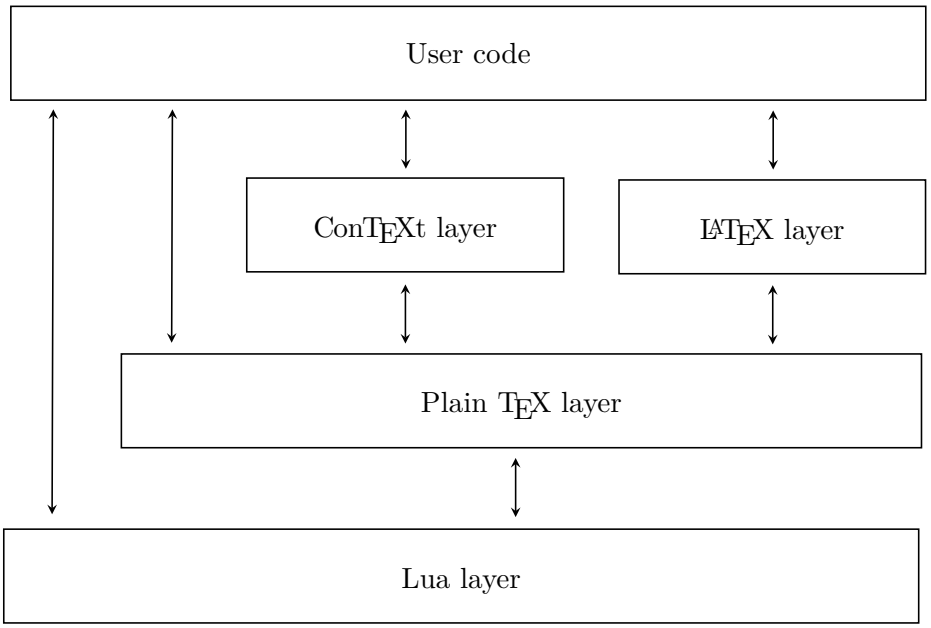

Figure 1: A block diagram of the Markdown package

</lua,lua-loader> <*lua>

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` table.

```
local defaultOptions = {}
```

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

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

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

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

```
    \prop_gput:cnn
      { g_@@_default_ #1 _options_prop }
      { #2 }
      { #4 }
    \@@_typecheck_option:n
      { #2 }
  }
\cs_generate_variant:Nn
  \@@_add_option:nnnn
  { Vnnn }
\tl_const:Nn \c_@@_option_value_true_tl  { true  }
\tl_const:Nn \c_@@_option_value_false_tl { false }
\cs_new:Nn \@@_typecheck_option:n
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_case_e:Vn
      \l_tmpa_tl
      {
        { \c_@@_option_type_boolean_tl }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \bool_if:nF
              {
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_true_tl ||
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_false_tl
              }
              {
                \msg_error:nnnV
                  { markdown }
                  { failed-typecheck-for-boolean-option }
                  { #1 }
                  \l_tmpa_tl
              }
          }
      }
  }
\msg_new:nnn
  { markdown }
  { failed-typecheck-for-boolean-option }
```

```
  {
    Option~#1~has~value~#2,~
    but~a~boolean~(true~or~false)~was~expected.
  }
\cs_generate_variant:Nn
  \str_case_e:nn
  { Vn }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnnV }
\seq_new:N
  \g_@@_option_types_seq
\tl_const:Nn
  \c_@@_option_type_clist_tl
  { clist }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_clist_tl
\tl_const:Nn
  \c_@@_option_type_counter_tl
  { counter }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_counter_tl
\tl_const:Nn
  \c_@@_option_type_boolean_tl
  { boolean }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_boolean_tl
\tl_const:Nn
  \c_@@_option_type_number_tl
  { number }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_number_tl
\tl_const:Nn
  \c_@@_option_type_path_tl
  { path }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_path_tl
\tl_const:Nn
  \c_@@_option_type_slice_tl
  { slice }
\seq_gput_right:NV
  \g_@@_option_types_seq
```

```
  \c_@@_option_type_slice_tl
\tl_const:Nn
  \c_@@_option_type_string_tl
  { string }
\seq_gput_right:NV
  \g_@@_option_types_seq
  \c_@@_option_type_string_tl
\cs_new:Nn
  \@@_get_option_type:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_ ##1 _option_types_prop }
          { #1 }
          \l_tmpa_tl
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { markdown }
          { undefined-option }
          { #1 }
      }
    \seq_if_in:NVF
      \g_@@_option_types_seq
      \l_tmpa_tl
      {
        \msg_error:nnnV
          { markdown }
          { unknown-option-type }
          { #1 }
          \l_tmpa_tl
      }
    \tl_set_eq:NN
      #2
      \l_tmpa_tl
  }
```

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

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

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

```
\cs_new:Nn \@@_with_various_cases:nn
  {
    \seq_clear:N
      \l_tmpa_seq
    \seq_map_inline:Nn
      \g_@@_cases_seq
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \use:c { ##1 }
          \l_tmpa_tl
        \seq_put_right:NV
          \l_tmpa_seq
          \l_tmpa_tl
      }
    \seq_map_inline:Nn
      \l_tmpa_seq
      { #2 }
  }
```

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

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

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

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

```
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }
```

2.1.4 General Behavior

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

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

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

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

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

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

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

`true` Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time-

and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions.

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

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

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

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

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

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

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

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

[6] See https://github.com/witiko/markdown/pull/226#issuecomment-1599641634.
[7] 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.

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

`contentBlocksLanguageMap`=⟨*filename*⟩
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.5.9 for more information.

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

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

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

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

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

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

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

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

2.1.6 Parser Options

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

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

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

# 123 Sesame Street
```

`false` Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

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

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

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

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

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

`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
```

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

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

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

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

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

`blankBeforeHeading`=`true`, `false` default: `false`

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

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc bracketed span syntax extension.

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

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

`true` A blank line separates block quotes.

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

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

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

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

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

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

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

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

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

```
Here is a simple parenthetical citation [@doe99] and here
is a string of several [see @doe99, pp. 33-35; also
@smith04, chap. 1].

A parenthetical citation can have a [prenote @doe99] and
a [@smith04 postnote]. The name of the author can be
suppressed by inserting a dash before the name of an
author as follows [-@smith04].

Here is a simple text citation @doe99 and here is
a string of several @doe99 [pp. 33-35; also @smith04,
chap. 1]. Here is one with the name of the author
suppressed -@doe99.
```

`false` Disable the Pandoc citation syntax extension.

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

`codeSpans`=true, false default: `true`

`true` Enable the code span syntax:

```
Use the `printf()` function.
``There is a literal backtick (`) here.``
```

`false` Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

```
``This is a quote.''
```

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

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

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

true

: Enable the iA Writer content blocks syntax extension [3]:

```
``` md
http://example.com/minard.jpg (Napoleon's
  disastrous Russian campaign of 1812)
/Flowchart.png "Engineering Flowchart"
/Savings Account.csv 'Recent Transactions'
/Example.swift
/Lorem Ipsum.txt
``````
```

`false` Disable the iA Writer content blocks syntax extension.

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

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

`block` Treat content as a sequence of blocks.

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

`inline` Treat all content as inline content.

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

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

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

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

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

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

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

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

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

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

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

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

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

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

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

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

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

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

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

`extensions`=⟨*filenames*⟩

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

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

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

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

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

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

return strike_through
```

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc fancy list syntax extension.

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

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

`true` Enable the commonmark fenced code block extension:

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

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

`false` Disable the commonmark fenced code block extension.

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

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

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

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

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

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

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

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

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

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

And another.
:::::
```

`false` Disable the Pandoc fenced div syntax extension.

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

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

`finalizeCache`=`true`, `false` default: `false`

Whether an output file specified with the `frozenCacheFileName` option (frozen cache) that contains a mapping between an enumeration of markdown documents and their auxiliary cache files will be created.

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

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

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

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

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

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

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

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

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

# 123 Sesame Street
```

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

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

See also the option autoIdentifiers.

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

hashEnumerators=true, false default: false

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

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

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

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

headerAttributes=true, false default: false

true Enable the assignment of HTML attributes to headings:

```
# My first heading {#foo}

## My second heading ##    {#bar .baz}

Yet another heading   {key=value}
==================
```

false Disable the assignment of HTML attributes to headings.

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

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

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

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

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

`hybrid`=`true`, `false` default: `false`

`true` Disable the escaping of special plain TEX characters, which makes it possible to intersperse your markdown markup with TEX code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix TEX and markdown markup freely.

`false` Enable the escaping of special plain TEX characters outside verbatim environments, so that they are not interpreted by TEX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

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

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc inline note syntax extension.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc line block syntax extension.

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

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

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

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

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

`false` Disable the Pandoc mark syntax extension.

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

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

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

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

[^1]: Here is the note.

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

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

        { some.code }

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

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

`false` Disable the Pandoc note syntax extension.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

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

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

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

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

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

`false` Disable relative references in autolinks.

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

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

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

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

```
\@@_add_lua_option:nnn
  { shiftHeadings }
  { number }
  { 0 }
defaultOptions.shiftHeadings = 0
```

`slice`=⟨*the beginning and the end of a slice*⟩ default: `^ $`

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

- The circumflex (`^`) selects the beginning of a document.
- The dollar sign (`$`) selects the end of a document.
- `^`⟨*identifier*⟩ selects the beginning of a section (see the `headerAttributes` option) or a fenced div (see the `fencedDivs` option) with the HTML attribute `#`⟨*identifier*⟩.
- `$`⟨*identifier*⟩ selects the end of a section with the HTML attribute `#`⟨*identifier*⟩.
- ⟨*identifier*⟩ corresponds to `^`⟨*identifier*⟩ for the first selector and to `$`⟨*identifier*⟩ for the second selector.

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

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

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

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

`false` Preserve all ellipses in the input.

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

`startNumber`=`true`, `false` default: `true`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`false` Disable the Pandoc subscript syntax extension.

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

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

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

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

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

`false` Disable the Pandoc superscript syntax extension.

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

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

true

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

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

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

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

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

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

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

true

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

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

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

`false` Disable the Pandoc table caption syntax extension.

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

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

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

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

`false` Disable the Pandoc task list syntax extension.

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

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

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

`true` Strip TeX-style comments.

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

Always enabled when `hybrid` is enabled.

`false` Do not strip TeX-style comments.

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

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

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

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

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

`false` Disable the Pandoc dollar math syntax extension.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- This is

  not a tight

- unordered list.
```

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

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

`underscores`=`true`, `false` default: `true`

`true` Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```
*single asterisks*
_single underscores_
**double asterisks**
__double underscores__
```

`false` Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the `hybrid` option without the need to constantly escape subscripts.

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

2.1.7 Command-Line Interface

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

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

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

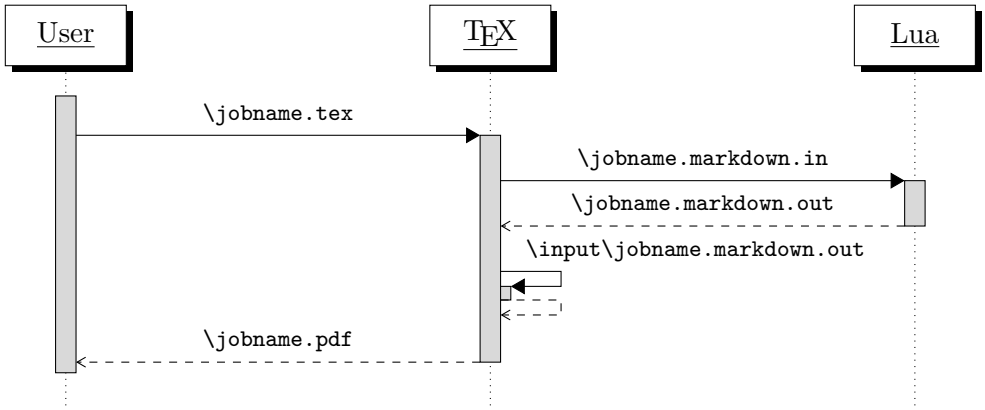

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
729
730 local HELP_STRING = [[
731 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
732 where OPTIONS are documented in the Lua interface section of the
733 technical Markdown package documentation.
734
735 When OUTPUT_FILE is unspecified, the result of the conversion will be
736 written to the standard output. When INPUT_FILE is also unspecified, the
737 result of the conversion will be read from the standard input.
738
739 Report bugs to: witiko@mail.muni.cz
740 Markdown package home page: <https://github.com/witiko/markdown>]]
741
```

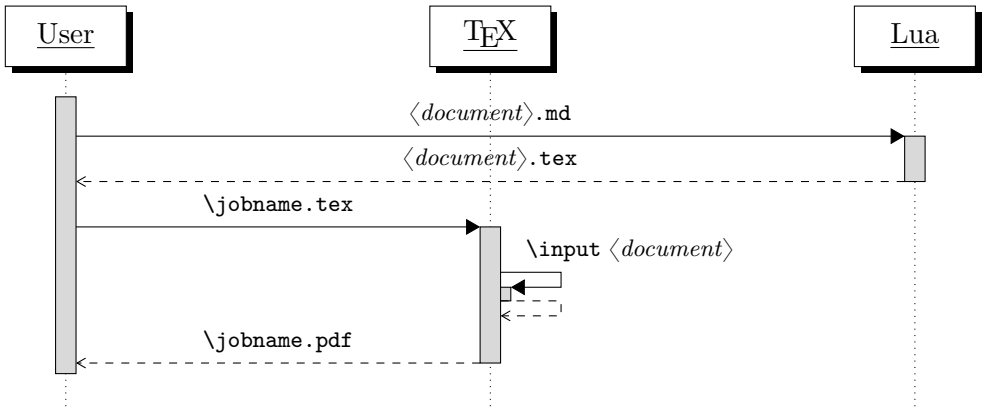

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
local VERSION_STRING = [[
markdown-cli.lua (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 [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake_case in addition to camelCase variants of options. As a bonus, studies [5] 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)
```

```
767     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
768   end)
769   return cased_option_name
770 end
771
772 local cases = {camel_case, snake_case}
773 local various_case_options = {}
774 for option_name, _ in pairs(defaultOptions) do
775   for _, case in ipairs(cases) do
776     various_case_options[case(option_name)] = option_name
777   end
778 end
779
780 local process_options = true
781 local options = {}
782 local input_filename
783 local output_filename
784 for i = 1, #arg do
785   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.

```
786     if arg[i] == "--" then
787       process_options = false
788       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.

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

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

```
795       local default_type = type(defaultOptions[key])
796       if default_type == "boolean" then
797         options[key] = (value == "true")
798       elseif default_type == "number" then
799         options[key] = tonumber(value)
800       elseif default_type == "table" then
801         options[key] = {}
802         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.

```
  elseif output_filename == nil then
    output_filename = arg[i]
  else
    error('Unexpected argument: "' .. arg[i] .. '".')
  end
  ::continue::
```

```
835 end
```

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

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

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

```
\input hello
```

2.2 Plain TEX Interface

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

```
836 \def\markdownLastModified{(((LASTMODIFIED)))}%
837 \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

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

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

```
838 \let\markdownBegin\relax
839 \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 [6, 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
```

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

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

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

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

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

```
\input markdown
\markdownSetup{contentLevel=inline}
```

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

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

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
```

This macro 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
```

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

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

2.2.2 Options

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

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

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

```
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_plain_tex_tl
  }
```

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

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

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

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

The plain TEX options may be also be specified via the `\markdownSetup` macro. Here, the plain TEX options are represented by a comma-delimited list of ⟨*key*⟩`=`⟨*value*⟩ pairs. For boolean options, the `=`⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩`=true` if the `=`⟨*value*⟩ part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

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

The `\markdownIfOption{`⟨*name*⟩`}{`⟨*iftrue*⟩`}{`⟨*iffalse*⟩`}` macro is provided for testing, whether the value of `\markdownOption`⟨*name*⟩ is `true`. If the value is `true`, then ⟨*iftrue*⟩ is expanded, otherwise ⟨*iffalse*⟩ is expanded.

```
\prg_new_conditional:Nnn
  \@@_if_option:n
  { TF, T, F }
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \msg_error:nnxx
          { markdown }
          { expected-boolean-option }
          { #1 }
          { \l_tmpa_tl }
      }
    \@@_get_option_value:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNTF
      \l_tmpa_tl
      \c_@@_option_value_true_tl
      { \prg_return_true: }
      { \prg_return_false: }
  }
\msg_new:nnn
  { markdown }
  { expected-boolean-option }
  {
    Option~#1~has~type~#2,~
    but~a~boolean~was~expected.
  }
\let\markdownIfOption=\@@_if_option:nTF
```

2.2.2.1 Finalizing and Freezing the Cache

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

The `\markdownOptionFrozenCache` option uses the mapping previously created by the `finalizeCache` option, and uses it to typeset the plain TEX document without invoking Lua. As a result, the plain TEX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

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

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

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `finalizeCache` option.
4. Typeset the plain TEX document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain TEX document and the `cacheDir` directory.

2.2.2.2 File and Directory Names

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

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

```
\@@_add_plain_tex_option:nnn
  { inputTempFileName }
  { path }
  { \jobname.markdown.in }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain TEX implementation. The option defaults to `.` or, since TEX Live 2024, to the value of the `-output-directory` option of your TEX engine.

The path must be set to the same value as the `-output-directory` option of your TEX engine for the package to function correctly. We need this macro to make the Lua implementation aware where it should store the helper files. The same limitations apply here as in the case of the `inputTempFileName` macro.

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

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

2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra

resources, especially with higher-level TEX formats such as LATEX and ConTEXt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (`%`) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing TEX package documentation using the Doc LATEX package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

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

2.2.2.5 Generating Plain TEX Option Macros and Key-Values

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

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

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

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

```
\cs_new:Nn
  \@@_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
```

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

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

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

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

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

Do not override options defined before loading the package.

```
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_if_exist:cF
          { \l_tmpa_tl }
          {
            \@@_get_default_option_value:nN
              { #1 }
              \l_tmpa_tl
            \@@_set_option_value:nV
              { #1 }
```

```
              \l_tmpa_tl
          }
      }
  }
\ExplSyntaxOff
\input lt3luabridge.tex
\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:
            )
          }
          {
            \luabridge_tl_set:Nn
              \l_tmpa_tl
              { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
            \tl_gset:NV
              \markdownOptionOutputDir
              \l_tmpa_tl
          }
          {
            \tl_gset:Nn
              \markdownOptionOutputDir
              { . }
          }
      }
  }
\cs_new:Nn
  \@@_set_option_value:nn
  {
    \@@_define_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
```

```
      \l_tmpa_tl
    \str_if_eq:NNTF
      \c_@@_option_type_counter_tl
      \l_tmpa_tl
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \int_gset:cn
          { \l_tmpa_tl }
          { #2 }
      }
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \cs_set:cpn
          { \l_tmpa_tl }
          { #2 }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\cs_new:Nn
  \@@_define_option:n
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cT
      { \l_tmpa_tl }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpb_tl
        \str_if_eq:NNT
          \c_@@_option_type_counter_tl
          \l_tmpb_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \int_new:c
              { \l_tmpa_tl }
          }
      }
```

```
  }
\cs_new:Nn
  \@@_define_option_keyval:nnn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \str_if_eq:VVTF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/options }
          {
```

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

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

```
    }
```

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

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

```
                { #2 }
                \l_tmpa_clist
            }
          }
        \keys_define:nV
          { markdown/options }
          \l_tmpa_tl
      }
  }
\cs_generate_variant:Nn
  \clist_set:Nn
  { NV }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { F }
\msg_new:nnn
  { markdown }
  { malformed-name-for-clist-option }
  {
    Clist~option~name~#1~does~not~end~with~-s.
  }
```

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

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

2.2.3 Themes

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

The key-values `theme`=⟨*theme name*⟩ and `import`=⟨*theme name*⟩ 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 is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer LaTeX package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

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`.

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

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

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

To keep track of the current theme when themes are nested, we will maintain the `\g_@@_themes_seq` stack of theme names. For convenience, the name of the current theme is also available in the `\g_@@_current_theme_tl` macro.

```
\seq_new:N
  \g_@@_themes_seq
\tl_new:N
  \g_@@_current_theme_tl
\tl_gset:Nn
  \g_@@_current_theme_tl
  { }
\seq_gput_right:NV
  \g_@@_themes_seq
  \g_@@_current_theme_tl
\cs_new:Nn
  \@@_set_theme:n
  {
```

First, we validate the theme name.

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

Next, we munge the theme name.

```
    \str_set:Nn
      \l_tmpa_str
      { #1 }
    \str_replace_all:Nnn
      \l_tmpa_str
      { / }
      { _ }
```

Finally, we load the theme.

```
    \tl_gset:Nn
      \g_@@_current_theme_tl
      { #1 / }
    \seq_gput_right:NV
```

```
      \g_@@_themes_seq
      \g_@@_current_theme_tl
    \@@_load_theme:nV
      { #1 }
      \l_tmpa_str
    \seq_gpop_right:NN
      \g_@@_themes_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_themes_seq
      \l_tmpa_tl
    \tl_gset:NV
      \g_@@_current_theme_tl
      \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 }
\ExplSyntaxOff
```

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

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

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

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

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

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

2.2.4 Snippets

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

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

```
      \l_tmpa_tl
      \l_tmpb_tl
  }
\cs_gset_eq:NN
  \markdownSetupSnippet
  \@@_setup_snippet:nn
\msg_new:nnnn
  { markdown }
  { empty-snippet-name }
  { Empty~snippet~name~#1 }
  { Pick~a~non-empty~name~for~your~snippet }
\msg_new:nnn
  { markdown }
  { redefined-snippet }
  { Redefined~snippet~#1 }
```

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

```
\prg_new_conditional:Nnn
  \@@_if_snippet_exists:n
  { TF, T, F }
  {
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_put_right:Nn
      \l_tmpa_tl
      { #1 }
    \prop_get:NVNTF
      \g_@@_snippets_prop
      \l_tmpa_tl
      \l_tmpb_tl
      { \prg_return_true: }
      { \prg_return_false: }
  }
\cs_gset_eq:NN
  \markdownIfSnippetExists
  \@@_if_snippet_exists:nTF
```

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

```
\keys_define:nn
  { markdown/options }
  {
    snippet .code:n = {
      \tl_set:NV
        \l_tmpa_tl
        \g_@@_current_theme_tl
      \tl_put_right:Nn
```

```
        \l_tmpa_tl
        { #1 }
      \@@_if_snippet_exists:nTF
        { #1 }
        {
          \prop_get:NVN
            \g_@@_snippets_prop
            \l_tmpa_tl
            \l_tmpb_tl
          \tl_use:N
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { undefined-snippet }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { undefined-snippet }
  { Can't~invoke~undefined~snippet~#1 }
\ExplSyntaxOff
```

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

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

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
```

```
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

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

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

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

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

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

Here, we import the snippets.

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

```
            \l_tmpb_tl
            { snippet = { \l_tmpa_tl } }
        }
```

Here, we load the theme.

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

2.2.5 Token Renderers

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

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

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

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

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

2.2.5.1 Attribute Renderers

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

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

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

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

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

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

2.2.5.2 Block Quote Renderers

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

```
\def\markdownRendererBlockQuoteBegin{%
  \markdownRendererBlockQuoteBeginPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.3 Bracketed Spans Attribute Context Renderers

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

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

```
\def\markdownRendererBracketedSpanAttributeContextBegin{%
  \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
```

```
\def\markdownRendererBracketedSpanAttributeContextEnd{%
  \markdownRendererBracketedSpanAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.4 Bullet List Renderers

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

```
\def\markdownRendererUlBegin{%
  \markdownRendererUlBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererUlBeginTight{%
  \markdownRendererUlBeginTightPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererUlItem{%
  \markdownRendererUlItemPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererUlItemEnd{%
  \markdownRendererUlItemEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererUlEnd{%
  \markdownRendererUlEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererUlEndTight{%
```

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

2.2.5.5 Citation Renderers

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

```
\def\markdownRendererCite{%
  \markdownRendererCitePrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.6 Code Block Renderers

The \markdownRendererInputVerbatim macro represents a code block. The macro receives a single argument that corresponds to the filename of a file containing the code block contents.

```
\def\markdownRendererInputVerbatim{%
  \markdownRendererInputVerbatimPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.7 Code Span Renderer

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

```
\def\markdownRendererCodeSpan{%
  \markdownRendererCodeSpanPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpan }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpan }
```

```
  { 1 }
\ExplSyntaxOff
```

2.2.5.8 Code Span Attribute Context Renderers

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

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

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

2.2.5.9 Content Block Renderers

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

```
\def\markdownRendererContentBlock{%
  \markdownRendererContentBlockPrototype}%
\ExplSyntaxOn
\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`.

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

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

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

```
\def\markdownRendererContentBlockCode{%
  \markdownRendererContentBlockCodePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

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

```
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

2.2.5.10 Definition List Renderers

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

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

```
\def\markdownRendererDlBegin{%
  \markdownRendererDlBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlBeginTight{%
  \markdownRendererDlBeginTightPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlItem{%
  \markdownRendererDlItemPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlItemEnd{%
  \markdownRendererDlItemEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlDefinitionBegin{%
  \markdownRendererDlDefinitionBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlDefinitionEnd{%
  \markdownRendererDlDefinitionEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererDlEnd{%
  \markdownRendererDlEndPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.11 Ellipsis Renderer

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

```
\def\markdownRendererEllipsis{%
  \markdownRendererEllipsisPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ellipsis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

```
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

2.2.5.12 Emphasis Renderers

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

```
\def\markdownRendererEmphasis{%
  \markdownRendererEmphasisPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```
\def\markdownRendererFencedCodeAttributeContextBegin{%
  \markdownRendererFencedCodeAttributeContextBeginPrototype}%
\ExplSyntaxOn
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedCodeAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererFencedCodeAttributeContextEnd{%
  \markdownRendererFencedCodeAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedCodeAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.14 Fenced Div Attribute Context Renderers

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

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

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

```
\ExplSyntaxOff
```

2.2.5.15 Header Attribute Context Renderers

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

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

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

2.2.5.16 Heading Renderers

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

```
\def\markdownRendererHeadingOne{%
  \markdownRendererHeadingOnePrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererHeadingTwo{%
  \markdownRendererHeadingTwoPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererHeadingThree{%
  \markdownRendererHeadingThreePrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererHeadingFour{%
  \markdownRendererHeadingFourPrototype}%
\ExplSyntaxOn
\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.

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

```
\ExplSyntaxOn
\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.

```
\def\markdownRendererHeadingSix{%
  \markdownRendererHeadingSixPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingSix }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingSix }
  { 1 }
\ExplSyntaxOff
```

2.2.5.17 Inline HTML Comment Renderer

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

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

2.2.5.18 HTML Tag and Element Renderers

The \markdownRendererInlineHtmlTag macro represents an opening, closing, or empty inline HTML tag. This macro will only be produced, when the html option is

enabled. The macro receives a single argument that corresponds to the contents of the HTML tag.

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

```
\def\markdownRendererInlineHtmlTag{%
  \markdownRendererInlineHtmlTagPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlTag }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererInputBlockHtmlElement{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlElement }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlElement }
  { 1 }
\ExplSyntaxOff
```

2.2.5.19 Image Renderer

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

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

2.2.5.20 Image Attribute Context Renderers

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

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

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

2.2.5.21 Interblock Separator Renderers

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

```
\def\markdownRendererInterblockSeparator{%
  \markdownRendererInterblockSeparatorPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.22 Line Block Renderers

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

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

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

2.2.5.23 Line Break Renderers

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

```
\def\markdownRendererSoftLineBreak{%
  \markdownRendererSoftLineBreakPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.24 Link Renderer

The \markdownRendererLink macro represents a hyperlink. It receives four arguments: the label, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the link.

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

2.2.5.25 Link Attribute Context Renderers

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

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

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

2.2.5.26 Marked Text Renderer

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

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

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

2.2.5.27 Markdown Document Renderers

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

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

```
\def\markdownRendererDocumentBegin{%
  \markdownRendererDocumentBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDocumentEnd{%
  \markdownRendererDocumentEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.28 Non-Breaking Space Renderer

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

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

2.2.5.29 Note Renderer

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

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

2.2.5.30 Ordered List Renderers

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

```
\def\markdownRendererOlBegin{%
  \markdownRendererOlBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlBeginTight{%
  \markdownRendererOlBeginTightPrototype}%
\ExplSyntaxOn
\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`).

```
\def\markdownRendererFancyOlBegin{%
  \markdownRendererFancyOlBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererFancyOlBeginTight{%
  \markdownRendererFancyOlBeginTightPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlItem{%
```

```
  \markdownRendererOlItemPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlItemEnd{%
  \markdownRendererOlItemEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlItemWithNumber{%
  \markdownRendererOlItemWithNumberPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererFancyOlItem{%
```

```
  \markdownRendererFancyOlItemPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererFancyOlItemEnd{%
  \markdownRendererFancyOlItemEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererFancyOlItemWithNumber{%
  \markdownRendererFancyOlItemWithNumberPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlEnd{%
  \markdownRendererOlEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererOlEndTight{%
  \markdownRendererOlEndTightPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererFancyOlEnd{%
  \markdownRendererFancyOlEndPrototype}%
\ExplSyntaxOn
\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.

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

2.2.5.31 Raw Content Renderers

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

```
\def\markdownRendererInputRawInline{%
  \markdownRendererInputRawInlinePrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererInputRawBlock{%
  \markdownRendererInputRawBlockPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputRawBlock }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputRawBlock }
  { 2 }
```

```
\ExplSyntaxOff
```

2.2.5.32 Section Renderers

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

```
\def\markdownRendererSectionBegin{%
  \markdownRendererSectionBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { sectionBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { sectionBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererSectionEnd{%
  \markdownRendererSectionEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { sectionEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { sectionEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.33 Replacement Character Renderers

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

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

2.2.5.34 Special Character Renderers

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

```
\def\markdownRendererLeftBrace{%
  \markdownRendererLeftBracePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { leftBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { leftBrace }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererRightBrace{%
  \markdownRendererRightBracePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { rightBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { rightBrace }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDollarSign{%
  \markdownRendererDollarSignPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dollarSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dollarSign }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPercentSign{%
  \markdownRendererPercentSignPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { percentSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { percentSign }
  { 0 }
```

```
\ExplSyntaxOff
\def\markdownRendererAmpersand{%
  \markdownRendererAmpersandPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ampersand }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ampersand }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererUnderscore{%
  \markdownRendererUnderscorePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { underscore }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { underscore }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHash{%
  \markdownRendererHashPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { hash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { hash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererCircumflex{%
  \markdownRendererCircumflexPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { circumflex }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { circumflex }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBackslash{%
  \markdownRendererBackslashPrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { backslash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { backslash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererTilde{%
  \markdownRendererTildePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tilde }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tilde }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPipe{%
  \markdownRendererPipePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { pipe }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { pipe }
  { 0 }
\ExplSyntaxOff
```

2.2.5.35 Strike-Through Renderer

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

```
\def\markdownRendererStrikeThrough{%
  \markdownRendererStrikeThroughPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { strikeThrough }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { strikeThrough }
```

```
  { 1 }
\ExplSyntaxOff
```

2.2.5.36 Subscript Renderer

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

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

2.2.5.37 Superscript Renderer

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

```
\def\markdownRendererSuperscript{%
  \markdownRendererSuperscriptPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { superscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { superscript }
  { 1 }
\ExplSyntaxOff
```

2.2.5.38 Table Attribute Context Renderers

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

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

```
\def\markdownRendererTableAttributeContextBegin{%
  \markdownRendererTableAttributeContextBeginPrototype}%
\ExplSyntaxOn
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererTableAttributeContextEnd{%
  \markdownRendererTableAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tableAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tableAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.5.39 Table Renderer

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

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

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

2.2.5.40 TEX Math Renderers

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

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

2.2.5.41 Thematic Break Renderer

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

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

2.2.5.42 Tickbox Renderers

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

```
\def\markdownRendererTickedBox{%
  \markdownRendererTickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tickedBox }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHalfTickedBox{%
  \markdownRendererHalfTickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { halfTickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { halfTickedBox }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererUntickedBox{%
  \markdownRendererUntickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { untickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { untickedBox }
  { 0 }
\ExplSyntaxOff
```

2.2.5.43 Warning and Error Renderers

The \markdownRendererWarning and \markdownRendererError macros represent warnings and errors produced by the markdown parser. Both macros receive a single parameter with the text of the warning or error.

```
\def\markdownRendererWarning{%
  \markdownRendererWarningPrototype}%
```

```
\def\markdownRendererError{%
  \markdownRendererErrorPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { warning }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { warning }
  { 1 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { error }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { error }
  { 1 }
\ExplSyntaxOff
```

2.2.5.44 YAML Metadata Renderers

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

```
\def\markdownRendererJekyllDataBegin{%
  \markdownRendererJekyllDataBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataEnd{%
  \markdownRendererJekyllDataEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataMappingBegin{%
  \markdownRendererJekyllDataMappingBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataMappingEnd{%
  \markdownRendererJekyllDataMappingEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataSequenceBegin{%
  \markdownRendererJekyllDataSequenceBeginPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataSequenceEnd{%
  \markdownRendererJekyllDataSequenceEndPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataBoolean{%
  \markdownRendererJekyllDataBooleanPrototype}%
\ExplSyntaxOn
\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.

```
\def\markdownRendererJekyllDataNumber{%
  \markdownRendererJekyllDataNumberPrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataNumber }
  { 2 }
\ExplSyntaxOff
```

The \markdownRendererJekyllDataTypographicString and \markdownRendererJekyllDataP macros represent string scalar values in a YAML document. This macro will only be produced when the jekyllData option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

For each string scalar value, both macros are produced. Whereas \markdownRendererJekyllDataT receives the scalar value after all markdown markup and special TEX characters in the string have been replaced by TEX macros, \markdownRendererJekyllDataProgrammaticString receives the raw scalar value. Therefore, whereas the \markdownRendererJekyllDataTypographicSt macro is more appropriate for texts that are supposed to be typeset with TEX, such as document titles, author names, or exam questions, the \markdownRendererJekyllDataProgrammaticString macro is more appropriate for identifiers and other programmatic text that won't be typeset by TEX.

```
\def\markdownRendererJekyllDataTypographicString{%
  \markdownRendererJekyllDataTypographicStringPrototype}%
\def\markdownRendererJekyllDataProgrammaticString{%
  \markdownRendererJekyllDataProgrammaticStringPrototype}%
\ExplSyntaxOn
\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 \markdownRendererJekyllDataI

macro was not produced. The \markdownRendererJekyllDataString has been deprecated and will be removed in Markdown 4.0.0.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererJekyllDataTypographicString
  {
    \cs_if_exist:NTF
      \markdownRendererJekyllDataString
      {
        \markdownWarning
          {
            The~jekyllDataString~renderer~has~been~deprecated,~
            to~be~removed~in~Markdown~4.0.0
          }
        \markdownRendererJekyllDataString
      }
      {
        \cs_if_exist:NTF
          \markdownRendererJekyllDataStringPrototype
          {
            \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.

```
\def\markdownRendererJekyllDataEmpty{%
  \markdownRendererJekyllDataEmptyPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEmpty }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEmpty }
  { 1 }
\ExplSyntaxOff
```

2.2.5.45 Generating Plain TEX Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain TEX macros for token renderers. Furthermore, the `\markdownSetup` macro also accepts the `renderers` key, whose value 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.

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderers:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
```

The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```
        \str_if_eq:nnF
          { ##1 }
          { jekyllDataString }
          {
            \@@_define_renderer:n
              { ##1 }
          }
      }
  }
\cs_new:Nn \@@_define_renderer:n
  {
    \@@_renderer_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
```

```
      \l_tmpb_tl
    \@@_define_renderer:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
\tl_new:N
  \l_@@_renderer_definition_tl
\bool_new:N
  \g_@@_appending_renderer_bool
\cs_new:Nn \@@_define_renderer:nNn
  {
    \keys_define:nn
      { markdown/options/renderers }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_@@_renderer_definition_tl
            { ##1 }
          \regex_replace_all:nnN
            { \cP\#0 }
            { #1 }
            \l_@@_renderer_definition_tl
          \bool_if:NT
            \g_@@_appending_renderer_bool
            {
              \@@_tl_set_from_cs:NNn
                \l_tmpa_tl
                #2
                { #3 }
              \tl_put_left:NV
                \l_@@_renderer_definition_tl
                \l_tmpa_tl
            }
```

```
          \cs_generate_from_arg_count:NNnV
            #2
            \cs_set:Npn
            { #3 }
            \l_@@_renderer_definition_tl
        },
    }
}
```

We define the function `\@@_tl_set_from_cs:NNn` [9]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```
\cs_new_protected:Nn
  \@@_tl_set_from_cs:NNn
  {
    \tl_set:Nn
      \l_tmpa_tl
      { #2 }
    \int_step_inline:nn
      { #3 }
      {
        \exp_args:NNc
          \tl_put_right:Nn
          \l_tmpa_tl
          { @@_tl_set_from_cs_parameter_ ##1 }
      }
    \exp_args:NNV
      \tl_set:No
      \l_tmpb_tl
      \l_tmpa_tl
    \regex_replace_all:nnN
      { \cP. }
      { \0\0 }
      \l_tmpb_tl
    \int_step_inline:nn
      { #3 }
      {
        \regex_replace_all:nnN
          { \c { @@_tl_set_from_cs_parameter_ ##1 } }
          { \cP\# ##1 }
          \l_tmpb_tl
      }
    \tl_set:NV
      #1
      \l_tmpb_tl
  }
```

```
\cs_generate_variant:Nn
  \@@_define_renderer:nNn
  { ncV }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { NNnV }
\cs_generate_variant:Nn
  \tl_put_left:Nn
  { Nv }
\keys_define:nn
  { markdown/options }
  {
    renderers .code:n = {
      \keys_set:nn
        { markdown/options/renderers }
        { #1 }
    },
  }
```

The following example code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` token renderer macros.

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

```
\tl_new:N
  \l_@@_renderer_glob_definition_tl
\seq_new:N
  \l_@@_renderer_glob_results_seq
\regex_const:Nn
  \c_@@_appending_key_regex
  { \s*+$ }
\keys_define:nn
  { markdown/options/renderers }
  {
    unknown .code:n = {
```

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

```
\markdownSetup{
  renderers = {
```

```
    % Start with empty renderers.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        renderers = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
2968        \regex_match:NVTF
2969          \c_@@_appending_key_regex
2970          \l_keys_key_str
2971          {
2972            \bool_gset_true:N
2973              \g_@@_appending_renderer_bool
2974            \tl_set:NV
2975              \l_tmpa_tl
2976              \l_keys_key_str
2977            \regex_replace_once:NnN
2978              \c_@@_appending_key_regex
2979              { }
2980              \l_tmpa_tl
2981            \tl_set:Nx
2982              \l_tmpb_tl
2983              { { \l_tmpa_tl } = }
2984            \tl_put_right:Nn
2985              \l_tmpb_tl
2986              { { #1 } }
2987            \keys_set:nV
2988              { markdown/options/renderers }
```

```
            \l_tmpb_tl
          \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.
}
```

```
        {
          \@@_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 }
                \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/renderers }
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer }
  {
    Renderer~#1~is~undefined.
  }
\cs_generate_variant:Nn
  \@@_glob_seq:nnN
  { VnN }
\cs_generate_variant:Nn
  \cs_generate_from_arg_count:NNnn
  { cNVV }
\cs_generate_variant:Nn
  \msg_error:nnn
  { nnV }
\prg_generate_conditional_variant:Nnn
  \regex_match:Nn
  { NV }
  { TF }
\prop_new:N
  \g_@@_glob_cache_prop
\tl_new:N
  \l_@@_current_glob_tl
```

```
\cs_new:Nn
  \@@_glob_seq:nnN
  {
    \tl_set:Nn
      \l_@@_current_glob_tl
      { ^ #1 $ }
    \prop_get:NeNTF
      \g_@@_glob_cache_prop
      { #2 / \l_@@_current_glob_tl }
      \l_tmpa_clist
      {
        \seq_set_from_clist:NN
          #3
          \l_tmpa_clist
      }
      {
        \seq_clear:N
          #3
        \regex_replace_all:nnN
          { \* }
          { .* }
          \l_@@_current_glob_tl
        \regex_set:NV
          \l_tmpa_regex
          \l_@@_current_glob_tl
        \seq_map_inline:cn
          { #2 }
          {
            \regex_match:NnT
              \l_tmpa_regex
              { ##1 }
              {
                \seq_put_right:Nn
                  #3
                  { ##1 }
              }
          }
        \clist_set_from_seq:NN
          \l_tmpa_clist
          #3
        \prop_gput:NeV
          \g_@@_glob_cache_prop
          { #2 / \l_@@_current_glob_tl }
          \l_tmpa_clist
      }
  }
% TODO: Remove in TeX Live 2023.
```

```
\prg_generate_conditional_variant:Nnn
  \prop_get:NnN
  { NeN }
  { TF }
\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

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key–values from the l3keys module of the LATEX3 kernel.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  { }
\ExplSyntaxOff
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key–values without using the expl3 language.

```
\ExplSyntaxOn
\@@_with_various_cases:nn
  { jekyllDataRenderers }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_tmpa_tl
            { ##1 }
```

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

```
            \tl_replace_all:NnV
              \l_tmpa_tl
              { / }
              \c_backslash_str
            \keys_set:nV
              { markdown/options/jekyll-data-renderers }
              \l_tmpa_tl
          },
        }
    }
\keys_define:nn
  { markdown/options/jekyll-data-renderers }
  {
    unknown .code:n = {
      \tl_set_eq:NN
        \l_tmpa_tl
        \l_keys_key_str
      \tl_replace_all:NVn
        \l_tmpa_tl
        \c_backslash_str
        { / }
      \tl_put_right:Nn
        \l_tmpa_tl
        {
          .code:n = { #1 }
        }
      \keys_define:nV
        { markdown/jekyllData }
        \l_tmpa_tl
    }
  }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\ExplSyntaxOff
```

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 `rendererPrototype` key, whose value 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.

```
\ExplSyntaxOn
\cs_new:Nn \@@_define_renderer_prototypes:
  {
    \seq_map_inline:Nn
      \g_@@_renderers_seq
      {
```

The `\markdownRendererJekyllDataString` has been deprecated and will be removed in Markdown 4.0.0.

```
        \str_if_eq:nnF
          { ##1 }
          { jekyllDataString }
          {
            \@@_define_renderer_prototype:n
              { ##1 }
          }
      }
  }
\cs_new:Nn \@@_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_define_renderer_prototype:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
        Prototype
```

```
      }
  }
\tl_new:N
  \l_@@_renderer_prototype_definition_tl
\bool_new:N
  \g_@@_appending_renderer_prototype_bool
\cs_new:Nn \@@_define_renderer_prototype:nNn
  {
    \keys_define:nn
      { markdown/options/renderer-prototypes }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_@@_renderer_prototype_definition_tl
            { ##1 }
          \regex_replace_all:nnN
            { \cP\#0 }
            { #1 }
            \l_@@_renderer_prototype_definition_tl
          \bool_if:NT
            \g_@@_appending_renderer_prototype_bool
            {
              \@@_tl_set_from_cs:NNn
                \l_tmpa_tl
                #2
                { #3 }
              \tl_put_left:NV
                \l_@@_renderer_prototype_definition_tl
                \l_tmpa_tl
            }
          \cs_generate_from_arg_count:NNnV
            #2
            \cs_set:Npn
            { #3 }
            \l_@@_renderer_prototype_definition_tl
        },
      }
```

Unless the token renderer prototype macro has already been defined, we provide an empty definition.

```
    \cs_if_free:NT
      #2
      {
        \cs_generate_from_arg_count:NNnn
          #2
          \cs_set:Npn
          { #3 }
```

```
3250            { }
3251        }
3252    }
3253 \cs_generate_variant:Nn
3254    \@@_define_renderer_prototype:nNn
3255    { ncV }
```

The following example code showcases a possible configuration of the `\markdownRendererImagePrototype` 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.
  }
}
```

```
3256 \keys_define:nn
3257    { markdown/options/renderer-prototypes }
3258    {
3259      unknown .code:n = {
```

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

```
\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = {},
    attributeClassName = {},
    attributeIdentifier = {},
    % Define the processing of a single specific HTML class name.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
```

```
        rendererPrototypes = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
3260        \regex_match:NVTF
3261          \c_@@_appending_key_regex
3262          \l_keys_key_str
3263          {
3264            \bool_gset_true:N
3265              \g_@@_appending_renderer_prototype_bool
3266            \tl_set:NV
3267              \l_tmpa_tl
3268              \l_keys_key_str
3269            \regex_replace_once:NnN
3270              \c_@@_appending_key_regex
3271              { }
3272              \l_tmpa_tl
3273            \tl_set:Nx
3274              \l_tmpb_tl
3275              { { \l_tmpa_tl } = }
3276            \tl_put_right:Nn
3277              \l_tmpb_tl
3278              { { #1 } }
3279            \keys_set:nV
3280              { markdown/options/renderer-prototypes }
3281              \l_tmpb_tl
3282            \bool_gset_false:N
3283              \g_@@_appending_renderer_prototype_bool
3284          }
```

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.
}
```

```
3285        {
3286          \@@_glob_seq:VnN
3287            \l_keys_key_str
3288            { g_@@_renderers_seq }
3289            \l_@@_renderer_glob_results_seq
3290          \seq_if_empty:NTF
3291            \l_@@_renderer_glob_results_seq
3292            {
3293              \msg_error:nnV
3294                { markdown }
3295                { undefined-renderer-prototype }
3296                \l_keys_key_str
3297            }
3298            {
3299              \tl_set:Nn
3300                \l_@@_renderer_glob_definition_tl
3301                { \exp_not:n { #1 } }
3302              \seq_map_inline:Nn
3303                \l_@@_renderer_glob_results_seq
3304                {
3305                  \tl_set:Nn
3306                    \l_tmpa_tl
3307                    { { ##1 } = }
3308                  \tl_put_right:Nx
3309                    \l_tmpa_tl
3310                    { { \l_@@_renderer_glob_definition_tl } }
3311                  \keys_set:nV
3312                    { markdown/options/renderer-prototypes }
```

```
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer-prototype }
  {
    Renderer~prototype~#1~is~undefined.
  }
\@@_with_various_cases:nn
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \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` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain TEX special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
\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}}%
|endgroup
```

The macro is exposed in the interface, so that users can create their own markdown environments. Due to the way the arguments are passed to Lua, the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol).

The `code` key, which can be used to immediately expand and execute code.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/options }
  {
    code .code:n = { #1 },
  }
\ExplSyntaxOff
```

This can be especially useful in snippets.

2.3 LaTeX Interface

The LaTeX interface provides LaTeX environments for the typesetting of markdown input from within LaTeX, facilities for setting Lua, plain TEX, and LaTeX options used during the conversion from markdown to plain TEX, and facilities for changing

the way markdown tokens are rendered. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

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

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_latex_tl { latex }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_latex_tl
  }
\ExplSyntaxOff
\input markdown/markdown
```

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

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

where ⟨*options*⟩ are the LATEX interface options (see Section 2.3.2). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single LATEX theme (see Section 2.3.3) 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 2ε parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` LATEX environments, and redefines the `\markinline` and `\markdownInput` commands.

2.3.1.1 The `markdown` and `markdown*` LATEX environments

The `markdown` and `markdown*` LATEX environments are used to typeset markdown document fragments. Both LATEX environments accept LATEX interface options (see Section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown*` environment has been deprecated and will be removed in the next major version of the Markdown package.

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

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

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

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

```
\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 text and must be followed by text that has not yet been ingested by TeX's input processor. Furthermore, using the `\markdownEnd` macro 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 is 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}
```

2.3.1.2 The `\markinline` and `\markdownInput` macros

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 2.3.2) as its optional argument. These options will only influnce this markdown content.

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

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

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

2.3.2 Options

The LATEX options are represented by a comma-delimited list of ⟨*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.

2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizecache` and `frozencache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain TeX options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizecache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

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

We hope that other packages will support the `finalizecache` and `frozencache` package options in the future, so that they can become a standard interface for preparing LaTeX document sources for distribution.

```
\DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
\DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

2.3.2.2 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,
}
```

2.3.3 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, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` LaTeX package, and finally the `markdownthemewitiko_dot.sty` LaTeX package:

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

```
3388 \newif\ifmarkdownLaTeXLoaded
3389   \markdownLaTeXLoadedfalse
```

Due to limitations of LaTeX, themes may not be loaded after the beginning of a LaTeX document.

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

witiko/dot A theme that typesets fenced code blocks with the `dot …` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

```
\documentclass{article}
\usepackage[import=witiko/dot]{markdown}
\setkeys{Gin}{
  width = \columnwidth,
  height = 0.65\paperheight,
  keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathemathical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

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

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

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

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

```
3390 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%
```

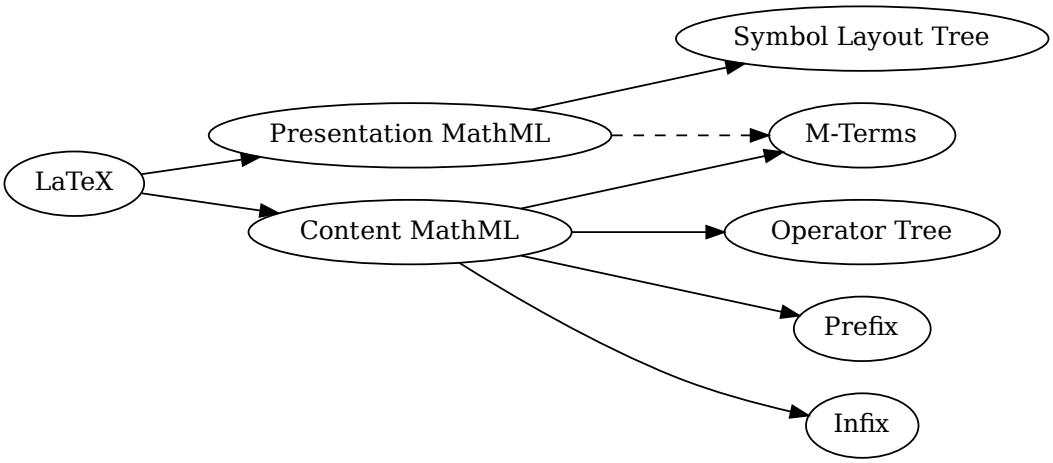

Figure 4: Various formats of mathemathical formulae

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 5. 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.

```
\ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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
```

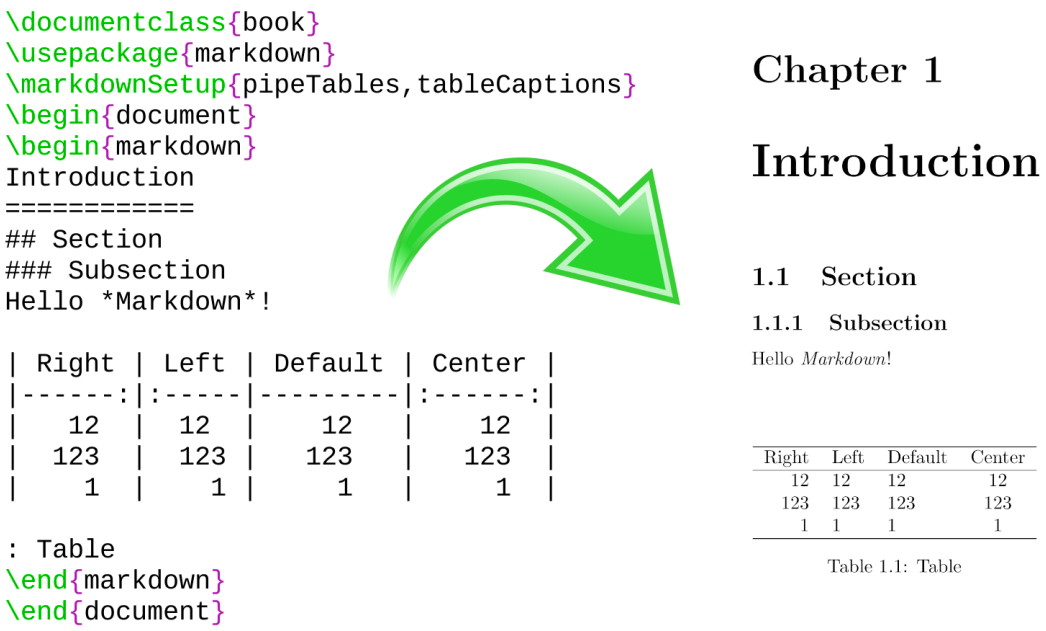

Figure 5: The banner of the Markdown package

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).

```
  \markdownIfOption{noDefaults}{}{
    \markdownSetup{theme=witiko/markdown/defaults}
  }
}
\ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%
```

Please, see Section 3.3.3 for implementation details of the built-in LaTeX themes.

2.4 ConTeXt Interface

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

```
\ExplSyntaxOn
\tl_const:Nn \c_@@_option_layer_context_tl { context }
\cs_generate_variant:Nn
  \tl_const:Nn
  { NV }
\tl_if_exist:NF
```

```
  \c_@@_top_layer_tl
  {
    \tl_const:NV
      \c_@@_top_layer_tl
      \c_@@_option_layer_context_tl
  }
\ExplSyntaxOff
```

The ConTEXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTEXt and facilities for setting Lua, plain TEX, and ConTEXt options used during the conversion from markdown to plain TEX. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

```
\writestatus{loading}{ConTeXt User Module / markdown}%
\startmodule[markdown]
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\input markdown/markdown
```

The ConTEXt interface is implemented by the `t-markdown.tex` ConTEXt module file that can be loaded as follows:

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

It is expected that the special plain TEX characters have the expected category codes, when `\input`ting the file.

2.4.1 Typesetting Markdown

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

```
\let\startmarkdown\relax
\let\stopmarkdown\relax
\let\inputmarkdown\relax
```

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

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

The following example ConTEXt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```
\usemodule[t][markdown]
\starttext
\startmarkdown
```

```
_Hello_ **world** ...
\stopmarkdown
\stoptext
```

The `\inputmarkdown` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TEX. Unlike the `\markdownInput` macro provided by the plain TEX interface, this macro also accepts ConTEXt interface options (see Section 2.4.2) as its optional argument. These options will only influnce this markdown document.

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

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

2.4.2 Options

The ConTEXt options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` (or, equivalently, ⟨*key*⟩=`yes`) if the =⟨*value*⟩ part has been omitted.

ConTEXt options map directly to the options recognized by the plain TEX interface (see Section 2.2.2).

The ConTEXt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

```
\ExplSyntaxOn
\cs_new:Npn
  \setupmarkdown
  [ #1 ]
  {
    \@@_setup:n
      { #1 }
  }
\ExplSyntaxOff
```

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.

```
\ExplSyntaxOn
```

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

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

```
\str_if_eq:VVT
  \c_@@_top_layer_tl
  \c_@@_option_layer_context_tl
  {
    \@@_define_option_commands_and_keyvals:
    \@@_define_renderers:
    \@@_define_renderer_prototypes:
  }
\ExplSyntaxOff
```

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]
```

Built-in ConTEXt themes provided with the Markdown package include:

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

```
\startmodule[markdownthemewitiko_markdown_defaults]
\unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConTEXt themes.

3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TEX *token renderers* is performed by the Lua layer. The plain TEX layer provides default definitions for the token renderers. The LATEX and ConTEXt layers correct idiosyncrasies of the respective TEX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain TEX, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

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

```
local upper, format, length =
  string.upper, string.format, string.len
local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
  lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
  lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain TEX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
local util = {}
```

The `util.err` method prints an error message `msg` and exits. If `exit_code` is provided, it specifies the exit code. Otherwise, the exit code will be 1.

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

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exists, it gets created with `transform(string)` as its content. Regardless, the pathname is then returned.

```
function util.cache(dir, string, salt, transform, suffix)
  local digest = md5.sumhexa(string .. (salt or ""))
  local name = util.pathname(dir, digest .. suffix)
  local file = io.open(name, "r")
  if file == nil then -- If no cache entry exists, create a new one.
    file = assert(io.open(name, "w"),
      [[Could not open file "]] .. name .. [[" for writing]])
    local result = string
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  return name
end
```

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

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

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

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

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

```
function util.encode_json_string(s)
  s = s:gsub([[\]], [[\\]])
  s = s:gsub([["]], [[\"]])
  return [["]] .. s .. [["]]
end
```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [11, Chapter 21].

```
function util.expand_tabs_in_line(s, tabstop)
  local tab = tabstop or 4
  local corr = 0
  return (s:gsub("()\t", function(p)
            local sp = tab - (p - 1 + corr) % tab
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
end
```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

```
function util.walk(t, f)
  local typ = type(t)
  if typ == "string" then
    f(t)
  elseif typ == "table" then
    local i = 1
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

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

```
function util.flatten(ary)
  local new = {}
  for _,v in ipairs(ary) do
    if type(v) == "table" then
```

```
3527        for _,w in ipairs(util.flatten(v)) do
3528          new[#new + 1] = w
3529        end
3530      else
3531        new[#new + 1] = v
3532      end
3533    end
3534    return new
3535  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.

```
3536  function util.rope_to_string(rope)
3537    local buffer = {}
3538    util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3539    return table.concat(buffer)
3540  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.

```
3541  function util.rope_last(rope)
3542    if #rope == 0 then
3543      return nil
3544    else
3545      local l = rope[#rope]
3546      if type(l) == "table" then
3547        return util.rope_last(l)
3548      else
3549        return l
3550      end
3551    end
3552  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`.

```
3553  function util.intersperse(ary, x)
3554    local new = {}
3555    local l = #ary
3556    for i,v in ipairs(ary) do
3557      local n = #new
3558      new[n + 1] = v
3559      if i ~= l then
3560        new[n + 2] = x
3561      end
3562    end
3563    return new
3564  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
```

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
local entities = {}

local character_entities = {
  ["Tab"] = 9,
  ["NewLine"] = 10,
```

```
  ["excl"] = 33,
  ["QUOT"] = 34,
  ["quot"] = 34,
  ["num"] = 35,
  ["dollar"] = 36,
  ["percnt"] = 37,
  ["AMP"] = 38,
  ["amp"] = 38,
  ["apos"] = 39,
  ["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,
  ["lbrack"] = 91,
  ["lsqb"] = 91,
  ["bsol"] = 92,
  ["rbrack"] = 93,
  ["rsqb"] = 93,
  ["Hat"] = 94,
  ["UnderBar"] = 95,
  ["lowbar"] = 95,
  ["DiacriticalGrave"] = 96,
  ["grave"] = 96,
  ["fjlig"] = {102, 106},
  ["lbrace"] = 123,
  ["lcub"] = 123,
  ["VerticalLine"] = 124,
  ["verbar"] = 124,
  ["vert"] = 124,
  ["rbrace"] = 125,
  ["rcub"] = 125,
```

```
  ["NonBreakingSpace"] = 160,
  ["nbsp"] = 160,
  ["iexcl"] = 161,
  ["cent"] = 162,
  ["pound"] = 163,
  ["curren"] = 164,
  ["yen"] = 165,
  ["brvbar"] = 166,
  ["sect"] = 167,
  ["Dot"] = 168,
  ["DoubleDot"] = 168,
  ["die"] = 168,
  ["uml"] = 168,
  ["COPY"] = 169,
  ["copy"] = 169,
  ["ordf"] = 170,
  ["laquo"] = 171,
  ["not"] = 172,
  ["shy"] = 173,
  ["REG"] = 174,
  ["circledR"] = 174,
  ["reg"] = 174,
  ["macr"] = 175,
  ["strns"] = 175,
  ["deg"] = 176,
  ["PlusMinus"] = 177,
  ["plusmn"] = 177,
  ["pm"] = 177,
  ["sup2"] = 178,
  ["sup3"] = 179,
  ["DiacriticalAcute"] = 180,
  ["acute"] = 180,
  ["micro"] = 181,
  ["para"] = 182,
  ["CenterDot"] = 183,
  ["centerdot"] = 183,
  ["middot"] = 183,
  ["Cedilla"] = 184,
  ["cedil"] = 184,
  ["sup1"] = 185,
  ["ordm"] = 186,
  ["raquo"] = 187,
  ["frac14"] = 188,
  ["frac12"] = 189,
  ["half"] = 189,
  ["frac34"] = 190,
  ["iquest"] = 191,
```

```
3711    ["Agrave"] = 192,
3712    ["Aacute"] = 193,
3713    ["Acirc"] = 194,
3714    ["Atilde"] = 195,
3715    ["Auml"] = 196,
3716    ["Aring"] = 197,
3717    ["angst"] = 197,
3718    ["AElig"] = 198,
3719    ["Ccedil"] = 199,
3720    ["Egrave"] = 200,
3721    ["Eacute"] = 201,
3722    ["Ecirc"] = 202,
3723    ["Euml"] = 203,
3724    ["Igrave"] = 204,
3725    ["Iacute"] = 205,
3726    ["Icirc"] = 206,
3727    ["Iuml"] = 207,
3728    ["ETH"] = 208,
3729    ["Ntilde"] = 209,
3730    ["Ograve"] = 210,
3731    ["Oacute"] = 211,
3732    ["Ocirc"] = 212,
3733    ["Otilde"] = 213,
3734    ["Ouml"] = 214,
3735    ["times"] = 215,
3736    ["Oslash"] = 216,
3737    ["Ugrave"] = 217,
3738    ["Uacute"] = 218,
3739    ["Ucirc"] = 219,
3740    ["Uuml"] = 220,
3741    ["Yacute"] = 221,
3742    ["THORN"] = 222,
3743    ["szlig"] = 223,
3744    ["agrave"] = 224,
3745    ["aacute"] = 225,
3746    ["acirc"] = 226,
3747    ["atilde"] = 227,
3748    ["auml"] = 228,
3749    ["aring"] = 229,
3750    ["aelig"] = 230,
3751    ["ccedil"] = 231,
3752    ["egrave"] = 232,
3753    ["eacute"] = 233,
3754    ["ecirc"] = 234,
3755    ["euml"] = 235,
3756    ["igrave"] = 236,
3757    ["iacute"] = 237,
```

```
3758   ["icirc"] = 238,
3759   ["iuml"] = 239,
3760   ["eth"] = 240,
3761   ["ntilde"] = 241,
3762   ["ograve"] = 242,
3763   ["oacute"] = 243,
3764   ["ocirc"] = 244,
3765   ["otilde"] = 245,
3766   ["ouml"] = 246,
3767   ["div"] = 247,
3768   ["divide"] = 247,
3769   ["oslash"] = 248,
3770   ["ugrave"] = 249,
3771   ["uacute"] = 250,
3772   ["ucirc"] = 251,
3773   ["uuml"] = 252,
3774   ["yacute"] = 253,
3775   ["thorn"] = 254,
3776   ["yuml"] = 255,
3777   ["Amacr"] = 256,
3778   ["amacr"] = 257,
3779   ["Abreve"] = 258,
3780   ["abreve"] = 259,
3781   ["Aogon"] = 260,
3782   ["aogon"] = 261,
3783   ["Cacute"] = 262,
3784   ["cacute"] = 263,
3785   ["Ccirc"] = 264,
3786   ["ccirc"] = 265,
3787   ["Cdot"] = 266,
3788   ["cdot"] = 267,
3789   ["Ccaron"] = 268,
3790   ["ccaron"] = 269,
3791   ["Dcaron"] = 270,
3792   ["dcaron"] = 271,
3793   ["Dstrok"] = 272,
3794   ["dstrok"] = 273,
3795   ["Emacr"] = 274,
3796   ["emacr"] = 275,
3797   ["Edot"] = 278,
3798   ["edot"] = 279,
3799   ["Eogon"] = 280,
3800   ["eogon"] = 281,
3801   ["Ecaron"] = 282,
3802   ["ecaron"] = 283,
3803   ["Gcirc"] = 284,
3804   ["gcirc"] = 285,
```

```
3805   ["Gbreve"] = 286,
3806   ["gbreve"] = 287,
3807   ["Gdot"] = 288,
3808   ["gdot"] = 289,
3809   ["Gcedil"] = 290,
3810   ["Hcirc"] = 292,
3811   ["hcirc"] = 293,
3812   ["Hstrok"] = 294,
3813   ["hstrok"] = 295,
3814   ["Itilde"] = 296,
3815   ["itilde"] = 297,
3816   ["Imacr"] = 298,
3817   ["imacr"] = 299,
3818   ["Iogon"] = 302,
3819   ["iogon"] = 303,
3820   ["Idot"] = 304,
3821   ["imath"] = 305,
3822   ["inodot"] = 305,
3823   ["IJlig"] = 306,
3824   ["ijlig"] = 307,
3825   ["Jcirc"] = 308,
3826   ["jcirc"] = 309,
3827   ["Kcedil"] = 310,
3828   ["kcedil"] = 311,
3829   ["kgreen"] = 312,
3830   ["Lacute"] = 313,
3831   ["lacute"] = 314,
3832   ["Lcedil"] = 315,
3833   ["lcedil"] = 316,
3834   ["Lcaron"] = 317,
3835   ["lcaron"] = 318,
3836   ["Lmidot"] = 319,
3837   ["lmidot"] = 320,
3838   ["Lstrok"] = 321,
3839   ["lstrok"] = 322,
3840   ["Nacute"] = 323,
3841   ["nacute"] = 324,
3842   ["Ncedil"] = 325,
3843   ["ncedil"] = 326,
3844   ["Ncaron"] = 327,
3845   ["ncaron"] = 328,
3846   ["napos"] = 329,
3847   ["ENG"] = 330,
3848   ["eng"] = 331,
3849   ["Omacr"] = 332,
3850   ["omacr"] = 333,
3851   ["Odblac"] = 336,
```

```
3852   ["odblac"] = 337,
3853   ["OElig"] = 338,
3854   ["oelig"] = 339,
3855   ["Racute"] = 340,
3856   ["racute"] = 341,
3857   ["Rcedil"] = 342,
3858   ["rcedil"] = 343,
3859   ["Rcaron"] = 344,
3860   ["rcaron"] = 345,
3861   ["Sacute"] = 346,
3862   ["sacute"] = 347,
3863   ["Scirc"] = 348,
3864   ["scirc"] = 349,
3865   ["Scedil"] = 350,
3866   ["scedil"] = 351,
3867   ["Scaron"] = 352,
3868   ["scaron"] = 353,
3869   ["Tcedil"] = 354,
3870   ["tcedil"] = 355,
3871   ["Tcaron"] = 356,
3872   ["tcaron"] = 357,
3873   ["Tstrok"] = 358,
3874   ["tstrok"] = 359,
3875   ["Utilde"] = 360,
3876   ["utilde"] = 361,
3877   ["Umacr"] = 362,
3878   ["umacr"] = 363,
3879   ["Ubreve"] = 364,
3880   ["ubreve"] = 365,
3881   ["Uring"] = 366,
3882   ["uring"] = 367,
3883   ["Udblac"] = 368,
3884   ["udblac"] = 369,
3885   ["Uogon"] = 370,
3886   ["uogon"] = 371,
3887   ["Wcirc"] = 372,
3888   ["wcirc"] = 373,
3889   ["Ycirc"] = 374,
3890   ["ycirc"] = 375,
3891   ["Yuml"] = 376,
3892   ["Zacute"] = 377,
3893   ["zacute"] = 378,
3894   ["Zdot"] = 379,
3895   ["zdot"] = 380,
3896   ["Zcaron"] = 381,
3897   ["zcaron"] = 382,
3898   ["fnof"] = 402,
```

```
  ["imped"] = 437,
  ["gacute"] = 501,
  ["jmath"] = 567,
  ["circ"] = 710,
  ["Hacek"] = 711,
  ["caron"] = 711,
  ["Breve"] = 728,
  ["breve"] = 728,
  ["DiacriticalDot"] = 729,
  ["dot"] = 729,
  ["ring"] = 730,
  ["ogon"] = 731,
  ["DiacriticalTilde"] = 732,
  ["tilde"] = 732,
  ["DiacriticalDoubleAcute"] = 733,
  ["dblac"] = 733,
  ["DownBreve"] = 785,
  ["Alpha"] = 913,
  ["Beta"] = 914,
  ["Gamma"] = 915,
  ["Delta"] = 916,
  ["Epsilon"] = 917,
  ["Zeta"] = 918,
  ["Eta"] = 919,
  ["Theta"] = 920,
  ["Iota"] = 921,
  ["Kappa"] = 922,
  ["Lambda"] = 923,
  ["Mu"] = 924,
  ["Nu"] = 925,
  ["Xi"] = 926,
  ["Omicron"] = 927,
  ["Pi"] = 928,
  ["Rho"] = 929,
  ["Sigma"] = 931,
  ["Tau"] = 932,
  ["Upsilon"] = 933,
  ["Phi"] = 934,
  ["Chi"] = 935,
  ["Psi"] = 936,
  ["Omega"] = 937,
  ["ohm"] = 937,
  ["alpha"] = 945,
  ["beta"] = 946,
  ["gamma"] = 947,
  ["delta"] = 948,
  ["epsi"] = 949,
```

```
  ["epsilon"] = 949,
  ["zeta"] = 950,
  ["eta"] = 951,
  ["theta"] = 952,
  ["iota"] = 953,
  ["kappa"] = 954,
  ["lambda"] = 955,
  ["mu"] = 956,
  ["nu"] = 957,
  ["xi"] = 958,
  ["omicron"] = 959,
  ["pi"] = 960,
  ["rho"] = 961,
  ["sigmaf"] = 962,
  ["sigmav"] = 962,
  ["varsigma"] = 962,
  ["sigma"] = 963,
  ["tau"] = 964,
  ["upsi"] = 965,
  ["upsilon"] = 965,
  ["phi"] = 966,
  ["chi"] = 967,
  ["psi"] = 968,
  ["omega"] = 969,
  ["thetasym"] = 977,
  ["thetav"] = 977,
  ["vartheta"] = 977,
  ["Upsi"] = 978,
  ["upsih"] = 978,
  ["phiv"] = 981,
  ["straightphi"] = 981,
  ["varphi"] = 981,
  ["piv"] = 982,
  ["varpi"] = 982,
  ["Gammad"] = 988,
  ["digamma"] = 989,
  ["gammad"] = 989,
  ["kappav"] = 1008,
  ["varkappa"] = 1008,
  ["rhov"] = 1009,
  ["varrho"] = 1009,
  ["epsiv"] = 1013,
  ["straightepsilon"] = 1013,
  ["varepsilon"] = 1013,
  ["backepsilon"] = 1014,
  ["bepsi"] = 1014,
  ["IOcy"] = 1025,
```

```
3993    ["DJcy"] = 1026,
3994    ["GJcy"] = 1027,
3995    ["Jukcy"] = 1028,
3996    ["DScy"] = 1029,
3997    ["Iukcy"] = 1030,
3998    ["YIcy"] = 1031,
3999    ["Jsercy"] = 1032,
4000    ["LJcy"] = 1033,
4001    ["NJcy"] = 1034,
4002    ["TSHcy"] = 1035,
4003    ["KJcy"] = 1036,
4004    ["Ubrcy"] = 1038,
4005    ["DZcy"] = 1039,
4006    ["Acy"] = 1040,
4007    ["Bcy"] = 1041,
4008    ["Vcy"] = 1042,
4009    ["Gcy"] = 1043,
4010    ["Dcy"] = 1044,
4011    ["IEcy"] = 1045,
4012    ["ZHcy"] = 1046,
4013    ["Zcy"] = 1047,
4014    ["Icy"] = 1048,
4015    ["Jcy"] = 1049,
4016    ["Kcy"] = 1050,
4017    ["Lcy"] = 1051,
4018    ["Mcy"] = 1052,
4019    ["Ncy"] = 1053,
4020    ["Ocy"] = 1054,
4021    ["Pcy"] = 1055,
4022    ["Rcy"] = 1056,
4023    ["Scy"] = 1057,
4024    ["Tcy"] = 1058,
4025    ["Ucy"] = 1059,
4026    ["Fcy"] = 1060,
4027    ["KHcy"] = 1061,
4028    ["TScy"] = 1062,
4029    ["CHcy"] = 1063,
4030    ["SHcy"] = 1064,
4031    ["SHCHcy"] = 1065,
4032    ["HARDcy"] = 1066,
4033    ["Ycy"] = 1067,
4034    ["SOFTcy"] = 1068,
4035    ["Ecy"] = 1069,
4036    ["YUcy"] = 1070,
4037    ["YAcy"] = 1071,
4038    ["acy"] = 1072,
4039    ["bcy"] = 1073,
```

```
4040    ["vcy"] = 1074,
4041    ["gcy"] = 1075,
4042    ["dcy"] = 1076,
4043    ["iecy"] = 1077,
4044    ["zhcy"] = 1078,
4045    ["zcy"] = 1079,
4046    ["icy"] = 1080,
4047    ["jcy"] = 1081,
4048    ["kcy"] = 1082,
4049    ["lcy"] = 1083,
4050    ["mcy"] = 1084,
4051    ["ncy"] = 1085,
4052    ["ocy"] = 1086,
4053    ["pcy"] = 1087,
4054    ["rcy"] = 1088,
4055    ["scy"] = 1089,
4056    ["tcy"] = 1090,
4057    ["ucy"] = 1091,
4058    ["fcy"] = 1092,
4059    ["khcy"] = 1093,
4060    ["tscy"] = 1094,
4061    ["chcy"] = 1095,
4062    ["shcy"] = 1096,
4063    ["shchcy"] = 1097,
4064    ["hardcy"] = 1098,
4065    ["ycy"] = 1099,
4066    ["softcy"] = 1100,
4067    ["ecy"] = 1101,
4068    ["yucy"] = 1102,
4069    ["yacy"] = 1103,
4070    ["iocy"] = 1105,
4071    ["djcy"] = 1106,
4072    ["gjcy"] = 1107,
4073    ["jukcy"] = 1108,
4074    ["dscy"] = 1109,
4075    ["iukcy"] = 1110,
4076    ["yicy"] = 1111,
4077    ["jsercy"] = 1112,
4078    ["ljcy"] = 1113,
4079    ["njcy"] = 1114,
4080    ["tshcy"] = 1115,
4081    ["kjcy"] = 1116,
4082    ["ubrcy"] = 1118,
4083    ["dzcy"] = 1119,
4084    ["ensp"] = 8194,
4085    ["emsp"] = 8195,
4086    ["emsp13"] = 8196,
```

```
4087   ["emsp14"] = 8197,
4088   ["numsp"] = 8199,
4089   ["puncsp"] = 8200,
4090   ["ThinSpace"] = 8201,
4091   ["thinsp"] = 8201,
4092   ["VeryThinSpace"] = 8202,
4093   ["hairsp"] = 8202,
4094   ["NegativeMediumSpace"] = 8203,
4095   ["NegativeThickSpace"] = 8203,
4096   ["NegativeThinSpace"] = 8203,
4097   ["NegativeVeryThinSpace"] = 8203,
4098   ["ZeroWidthSpace"] = 8203,
4099   ["zwnj"] = 8204,
4100   ["zwj"] = 8205,
4101   ["lrm"] = 8206,
4102   ["rlm"] = 8207,
4103   ["dash"] = 8208,
4104   ["hyphen"] = 8208,
4105   ["ndash"] = 8211,
4106   ["mdash"] = 8212,
4107   ["horbar"] = 8213,
4108   ["Verbar"] = 8214,
4109   ["Vert"] = 8214,
4110   ["OpenCurlyQuote"] = 8216,
4111   ["lsquo"] = 8216,
4112   ["CloseCurlyQuote"] = 8217,
4113   ["rsquo"] = 8217,
4114   ["rsquor"] = 8217,
4115   ["lsquor"] = 8218,
4116   ["sbquo"] = 8218,
4117   ["OpenCurlyDoubleQuote"] = 8220,
4118   ["ldquo"] = 8220,
4119   ["CloseCurlyDoubleQuote"] = 8221,
4120   ["rdquo"] = 8221,
4121   ["rdquor"] = 8221,
4122   ["bdquo"] = 8222,
4123   ["ldquor"] = 8222,
4124   ["dagger"] = 8224,
4125   ["Dagger"] = 8225,
4126   ["ddagger"] = 8225,
4127   ["bull"] = 8226,
4128   ["bullet"] = 8226,
4129   ["nldr"] = 8229,
4130   ["hellip"] = 8230,
4131   ["mldr"] = 8230,
4132   ["permil"] = 8240,
4133   ["pertenk"] = 8241,
```

```
  ["prime"] = 8242,
  ["Prime"] = 8243,
  ["tprime"] = 8244,
  ["backprime"] = 8245,
  ["bprime"] = 8245,
  ["lsaquo"] = 8249,
  ["rsaquo"] = 8250,
  ["OverBar"] = 8254,
  ["oline"] = 8254,
  ["caret"] = 8257,
  ["hybull"] = 8259,
  ["frasl"] = 8260,
  ["bsemi"] = 8271,
  ["qprime"] = 8279,
  ["MediumSpace"] = 8287,
  ["ThickSpace"] = {8287, 8202},
  ["NoBreak"] = 8288,
  ["ApplyFunction"] = 8289,
  ["af"] = 8289,
  ["InvisibleTimes"] = 8290,
  ["it"] = 8290,
  ["InvisibleComma"] = 8291,
  ["ic"] = 8291,
  ["euro"] = 8364,
  ["TripleDot"] = 8411,
  ["tdot"] = 8411,
  ["DotDot"] = 8412,
  ["Copf"] = 8450,
  ["complexes"] = 8450,
  ["incare"] = 8453,
  ["gscr"] = 8458,
  ["HilbertSpace"] = 8459,
  ["Hscr"] = 8459,
  ["hamilt"] = 8459,
  ["Hfr"] = 8460,
  ["Poincareplane"] = 8460,
  ["Hopf"] = 8461,
  ["quaternions"] = 8461,
  ["planckh"] = 8462,
  ["hbar"] = 8463,
  ["hslash"] = 8463,
  ["planck"] = 8463,
  ["plankv"] = 8463,
  ["Iscr"] = 8464,
  ["imagline"] = 8464,
  ["Ifr"] = 8465,
  ["Im"] = 8465,
```

```
  ["image"] = 8465,
  ["imagpart"] = 8465,
  ["Laplacetrf"] = 8466,
  ["Lscr"] = 8466,
  ["lagran"] = 8466,
  ["ell"] = 8467,
  ["Nopf"] = 8469,
  ["naturals"] = 8469,
  ["numero"] = 8470,
  ["copysr"] = 8471,
  ["weierp"] = 8472,
  ["wp"] = 8472,
  ["Popf"] = 8473,
  ["primes"] = 8473,
  ["Qopf"] = 8474,
  ["rationals"] = 8474,
  ["Rscr"] = 8475,
  ["realine"] = 8475,
  ["Re"] = 8476,
  ["Rfr"] = 8476,
  ["real"] = 8476,
  ["realpart"] = 8476,
  ["Ropf"] = 8477,
  ["reals"] = 8477,
  ["rx"] = 8478,
  ["TRADE"] = 8482,
  ["trade"] = 8482,
  ["Zopf"] = 8484,
  ["integers"] = 8484,
  ["mho"] = 8487,
  ["Zfr"] = 8488,
  ["zeetrf"] = 8488,
  ["iiota"] = 8489,
  ["Bernoullis"] = 8492,
  ["Bscr"] = 8492,
  ["bernou"] = 8492,
  ["Cayleys"] = 8493,
  ["Cfr"] = 8493,
  ["escr"] = 8495,
  ["Escr"] = 8496,
  ["expectation"] = 8496,
  ["Fouriertrf"] = 8497,
  ["Fscr"] = 8497,
  ["Mellintrf"] = 8499,
  ["Mscr"] = 8499,
  ["phmmat"] = 8499,
  ["order"] = 8500,
```

```
4228  ["orderof"] = 8500,
4229  ["oscr"] = 8500,
4230  ["alefsym"] = 8501,
4231  ["aleph"] = 8501,
4232  ["beth"] = 8502,
4233  ["gimel"] = 8503,
4234  ["daleth"] = 8504,
4235  ["CapitalDifferentialD"] = 8517,
4236  ["DD"] = 8517,
4237  ["DifferentialD"] = 8518,
4238  ["dd"] = 8518,
4239  ["ExponentialE"] = 8519,
4240  ["ee"] = 8519,
4241  ["exponentiale"] = 8519,
4242  ["ImaginaryI"] = 8520,
4243  ["ii"] = 8520,
4244  ["frac13"] = 8531,
4245  ["frac23"] = 8532,
4246  ["frac15"] = 8533,
4247  ["frac25"] = 8534,
4248  ["frac35"] = 8535,
4249  ["frac45"] = 8536,
4250  ["frac16"] = 8537,
4251  ["frac56"] = 8538,
4252  ["frac18"] = 8539,
4253  ["frac38"] = 8540,
4254  ["frac58"] = 8541,
4255  ["frac78"] = 8542,
4256  ["LeftArrow"] = 8592,
4257  ["ShortLeftArrow"] = 8592,
4258  ["larr"] = 8592,
4259  ["leftarrow"] = 8592,
4260  ["slarr"] = 8592,
4261  ["ShortUpArrow"] = 8593,
4262  ["UpArrow"] = 8593,
4263  ["uarr"] = 8593,
4264  ["uparrow"] = 8593,
4265  ["RightArrow"] = 8594,
4266  ["ShortRightArrow"] = 8594,
4267  ["rarr"] = 8594,
4268  ["rightarrow"] = 8594,
4269  ["srarr"] = 8594,
4270  ["DownArrow"] = 8595,
4271  ["ShortDownArrow"] = 8595,
4272  ["darr"] = 8595,
4273  ["downarrow"] = 8595,
4274  ["LeftRightArrow"] = 8596,
```

```
  ["harr"] = 8596,
  ["leftrightarrow"] = 8596,
  ["UpDownArrow"] = 8597,
  ["updownarrow"] = 8597,
  ["varr"] = 8597,
  ["UpperLeftArrow"] = 8598,
  ["nwarr"] = 8598,
  ["nwarrow"] = 8598,
  ["UpperRightArrow"] = 8599,
  ["nearr"] = 8599,
  ["nearrow"] = 8599,
  ["LowerRightArrow"] = 8600,
  ["searr"] = 8600,
  ["searrow"] = 8600,
  ["LowerLeftArrow"] = 8601,
  ["swarr"] = 8601,
  ["swarrow"] = 8601,
  ["nlarr"] = 8602,
  ["nleftarrow"] = 8602,
  ["nrarr"] = 8603,
  ["nrightarrow"] = 8603,
  ["nrarrw"] = {8605, 824},
  ["rarrw"] = 8605,
  ["rightsquigarrow"] = 8605,
  ["Larr"] = 8606,
  ["twoheadleftarrow"] = 8606,
  ["Uarr"] = 8607,
  ["Rarr"] = 8608,
  ["twoheadrightarrow"] = 8608,
  ["Darr"] = 8609,
  ["larrtl"] = 8610,
  ["leftarrowtail"] = 8610,
  ["rarrtl"] = 8611,
  ["rightarrowtail"] = 8611,
  ["LeftTeeArrow"] = 8612,
  ["mapstoleft"] = 8612,
  ["UpTeeArrow"] = 8613,
  ["mapstoup"] = 8613,
  ["RightTeeArrow"] = 8614,
  ["map"] = 8614,
  ["mapsto"] = 8614,
  ["DownTeeArrow"] = 8615,
  ["mapstodown"] = 8615,
  ["hookleftarrow"] = 8617,
  ["larrhk"] = 8617,
  ["hookrightarrow"] = 8618,
  ["rarrhk"] = 8618,
```

```
4322  ["larrlp"] = 8619,
4323  ["looparrowleft"] = 8619,
4324  ["looparrowright"] = 8620,
4325  ["rarrlp"] = 8620,
4326  ["harrw"] = 8621,
4327  ["leftrightsquigarrow"] = 8621,
4328  ["nharr"] = 8622,
4329  ["nleftrightarrow"] = 8622,
4330  ["Lsh"] = 8624,
4331  ["lsh"] = 8624,
4332  ["Rsh"] = 8625,
4333  ["rsh"] = 8625,
4334  ["ldsh"] = 8626,
4335  ["rdsh"] = 8627,
4336  ["crarr"] = 8629,
4337  ["cularr"] = 8630,
4338  ["curvearrowleft"] = 8630,
4339  ["curarr"] = 8631,
4340  ["curvearrowright"] = 8631,
4341  ["circlearrowleft"] = 8634,
4342  ["olarr"] = 8634,
4343  ["circlearrowright"] = 8635,
4344  ["orarr"] = 8635,
4345  ["LeftVector"] = 8636,
4346  ["leftharpoonup"] = 8636,
4347  ["lharu"] = 8636,
4348  ["DownLeftVector"] = 8637,
4349  ["leftharpoondown"] = 8637,
4350  ["lhard"] = 8637,
4351  ["RightUpVector"] = 8638,
4352  ["uharr"] = 8638,
4353  ["upharpoonright"] = 8638,
4354  ["LeftUpVector"] = 8639,
4355  ["uharl"] = 8639,
4356  ["upharpoonleft"] = 8639,
4357  ["RightVector"] = 8640,
4358  ["rharu"] = 8640,
4359  ["rightharpoonup"] = 8640,
4360  ["DownRightVector"] = 8641,
4361  ["rhard"] = 8641,
4362  ["rightharpoondown"] = 8641,
4363  ["RightDownVector"] = 8642,
4364  ["dharr"] = 8642,
4365  ["downharpoonright"] = 8642,
4366  ["LeftDownVector"] = 8643,
4367  ["dharl"] = 8643,
4368  ["downharpoonleft"] = 8643,
```

```
4369    ["RightArrowLeftArrow"] = 8644,
4370    ["rightleftarrows"] = 8644,
4371    ["rlarr"] = 8644,
4372    ["UpArrowDownArrow"] = 8645,
4373    ["udarr"] = 8645,
4374    ["LeftArrowRightArrow"] = 8646,
4375    ["leftrightarrows"] = 8646,
4376    ["lrarr"] = 8646,
4377    ["leftleftarrows"] = 8647,
4378    ["llarr"] = 8647,
4379    ["upuparrows"] = 8648,
4380    ["uuarr"] = 8648,
4381    ["rightrightarrows"] = 8649,
4382    ["rrarr"] = 8649,
4383    ["ddarr"] = 8650,
4384    ["downdownarrows"] = 8650,
4385    ["ReverseEquilibrium"] = 8651,
4386    ["leftrightharpoons"] = 8651,
4387    ["lrhar"] = 8651,
4388    ["Equilibrium"] = 8652,
4389    ["rightleftharpoons"] = 8652,
4390    ["rlhar"] = 8652,
4391    ["nLeftarrow"] = 8653,
4392    ["nlArr"] = 8653,
4393    ["nLeftrightarrow"] = 8654,
4394    ["nhArr"] = 8654,
4395    ["nRightarrow"] = 8655,
4396    ["nrArr"] = 8655,
4397    ["DoubleLeftArrow"] = 8656,
4398    ["Leftarrow"] = 8656,
4399    ["lArr"] = 8656,
4400    ["DoubleUpArrow"] = 8657,
4401    ["Uparrow"] = 8657,
4402    ["uArr"] = 8657,
4403    ["DoubleRightArrow"] = 8658,
4404    ["Implies"] = 8658,
4405    ["Rightarrow"] = 8658,
4406    ["rArr"] = 8658,
4407    ["DoubleDownArrow"] = 8659,
4408    ["Downarrow"] = 8659,
4409    ["dArr"] = 8659,
4410    ["DoubleLeftRightArrow"] = 8660,
4411    ["Leftrightarrow"] = 8660,
4412    ["hArr"] = 8660,
4413    ["iff"] = 8660,
4414    ["DoubleUpDownArrow"] = 8661,
4415    ["Updownarrow"] = 8661,
```

```
4416   ["vArr"] = 8661,
4417   ["nwArr"] = 8662,
4418   ["neArr"] = 8663,
4419   ["seArr"] = 8664,
4420   ["swArr"] = 8665,
4421   ["Lleftarrow"] = 8666,
4422   ["lAarr"] = 8666,
4423   ["Rrightarrow"] = 8667,
4424   ["rAarr"] = 8667,
4425   ["zigrarr"] = 8669,
4426   ["LeftArrowBar"] = 8676,
4427   ["larrb"] = 8676,
4428   ["RightArrowBar"] = 8677,
4429   ["rarrb"] = 8677,
4430   ["DownArrowUpArrow"] = 8693,
4431   ["duarr"] = 8693,
4432   ["loarr"] = 8701,
4433   ["roarr"] = 8702,
4434   ["hoarr"] = 8703,
4435   ["ForAll"] = 8704,
4436   ["forall"] = 8704,
4437   ["comp"] = 8705,
4438   ["complement"] = 8705,
4439   ["PartialD"] = 8706,
4440   ["npart"] = {8706, 824},
4441   ["part"] = 8706,
4442   ["Exists"] = 8707,
4443   ["exist"] = 8707,
4444   ["NotExists"] = 8708,
4445   ["nexist"] = 8708,
4446   ["nexists"] = 8708,
4447   ["empty"] = 8709,
4448   ["emptyset"] = 8709,
4449   ["emptyv"] = 8709,
4450   ["varnothing"] = 8709,
4451   ["Del"] = 8711,
4452   ["nabla"] = 8711,
4453   ["Element"] = 8712,
4454   ["in"] = 8712,
4455   ["isin"] = 8712,
4456   ["isinv"] = 8712,
4457   ["NotElement"] = 8713,
4458   ["notin"] = 8713,
4459   ["notinva"] = 8713,
4460   ["ReverseElement"] = 8715,
4461   ["SuchThat"] = 8715,
4462   ["ni"] = 8715,
```

```
  ["niv"] = 8715,
  ["NotReverseElement"] = 8716,
  ["notni"] = 8716,
  ["notniva"] = 8716,
  ["Product"] = 8719,
  ["prod"] = 8719,
  ["Coproduct"] = 8720,
  ["coprod"] = 8720,
  ["Sum"] = 8721,
  ["sum"] = 8721,
  ["minus"] = 8722,
  ["MinusPlus"] = 8723,
  ["mnplus"] = 8723,
  ["mp"] = 8723,
  ["dotplus"] = 8724,
  ["plusdo"] = 8724,
  ["Backslash"] = 8726,
  ["setminus"] = 8726,
  ["setmn"] = 8726,
  ["smallsetminus"] = 8726,
  ["ssetmn"] = 8726,
  ["lowast"] = 8727,
  ["SmallCircle"] = 8728,
  ["compfn"] = 8728,
  ["Sqrt"] = 8730,
  ["radic"] = 8730,
  ["Proportional"] = 8733,
  ["prop"] = 8733,
  ["propto"] = 8733,
  ["varpropto"] = 8733,
  ["vprop"] = 8733,
  ["infin"] = 8734,
  ["angrt"] = 8735,
  ["ang"] = 8736,
  ["angle"] = 8736,
  ["nang"] = {8736, 8402},
  ["angmsd"] = 8737,
  ["measuredangle"] = 8737,
  ["angsph"] = 8738,
  ["VerticalBar"] = 8739,
  ["mid"] = 8739,
  ["shortmid"] = 8739,
  ["smid"] = 8739,
  ["NotVerticalBar"] = 8740,
  ["nmid"] = 8740,
  ["nshortmid"] = 8740,
  ["nsmid"] = 8740,
```

```
["DoubleVerticalBar"] = 8741,
["par"] = 8741,
["parallel"] = 8741,
["shortparallel"] = 8741,
["spar"] = 8741,
["NotDoubleVerticalBar"] = 8742,
["npar"] = 8742,
["nparallel"] = 8742,
["nshortparallel"] = 8742,
["nspar"] = 8742,
["and"] = 8743,
["wedge"] = 8743,
["or"] = 8744,
["vee"] = 8744,
["cap"] = 8745,
["caps"] = {8745, 65024},
["cup"] = 8746,
["cups"] = {8746, 65024},
["Integral"] = 8747,
["int"] = 8747,
["Int"] = 8748,
["iiint"] = 8749,
["tint"] = 8749,
["ContourIntegral"] = 8750,
["conint"] = 8750,
["oint"] = 8750,
["Conint"] = 8751,
["DoubleContourIntegral"] = 8751,
["Cconint"] = 8752,
["cwint"] = 8753,
["ClockwiseContourIntegral"] = 8754,
["cwconint"] = 8754,
["CounterClockwiseContourIntegral"] = 8755,
["awconint"] = 8755,
["Therefore"] = 8756,
["there4"] = 8756,
["therefore"] = 8756,
["Because"] = 8757,
["becaus"] = 8757,
["because"] = 8757,
["ratio"] = 8758,
["Colon"] = 8759,
["Proportion"] = 8759,
["dotminus"] = 8760,
["minusd"] = 8760,
["mDDot"] = 8762,
["homtht"] = 8763,
```

```
["Tilde"] = 8764,
["nvsim"] = {8764, 8402},
["sim"] = 8764,
["thicksim"] = 8764,
["thksim"] = 8764,
["backsim"] = 8765,
["bsim"] = 8765,
["race"] = {8765, 817},
["ac"] = 8766,
["acE"] = {8766, 819},
["mstpos"] = 8766,
["acd"] = 8767,
["VerticalTilde"] = 8768,
["wr"] = 8768,
["wreath"] = 8768,
["NotTilde"] = 8769,
["nsim"] = 8769,
["EqualTilde"] = 8770,
["NotEqualTilde"] = {8770, 824},
["eqsim"] = 8770,
["esim"] = 8770,
["nesim"] = {8770, 824},
["TildeEqual"] = 8771,
["sime"] = 8771,
["simeq"] = 8771,
["NotTildeEqual"] = 8772,
["nsime"] = 8772,
["nsimeq"] = 8772,
["TildeFullEqual"] = 8773,
["cong"] = 8773,
["simne"] = 8774,
["NotTildeFullEqual"] = 8775,
["ncong"] = 8775,
["TildeTilde"] = 8776,
["ap"] = 8776,
["approx"] = 8776,
["asymp"] = 8776,
["thickapprox"] = 8776,
["thkap"] = 8776,
["NotTildeTilde"] = 8777,
["nap"] = 8777,
["napprox"] = 8777,
["ape"] = 8778,
["approxeq"] = 8778,
["apid"] = 8779,
["napid"] = {8779, 824},
["backcong"] = 8780,
```

```
4604    ["bcong"] = 8780,
4605    ["CupCap"] = 8781,
4606    ["asympeq"] = 8781,
4607    ["nvap"] = {8781, 8402},
4608    ["Bumpeq"] = 8782,
4609    ["HumpDownHump"] = 8782,
4610    ["NotHumpDownHump"] = {8782, 824},
4611    ["bump"] = 8782,
4612    ["nbump"] = {8782, 824},
4613    ["HumpEqual"] = 8783,
4614    ["NotHumpEqual"] = {8783, 824},
4615    ["bumpe"] = 8783,
4616    ["bumpeq"] = 8783,
4617    ["nbumpe"] = {8783, 824},
4618    ["DotEqual"] = 8784,
4619    ["doteq"] = 8784,
4620    ["esdot"] = 8784,
4621    ["nedot"] = {8784, 824},
4622    ["doteqdot"] = 8785,
4623    ["eDot"] = 8785,
4624    ["efDot"] = 8786,
4625    ["fallingdotseq"] = 8786,
4626    ["erDot"] = 8787,
4627    ["risingdotseq"] = 8787,
4628    ["Assign"] = 8788,
4629    ["colone"] = 8788,
4630    ["coloneq"] = 8788,
4631    ["ecolon"] = 8789,
4632    ["eqcolon"] = 8789,
4633    ["ecir"] = 8790,
4634    ["eqcirc"] = 8790,
4635    ["circeq"] = 8791,
4636    ["cire"] = 8791,
4637    ["wedgeq"] = 8793,
4638    ["veeeq"] = 8794,
4639    ["triangleq"] = 8796,
4640    ["trie"] = 8796,
4641    ["equest"] = 8799,
4642    ["questeq"] = 8799,
4643    ["NotEqual"] = 8800,
4644    ["ne"] = 8800,
4645    ["Congruent"] = 8801,
4646    ["bnequiv"] = {8801, 8421},
4647    ["equiv"] = 8801,
4648    ["NotCongruent"] = 8802,
4649    ["nequiv"] = 8802,
4650    ["le"] = 8804,
```

```
  ["leq"] = 8804,
  ["nvle"] = {8804, 8402},
  ["GreaterEqual"] = 8805,
  ["ge"] = 8805,
  ["geq"] = 8805,
  ["nvge"] = {8805, 8402},
  ["LessFullEqual"] = 8806,
  ["lE"] = 8806,
  ["leqq"] = 8806,
  ["nlE"] = {8806, 824},
  ["nleqq"] = {8806, 824},
  ["GreaterFullEqual"] = 8807,
  ["NotGreaterFullEqual"] = {8807, 824},
  ["gE"] = 8807,
  ["geqq"] = 8807,
  ["ngE"] = {8807, 824},
  ["ngeqq"] = {8807, 824},
  ["lnE"] = 8808,
  ["lneqq"] = 8808,
  ["lvertneqq"] = {8808, 65024},
  ["lvnE"] = {8808, 65024},
  ["gnE"] = 8809,
  ["gneqq"] = 8809,
  ["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,
```

```
4698  ["nle"] = 8816,
4699  ["nleq"] = 8816,
4700  ["NotGreaterEqual"] = 8817,
4701  ["nge"] = 8817,
4702  ["ngeq"] = 8817,
4703  ["LessTilde"] = 8818,
4704  ["lesssim"] = 8818,
4705  ["lsim"] = 8818,
4706  ["GreaterTilde"] = 8819,
4707  ["gsim"] = 8819,
4708  ["gtrsim"] = 8819,
4709  ["NotLessTilde"] = 8820,
4710  ["nlsim"] = 8820,
4711  ["NotGreaterTilde"] = 8821,
4712  ["ngsim"] = 8821,
4713  ["LessGreater"] = 8822,
4714  ["lessgtr"] = 8822,
4715  ["lg"] = 8822,
4716  ["GreaterLess"] = 8823,
4717  ["gl"] = 8823,
4718  ["gtrless"] = 8823,
4719  ["NotLessGreater"] = 8824,
4720  ["ntlg"] = 8824,
4721  ["NotGreaterLess"] = 8825,
4722  ["ntgl"] = 8825,
4723  ["Precedes"] = 8826,
4724  ["pr"] = 8826,
4725  ["prec"] = 8826,
4726  ["Succeeds"] = 8827,
4727  ["sc"] = 8827,
4728  ["succ"] = 8827,
4729  ["PrecedesSlantEqual"] = 8828,
4730  ["prcue"] = 8828,
4731  ["preccurlyeq"] = 8828,
4732  ["SucceedsSlantEqual"] = 8829,
4733  ["sccue"] = 8829,
4734  ["succcurlyeq"] = 8829,
4735  ["PrecedesTilde"] = 8830,
4736  ["precsim"] = 8830,
4737  ["prsim"] = 8830,
4738  ["NotSucceedsTilde"] = {8831, 824},
4739  ["SucceedsTilde"] = 8831,
4740  ["scsim"] = 8831,
4741  ["succsim"] = 8831,
4742  ["NotPrecedes"] = 8832,
4743  ["npr"] = 8832,
4744  ["nprec"] = 8832,
```

```
4745    ["NotSucceeds"] = 8833,
4746    ["nsc"] = 8833,
4747    ["nsucc"] = 8833,
4748    ["NotSubset"] = {8834, 8402},
4749    ["nsubset"] = {8834, 8402},
4750    ["sub"] = 8834,
4751    ["subset"] = 8834,
4752    ["vnsub"] = {8834, 8402},
4753    ["NotSuperset"] = {8835, 8402},
4754    ["Superset"] = 8835,
4755    ["nsupset"] = {8835, 8402},
4756    ["sup"] = 8835,
4757    ["supset"] = 8835,
4758    ["vnsup"] = {8835, 8402},
4759    ["nsub"] = 8836,
4760    ["nsup"] = 8837,
4761    ["SubsetEqual"] = 8838,
4762    ["sube"] = 8838,
4763    ["subseteq"] = 8838,
4764    ["SupersetEqual"] = 8839,
4765    ["supe"] = 8839,
4766    ["supseteq"] = 8839,
4767    ["NotSubsetEqual"] = 8840,
4768    ["nsube"] = 8840,
4769    ["nsubseteq"] = 8840,
4770    ["NotSupersetEqual"] = 8841,
4771    ["nsupe"] = 8841,
4772    ["nsupseteq"] = 8841,
4773    ["subne"] = 8842,
4774    ["subsetneq"] = 8842,
4775    ["varsubsetneq"] = {8842, 65024},
4776    ["vsubne"] = {8842, 65024},
4777    ["supne"] = 8843,
4778    ["supsetneq"] = 8843,
4779    ["varsupsetneq"] = {8843, 65024},
4780    ["vsupne"] = {8843, 65024},
4781    ["cupdot"] = 8845,
4782    ["UnionPlus"] = 8846,
4783    ["uplus"] = 8846,
4784    ["NotSquareSubset"] = {8847, 824},
4785    ["SquareSubset"] = 8847,
4786    ["sqsub"] = 8847,
4787    ["sqsubset"] = 8847,
4788    ["NotSquareSuperset"] = {8848, 824},
4789    ["SquareSuperset"] = 8848,
4790    ["sqsup"] = 8848,
4791    ["sqsupset"] = 8848,
```

```
4792  ["SquareSubsetEqual"] = 8849,
4793  ["sqsube"] = 8849,
4794  ["sqsubseteq"] = 8849,
4795  ["SquareSupersetEqual"] = 8850,
4796  ["sqsupe"] = 8850,
4797  ["sqsupseteq"] = 8850,
4798  ["SquareIntersection"] = 8851,
4799  ["sqcap"] = 8851,
4800  ["sqcaps"] = {8851, 65024},
4801  ["SquareUnion"] = 8852,
4802  ["sqcup"] = 8852,
4803  ["sqcups"] = {8852, 65024},
4804  ["CirclePlus"] = 8853,
4805  ["oplus"] = 8853,
4806  ["CircleMinus"] = 8854,
4807  ["ominus"] = 8854,
4808  ["CircleTimes"] = 8855,
4809  ["otimes"] = 8855,
4810  ["osol"] = 8856,
4811  ["CircleDot"] = 8857,
4812  ["odot"] = 8857,
4813  ["circledcirc"] = 8858,
4814  ["ocir"] = 8858,
4815  ["circledast"] = 8859,
4816  ["oast"] = 8859,
4817  ["circleddash"] = 8861,
4818  ["odash"] = 8861,
4819  ["boxplus"] = 8862,
4820  ["plusb"] = 8862,
4821  ["boxminus"] = 8863,
4822  ["minusb"] = 8863,
4823  ["boxtimes"] = 8864,
4824  ["timesb"] = 8864,
4825  ["dotsquare"] = 8865,
4826  ["sdotb"] = 8865,
4827  ["RightTee"] = 8866,
4828  ["vdash"] = 8866,
4829  ["LeftTee"] = 8867,
4830  ["dashv"] = 8867,
4831  ["DownTee"] = 8868,
4832  ["top"] = 8868,
4833  ["UpTee"] = 8869,
4834  ["bot"] = 8869,
4835  ["bottom"] = 8869,
4836  ["perp"] = 8869,
4837  ["models"] = 8871,
4838  ["DoubleRightTee"] = 8872,
```

```
4839  ["vDash"] = 8872,
4840  ["Vdash"] = 8873,
4841  ["Vvdash"] = 8874,
4842  ["VDash"] = 8875,
4843  ["nvdash"] = 8876,
4844  ["nvDash"] = 8877,
4845  ["nVdash"] = 8878,
4846  ["nVDash"] = 8879,
4847  ["prurel"] = 8880,
4848  ["LeftTriangle"] = 8882,
4849  ["vartriangleleft"] = 8882,
4850  ["vltri"] = 8882,
4851  ["RightTriangle"] = 8883,
4852  ["vartriangleright"] = 8883,
4853  ["vrtri"] = 8883,
4854  ["LeftTriangleEqual"] = 8884,
4855  ["ltrie"] = 8884,
4856  ["nvltrie"] = {8884, 8402},
4857  ["trianglelefteq"] = 8884,
4858  ["RightTriangleEqual"] = 8885,
4859  ["nvrtrie"] = {8885, 8402},
4860  ["rtrie"] = 8885,
4861  ["trianglerighteq"] = 8885,
4862  ["origof"] = 8886,
4863  ["imof"] = 8887,
4864  ["multimap"] = 8888,
4865  ["mumap"] = 8888,
4866  ["hercon"] = 8889,
4867  ["intcal"] = 8890,
4868  ["intercal"] = 8890,
4869  ["veebar"] = 8891,
4870  ["barvee"] = 8893,
4871  ["angrtvb"] = 8894,
4872  ["lrtri"] = 8895,
4873  ["Wedge"] = 8896,
4874  ["bigwedge"] = 8896,
4875  ["xwedge"] = 8896,
4876  ["Vee"] = 8897,
4877  ["bigvee"] = 8897,
4878  ["xvee"] = 8897,
4879  ["Intersection"] = 8898,
4880  ["bigcap"] = 8898,
4881  ["xcap"] = 8898,
4882  ["Union"] = 8899,
4883  ["bigcup"] = 8899,
4884  ["xcup"] = 8899,
4885  ["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,
  ["isindot"] = 8949,
  ["notindot"] = {8949, 824},
  ["notinvc"] = 8950,
  ["notinvb"] = 8951,
  ["isinE"] = 8953,
  ["notinE"] = {8953, 824},
  ["nisd"] = 8954,
  ["xnis"] = 8955,
  ["nis"] = 8956,
  ["notnivc"] = 8957,
  ["notnivb"] = 8958,
```

```
4980    ["barwed"] = 8965,
4981    ["barwedge"] = 8965,
4982    ["Barwed"] = 8966,
4983    ["doublebarwedge"] = 8966,
4984    ["LeftCeiling"] = 8968,
4985    ["lceil"] = 8968,
4986    ["RightCeiling"] = 8969,
4987    ["rceil"] = 8969,
4988    ["LeftFloor"] = 8970,
4989    ["lfloor"] = 8970,
4990    ["RightFloor"] = 8971,
4991    ["rfloor"] = 8971,
4992    ["drcrop"] = 8972,
4993    ["dlcrop"] = 8973,
4994    ["urcrop"] = 8974,
4995    ["ulcrop"] = 8975,
4996    ["bnot"] = 8976,
4997    ["profline"] = 8978,
4998    ["profsurf"] = 8979,
4999    ["telrec"] = 8981,
5000    ["target"] = 8982,
5001    ["ulcorn"] = 8988,
5002    ["ulcorner"] = 8988,
5003    ["urcorn"] = 8989,
5004    ["urcorner"] = 8989,
5005    ["dlcorn"] = 8990,
5006    ["llcorner"] = 8990,
5007    ["drcorn"] = 8991,
5008    ["lrcorner"] = 8991,
5009    ["frown"] = 8994,
5010    ["sfrown"] = 8994,
5011    ["smile"] = 8995,
5012    ["ssmile"] = 8995,
5013    ["cylcty"] = 9005,
5014    ["profalar"] = 9006,
5015    ["topbot"] = 9014,
5016    ["ovbar"] = 9021,
5017    ["solbar"] = 9023,
5018    ["angzarr"] = 9084,
5019    ["lmoust"] = 9136,
5020    ["lmoustache"] = 9136,
5021    ["rmoust"] = 9137,
5022    ["rmoustache"] = 9137,
5023    ["OverBracket"] = 9140,
5024    ["tbrk"] = 9140,
5025    ["UnderBracket"] = 9141,
5026    ["bbrk"] = 9141,
```

```
5027   ["bbrktbrk"] = 9142,
5028   ["OverParenthesis"] = 9180,
5029   ["UnderParenthesis"] = 9181,
5030   ["OverBrace"] = 9182,
5031   ["UnderBrace"] = 9183,
5032   ["trpezium"] = 9186,
5033   ["elinters"] = 9191,
5034   ["blank"] = 9251,
5035   ["circledS"] = 9416,
5036   ["oS"] = 9416,
5037   ["HorizontalLine"] = 9472,
5038   ["boxh"] = 9472,
5039   ["boxv"] = 9474,
5040   ["boxdr"] = 9484,
5041   ["boxdl"] = 9488,
5042   ["boxur"] = 9492,
5043   ["boxul"] = 9496,
5044   ["boxvr"] = 9500,
5045   ["boxvl"] = 9508,
5046   ["boxhd"] = 9516,
5047   ["boxhu"] = 9524,
5048   ["boxvh"] = 9532,
5049   ["boxH"] = 9552,
5050   ["boxV"] = 9553,
5051   ["boxdR"] = 9554,
5052   ["boxDr"] = 9555,
5053   ["boxDR"] = 9556,
5054   ["boxdL"] = 9557,
5055   ["boxDl"] = 9558,
5056   ["boxDL"] = 9559,
5057   ["boxuR"] = 9560,
5058   ["boxUr"] = 9561,
5059   ["boxUR"] = 9562,
5060   ["boxuL"] = 9563,
5061   ["boxUl"] = 9564,
5062   ["boxUL"] = 9565,
5063   ["boxvR"] = 9566,
5064   ["boxVr"] = 9567,
5065   ["boxVR"] = 9568,
5066   ["boxvL"] = 9569,
5067   ["boxVl"] = 9570,
5068   ["boxVL"] = 9571,
5069   ["boxHd"] = 9572,
5070   ["boxhD"] = 9573,
5071   ["boxHD"] = 9574,
5072   ["boxHu"] = 9575,
5073   ["boxhU"] = 9576,
```

```
  ["boxHU"] = 9577,
  ["boxvH"] = 9578,
  ["boxVh"] = 9579,
  ["boxVH"] = 9580,
  ["uhblk"] = 9600,
  ["lhblk"] = 9604,
  ["block"] = 9608,
  ["blk14"] = 9617,
  ["blk12"] = 9618,
  ["blk34"] = 9619,
  ["Square"] = 9633,
  ["squ"] = 9633,
  ["square"] = 9633,
  ["FilledVerySmallSquare"] = 9642,
  ["blacksquare"] = 9642,
  ["squarf"] = 9642,
  ["squf"] = 9642,
  ["EmptyVerySmallSquare"] = 9643,
  ["rect"] = 9645,
  ["marker"] = 9646,
  ["fltns"] = 9649,
  ["bigtriangleup"] = 9651,
  ["xutri"] = 9651,
  ["blacktriangle"] = 9652,
  ["utrif"] = 9652,
  ["triangle"] = 9653,
  ["utri"] = 9653,
  ["blacktriangleright"] = 9656,
  ["rtrif"] = 9656,
  ["rtri"] = 9657,
  ["triangleright"] = 9657,
  ["bigtriangledown"] = 9661,
  ["xdtri"] = 9661,
  ["blacktriangledown"] = 9662,
  ["dtrif"] = 9662,
  ["dtri"] = 9663,
  ["triangledown"] = 9663,
  ["blacktriangleleft"] = 9666,
  ["ltrif"] = 9666,
  ["ltri"] = 9667,
  ["triangleleft"] = 9667,
  ["loz"] = 9674,
  ["lozenge"] = 9674,
  ["cir"] = 9675,
  ["tridot"] = 9708,
  ["bigcirc"] = 9711,
  ["xcirc"] = 9711,
```

```
5121   ["ultri"] = 9720,
5122   ["urtri"] = 9721,
5123   ["lltri"] = 9722,
5124   ["EmptySmallSquare"] = 9723,
5125   ["FilledSmallSquare"] = 9724,
5126   ["bigstar"] = 9733,
5127   ["starf"] = 9733,
5128   ["star"] = 9734,
5129   ["phone"] = 9742,
5130   ["female"] = 9792,
5131   ["male"] = 9794,
5132   ["spades"] = 9824,
5133   ["spadesuit"] = 9824,
5134   ["clubs"] = 9827,
5135   ["clubsuit"] = 9827,
5136   ["hearts"] = 9829,
5137   ["heartsuit"] = 9829,
5138   ["diamondsuit"] = 9830,
5139   ["diams"] = 9830,
5140   ["sung"] = 9834,
5141   ["flat"] = 9837,
5142   ["natur"] = 9838,
5143   ["natural"] = 9838,
5144   ["sharp"] = 9839,
5145   ["check"] = 10003,
5146   ["checkmark"] = 10003,
5147   ["cross"] = 10007,
5148   ["malt"] = 10016,
5149   ["maltese"] = 10016,
5150   ["sext"] = 10038,
5151   ["VerticalSeparator"] = 10072,
5152   ["lbbrk"] = 10098,
5153   ["rbbrk"] = 10099,
5154   ["bsolhsub"] = 10184,
5155   ["suphsol"] = 10185,
5156   ["LeftDoubleBracket"] = 10214,
5157   ["lobrk"] = 10214,
5158   ["RightDoubleBracket"] = 10215,
5159   ["robrk"] = 10215,
5160   ["LeftAngleBracket"] = 10216,
5161   ["lang"] = 10216,
5162   ["langle"] = 10216,
5163   ["RightAngleBracket"] = 10217,
5164   ["rang"] = 10217,
5165   ["rangle"] = 10217,
5166   ["Lang"] = 10218,
5167   ["Rang"] = 10219,
```

```
5168   ["loang"] = 10220,
5169   ["roang"] = 10221,
5170   ["LongLeftArrow"] = 10229,
5171   ["longleftarrow"] = 10229,
5172   ["xlarr"] = 10229,
5173   ["LongRightArrow"] = 10230,
5174   ["longrightarrow"] = 10230,
5175   ["xrarr"] = 10230,
5176   ["LongLeftRightArrow"] = 10231,
5177   ["longleftrightarrow"] = 10231,
5178   ["xharr"] = 10231,
5179   ["DoubleLongLeftArrow"] = 10232,
5180   ["Longleftarrow"] = 10232,
5181   ["xlArr"] = 10232,
5182   ["DoubleLongRightArrow"] = 10233,
5183   ["Longrightarrow"] = 10233,
5184   ["xrArr"] = 10233,
5185   ["DoubleLongLeftRightArrow"] = 10234,
5186   ["Longleftrightarrow"] = 10234,
5187   ["xhArr"] = 10234,
5188   ["longmapsto"] = 10236,
5189   ["xmap"] = 10236,
5190   ["dzigrarr"] = 10239,
5191   ["nvlArr"] = 10498,
5192   ["nvrArr"] = 10499,
5193   ["nvHarr"] = 10500,
5194   ["Map"] = 10501,
5195   ["lbarr"] = 10508,
5196   ["bkarow"] = 10509,
5197   ["rbarr"] = 10509,
5198   ["lBarr"] = 10510,
5199   ["dbkarow"] = 10511,
5200   ["rBarr"] = 10511,
5201   ["RBarr"] = 10512,
5202   ["drbkarow"] = 10512,
5203   ["DDotrahd"] = 10513,
5204   ["UpArrowBar"] = 10514,
5205   ["DownArrowBar"] = 10515,
5206   ["Rarrtl"] = 10518,
5207   ["latail"] = 10521,
5208   ["ratail"] = 10522,
5209   ["lAtail"] = 10523,
5210   ["rAtail"] = 10524,
5211   ["larrfs"] = 10525,
5212   ["rarrfs"] = 10526,
5213   ["larrbfs"] = 10527,
5214   ["rarrbfs"] = 10528,
```

```
["nwarhk"] = 10531,
["nearhk"] = 10532,
["hksearow"] = 10533,
["searhk"] = 10533,
["hkswarow"] = 10534,
["swarhk"] = 10534,
["nwnear"] = 10535,
["nesear"] = 10536,
["toea"] = 10536,
["seswar"] = 10537,
["tosa"] = 10537,
["swnwar"] = 10538,
["nrarrc"] = {10547, 824},
["rarrc"] = 10547,
["cudarrr"] = 10549,
["ldca"] = 10550,
["rdca"] = 10551,
["cudarrl"] = 10552,
["larrpl"] = 10553,
["curarrm"] = 10556,
["cularrp"] = 10557,
["rarrpl"] = 10565,
["harrcir"] = 10568,
["Uarrocir"] = 10569,
["lurdshar"] = 10570,
["ldrushar"] = 10571,
["LeftRightVector"] = 10574,
["RightUpDownVector"] = 10575,
["DownLeftRightVector"] = 10576,
["LeftUpDownVector"] = 10577,
["LeftVectorBar"] = 10578,
["RightVectorBar"] = 10579,
["RightUpVectorBar"] = 10580,
["RightDownVectorBar"] = 10581,
["DownLeftVectorBar"] = 10582,
["DownRightVectorBar"] = 10583,
["LeftUpVectorBar"] = 10584,
["LeftDownVectorBar"] = 10585,
["LeftTeeVector"] = 10586,
["RightTeeVector"] = 10587,
["RightUpTeeVector"] = 10588,
["RightDownTeeVector"] = 10589,
["DownLeftTeeVector"] = 10590,
["DownRightTeeVector"] = 10591,
["LeftUpTeeVector"] = 10592,
["LeftDownTeeVector"] = 10593,
["lHar"] = 10594,
```

```
5262   ["uHar"] = 10595,
5263   ["rHar"] = 10596,
5264   ["dHar"] = 10597,
5265   ["luruhar"] = 10598,
5266   ["ldrdhar"] = 10599,
5267   ["ruluhar"] = 10600,
5268   ["rdldhar"] = 10601,
5269   ["lharul"] = 10602,
5270   ["llhard"] = 10603,
5271   ["rharul"] = 10604,
5272   ["lrhard"] = 10605,
5273   ["UpEquilibrium"] = 10606,
5274   ["udhar"] = 10606,
5275   ["ReverseUpEquilibrium"] = 10607,
5276   ["duhar"] = 10607,
5277   ["RoundImplies"] = 10608,
5278   ["erarr"] = 10609,
5279   ["simrarr"] = 10610,
5280   ["larrsim"] = 10611,
5281   ["rarrsim"] = 10612,
5282   ["rarrap"] = 10613,
5283   ["ltlarr"] = 10614,
5284   ["gtrarr"] = 10616,
5285   ["subrarr"] = 10617,
5286   ["suplarr"] = 10619,
5287   ["lfisht"] = 10620,
5288   ["rfisht"] = 10621,
5289   ["ufisht"] = 10622,
5290   ["dfisht"] = 10623,
5291   ["lopar"] = 10629,
5292   ["ropar"] = 10630,
5293   ["lbrke"] = 10635,
5294   ["rbrke"] = 10636,
5295   ["lbrkslu"] = 10637,
5296   ["rbrksld"] = 10638,
5297   ["lbrksld"] = 10639,
5298   ["rbrkslu"] = 10640,
5299   ["langd"] = 10641,
5300   ["rangd"] = 10642,
5301   ["lparlt"] = 10643,
5302   ["rpargt"] = 10644,
5303   ["gtlPar"] = 10645,
5304   ["ltrPar"] = 10646,
5305   ["vzigzag"] = 10650,
5306   ["vangrt"] = 10652,
5307   ["angrtvbd"] = 10653,
5308   ["ange"] = 10660,
```

```
5309   ["range"] = 10661,
5310   ["dwangle"] = 10662,
5311   ["uwangle"] = 10663,
5312   ["angmsdaa"] = 10664,
5313   ["angmsdab"] = 10665,
5314   ["angmsdac"] = 10666,
5315   ["angmsdad"] = 10667,
5316   ["angmsdae"] = 10668,
5317   ["angmsdaf"] = 10669,
5318   ["angmsdag"] = 10670,
5319   ["angmsdah"] = 10671,
5320   ["bemptyv"] = 10672,
5321   ["demptyv"] = 10673,
5322   ["cemptyv"] = 10674,
5323   ["raemptyv"] = 10675,
5324   ["laemptyv"] = 10676,
5325   ["ohbar"] = 10677,
5326   ["omid"] = 10678,
5327   ["opar"] = 10679,
5328   ["operp"] = 10681,
5329   ["olcross"] = 10683,
5330   ["odsold"] = 10684,
5331   ["olcir"] = 10686,
5332   ["ofcir"] = 10687,
5333   ["olt"] = 10688,
5334   ["ogt"] = 10689,
5335   ["cirscir"] = 10690,
5336   ["cirE"] = 10691,
5337   ["solb"] = 10692,
5338   ["bsolb"] = 10693,
5339   ["boxbox"] = 10697,
5340   ["trisb"] = 10701,
5341   ["rtriltri"] = 10702,
5342   ["LeftTriangleBar"] = 10703,
5343   ["NotLeftTriangleBar"] = {10703, 824},
5344   ["NotRightTriangleBar"] = {10704, 824},
5345   ["RightTriangleBar"] = 10704,
5346   ["iinfin"] = 10716,
5347   ["infintie"] = 10717,
5348   ["nvinfin"] = 10718,
5349   ["eparsl"] = 10723,
5350   ["smeparsl"] = 10724,
5351   ["eqvparsl"] = 10725,
5352   ["blacklozenge"] = 10731,
5353   ["lozf"] = 10731,
5354   ["RuleDelayed"] = 10740,
5355   ["dsol"] = 10742,
```

```
5356   ["bigodot"] = 10752,
5357   ["xodot"] = 10752,
5358   ["bigoplus"] = 10753,
5359   ["xoplus"] = 10753,
5360   ["bigotimes"] = 10754,
5361   ["xotime"] = 10754,
5362   ["biguplus"] = 10756,
5363   ["xuplus"] = 10756,
5364   ["bigsqcup"] = 10758,
5365   ["xsqcup"] = 10758,
5366   ["iiiint"] = 10764,
5367   ["qint"] = 10764,
5368   ["fpartint"] = 10765,
5369   ["cirfnint"] = 10768,
5370   ["awint"] = 10769,
5371   ["rppolint"] = 10770,
5372   ["scpolint"] = 10771,
5373   ["npolint"] = 10772,
5374   ["pointint"] = 10773,
5375   ["quatint"] = 10774,
5376   ["intlarhk"] = 10775,
5377   ["pluscir"] = 10786,
5378   ["plusacir"] = 10787,
5379   ["simplus"] = 10788,
5380   ["plusdu"] = 10789,
5381   ["plussim"] = 10790,
5382   ["plustwo"] = 10791,
5383   ["mcomma"] = 10793,
5384   ["minusdu"] = 10794,
5385   ["loplus"] = 10797,
5386   ["roplus"] = 10798,
5387   ["Cross"] = 10799,
5388   ["timesd"] = 10800,
5389   ["timesbar"] = 10801,
5390   ["smashp"] = 10803,
5391   ["lotimes"] = 10804,
5392   ["rotimes"] = 10805,
5393   ["otimesas"] = 10806,
5394   ["Otimes"] = 10807,
5395   ["odiv"] = 10808,
5396   ["triplus"] = 10809,
5397   ["triminus"] = 10810,
5398   ["tritime"] = 10811,
5399   ["intprod"] = 10812,
5400   ["iprod"] = 10812,
5401   ["amalg"] = 10815,
5402   ["capdot"] = 10816,
```

```
5403   ["ncup"] = 10818,
5404   ["ncap"] = 10819,
5405   ["capand"] = 10820,
5406   ["cupor"] = 10821,
5407   ["cupcap"] = 10822,
5408   ["capcup"] = 10823,
5409   ["cupbrcap"] = 10824,
5410   ["capbrcup"] = 10825,
5411   ["cupcup"] = 10826,
5412   ["capcap"] = 10827,
5413   ["ccups"] = 10828,
5414   ["ccaps"] = 10829,
5415   ["ccupssm"] = 10832,
5416   ["And"] = 10835,
5417   ["Or"] = 10836,
5418   ["andand"] = 10837,
5419   ["oror"] = 10838,
5420   ["orslope"] = 10839,
5421   ["andslope"] = 10840,
5422   ["andv"] = 10842,
5423   ["orv"] = 10843,
5424   ["andd"] = 10844,
5425   ["ord"] = 10845,
5426   ["wedbar"] = 10847,
5427   ["sdote"] = 10854,
5428   ["simdot"] = 10858,
5429   ["congdot"] = 10861,
5430   ["ncongdot"] = {10861, 824},
5431   ["easter"] = 10862,
5432   ["apacir"] = 10863,
5433   ["apE"] = 10864,
5434   ["napE"] = {10864, 824},
5435   ["eplus"] = 10865,
5436   ["pluse"] = 10866,
5437   ["Esim"] = 10867,
5438   ["Colone"] = 10868,
5439   ["Equal"] = 10869,
5440   ["ddotseq"] = 10871,
5441   ["eDDot"] = 10871,
5442   ["equivDD"] = 10872,
5443   ["ltcir"] = 10873,
5444   ["gtcir"] = 10874,
5445   ["ltquest"] = 10875,
5446   ["gtquest"] = 10876,
5447   ["LessSlantEqual"] = 10877,
5448   ["NotLessSlantEqual"] = {10877, 824},
5449   ["leqslant"] = 10877,
```

```
5450    ["les"] = 10877,
5451    ["nleqslant"] = {10877, 824},
5452    ["nles"] = {10877, 824},
5453    ["GreaterSlantEqual"] = 10878,
5454    ["NotGreaterSlantEqual"] = {10878, 824},
5455    ["geqslant"] = 10878,
5456    ["ges"] = 10878,
5457    ["ngeqslant"] = {10878, 824},
5458    ["nges"] = {10878, 824},
5459    ["lesdot"] = 10879,
5460    ["gesdot"] = 10880,
5461    ["lesdoto"] = 10881,
5462    ["gesdoto"] = 10882,
5463    ["lesdotor"] = 10883,
5464    ["gesdotol"] = 10884,
5465    ["lap"] = 10885,
5466    ["lessapprox"] = 10885,
5467    ["gap"] = 10886,
5468    ["gtrapprox"] = 10886,
5469    ["lne"] = 10887,
5470    ["lneq"] = 10887,
5471    ["gne"] = 10888,
5472    ["gneq"] = 10888,
5473    ["lnap"] = 10889,
5474    ["lnapprox"] = 10889,
5475    ["gnap"] = 10890,
5476    ["gnapprox"] = 10890,
5477    ["lEg"] = 10891,
5478    ["lesseqqgtr"] = 10891,
5479    ["gEl"] = 10892,
5480    ["gtreqqless"] = 10892,
5481    ["lsime"] = 10893,
5482    ["gsime"] = 10894,
5483    ["lsimg"] = 10895,
5484    ["gsiml"] = 10896,
5485    ["lgE"] = 10897,
5486    ["glE"] = 10898,
5487    ["lesges"] = 10899,
5488    ["gesles"] = 10900,
5489    ["els"] = 10901,
5490    ["eqslantless"] = 10901,
5491    ["egs"] = 10902,
5492    ["eqslantgtr"] = 10902,
5493    ["elsdot"] = 10903,
5494    ["egsdot"] = 10904,
5495    ["el"] = 10905,
5496    ["eg"] = 10906,
```

```
  ["siml"] = 10909,
  ["simg"] = 10910,
  ["simlE"] = 10911,
  ["simgE"] = 10912,
  ["LessLess"] = 10913,
  ["NotNestedLessLess"] = {10913, 824},
  ["GreaterGreater"] = 10914,
  ["NotNestedGreaterGreater"] = {10914, 824},
  ["glj"] = 10916,
  ["gla"] = 10917,
  ["ltcc"] = 10918,
  ["gtcc"] = 10919,
  ["lescc"] = 10920,
  ["gescc"] = 10921,
  ["smt"] = 10922,
  ["lat"] = 10923,
  ["smte"] = 10924,
  ["smtes"] = {10924, 65024},
  ["late"] = 10925,
  ["lates"] = {10925, 65024},
  ["bumpE"] = 10926,
  ["NotPrecedesEqual"] = {10927, 824},
  ["PrecedesEqual"] = 10927,
  ["npre"] = {10927, 824},
  ["npreceq"] = {10927, 824},
  ["pre"] = 10927,
  ["preceq"] = 10927,
  ["NotSucceedsEqual"] = {10928, 824},
  ["SucceedsEqual"] = 10928,
  ["nsce"] = {10928, 824},
  ["nsucceq"] = {10928, 824},
  ["sce"] = 10928,
  ["succeq"] = 10928,
  ["prE"] = 10931,
  ["scE"] = 10932,
  ["precneqq"] = 10933,
  ["prnE"] = 10933,
  ["scnE"] = 10934,
  ["succneqq"] = 10934,
  ["prap"] = 10935,
  ["precapprox"] = 10935,
  ["scap"] = 10936,
  ["succapprox"] = 10936,
  ["precnapprox"] = 10937,
  ["prnap"] = 10937,
  ["scnap"] = 10938,
  ["succnapprox"] = 10938,
```

```
["Pr"] = 10939,
["Sc"] = 10940,
["subdot"] = 10941,
["supdot"] = 10942,
["subplus"] = 10943,
["supplus"] = 10944,
["submult"] = 10945,
["supmult"] = 10946,
["subedot"] = 10947,
["supedot"] = 10948,
["nsubE"] = {10949, 824},
["nsubseteqq"] = {10949, 824},
["subE"] = 10949,
["subseteqq"] = 10949,
["nsupE"] = {10950, 824},
["nsupseteqq"] = {10950, 824},
["supE"] = 10950,
["supseteqq"] = 10950,
["subsim"] = 10951,
["supsim"] = 10952,
["subnE"] = 10955,
["subsetneqq"] = 10955,
["varsubsetneqq"] = {10955, 65024},
["vsubnE"] = {10955, 65024},
["supnE"] = 10956,
["supsetneqq"] = 10956,
["varsupsetneqq"] = {10956, 65024},
["vsupnE"] = {10956, 65024},
["csub"] = 10959,
["csup"] = 10960,
["csube"] = 10961,
["csupe"] = 10962,
["subsup"] = 10963,
["supsub"] = 10964,
["subsub"] = 10965,
["supsup"] = 10966,
["suphsub"] = 10967,
["supdsub"] = 10968,
["forkv"] = 10969,
["topfork"] = 10970,
["mlcp"] = 10971,
["Dashv"] = 10980,
["DoubleLeftTee"] = 10980,
["Vdashl"] = 10982,
["Barv"] = 10983,
["vBar"] = 10984,
["vBarv"] = 10985,
```

```
  ["Vbar"] = 10987,
  ["Not"] = 10988,
  ["bNot"] = 10989,
  ["rnmid"] = 10990,
  ["cirmid"] = 10991,
  ["midcir"] = 10992,
  ["topcir"] = 10993,
  ["nhpar"] = 10994,
  ["parsim"] = 10995,
  ["nparsl"] = {11005, 8421},
  ["parsl"] = 11005,
  ["fflig"] = 64256,
  ["filig"] = 64257,
  ["fllig"] = 64258,
  ["ffilig"] = 64259,
  ["ffllig"] = 64260,
  ["Ascr"] = 119964,
  ["Cscr"] = 119966,
  ["Dscr"] = 119967,
  ["Gscr"] = 119970,
  ["Jscr"] = 119973,
  ["Kscr"] = 119974,
  ["Nscr"] = 119977,
  ["Oscr"] = 119978,
  ["Pscr"] = 119979,
  ["Qscr"] = 119980,
  ["Sscr"] = 119982,
  ["Tscr"] = 119983,
  ["Uscr"] = 119984,
  ["Vscr"] = 119985,
  ["Wscr"] = 119986,
  ["Xscr"] = 119987,
  ["Yscr"] = 119988,
  ["Zscr"] = 119989,
  ["ascr"] = 119990,
  ["bscr"] = 119991,
  ["cscr"] = 119992,
  ["dscr"] = 119993,
  ["fscr"] = 119995,
  ["hscr"] = 119997,
  ["iscr"] = 119998,
  ["jscr"] = 119999,
  ["kscr"] = 120000,
  ["lscr"] = 120001,
  ["mscr"] = 120002,
  ["nscr"] = 120003,
  ["pscr"] = 120005,
```

```
  ["qscr"] = 120006,
  ["rscr"] = 120007,
  ["sscr"] = 120008,
  ["tscr"] = 120009,
  ["uscr"] = 120010,
  ["vscr"] = 120011,
  ["wscr"] = 120012,
  ["xscr"] = 120013,
  ["yscr"] = 120014,
  ["zscr"] = 120015,
  ["Afr"] = 120068,
  ["Bfr"] = 120069,
  ["Dfr"] = 120071,
  ["Efr"] = 120072,
  ["Ffr"] = 120073,
  ["Gfr"] = 120074,
  ["Jfr"] = 120077,
  ["Kfr"] = 120078,
  ["Lfr"] = 120079,
  ["Mfr"] = 120080,
  ["Nfr"] = 120081,
  ["Ofr"] = 120082,
  ["Pfr"] = 120083,
  ["Qfr"] = 120084,
  ["Sfr"] = 120086,
  ["Tfr"] = 120087,
  ["Ufr"] = 120088,
  ["Vfr"] = 120089,
  ["Wfr"] = 120090,
  ["Xfr"] = 120091,
  ["Yfr"] = 120092,
  ["afr"] = 120094,
  ["bfr"] = 120095,
  ["cfr"] = 120096,
  ["dfr"] = 120097,
  ["efr"] = 120098,
  ["ffr"] = 120099,
  ["gfr"] = 120100,
  ["hfr"] = 120101,
  ["ifr"] = 120102,
  ["jfr"] = 120103,
  ["kfr"] = 120104,
  ["lfr"] = 120105,
  ["mfr"] = 120106,
  ["nfr"] = 120107,
  ["ofr"] = 120108,
  ["pfr"] = 120109,
```

```
5685   ["qfr"] = 120110,
5686   ["rfr"] = 120111,
5687   ["sfr"] = 120112,
5688   ["tfr"] = 120113,
5689   ["ufr"] = 120114,
5690   ["vfr"] = 120115,
5691   ["wfr"] = 120116,
5692   ["xfr"] = 120117,
5693   ["yfr"] = 120118,
5694   ["zfr"] = 120119,
5695   ["Aopf"] = 120120,
5696   ["Bopf"] = 120121,
5697   ["Dopf"] = 120123,
5698   ["Eopf"] = 120124,
5699   ["Fopf"] = 120125,
5700   ["Gopf"] = 120126,
5701   ["Iopf"] = 120128,
5702   ["Jopf"] = 120129,
5703   ["Kopf"] = 120130,
5704   ["Lopf"] = 120131,
5705   ["Mopf"] = 120132,
5706   ["Oopf"] = 120134,
5707   ["Sopf"] = 120138,
5708   ["Topf"] = 120139,
5709   ["Uopf"] = 120140,
5710   ["Vopf"] = 120141,
5711   ["Wopf"] = 120142,
5712   ["Xopf"] = 120143,
5713   ["Yopf"] = 120144,
5714   ["aopf"] = 120146,
5715   ["bopf"] = 120147,
5716   ["copf"] = 120148,
5717   ["dopf"] = 120149,
5718   ["eopf"] = 120150,
5719   ["fopf"] = 120151,
5720   ["gopf"] = 120152,
5721   ["hopf"] = 120153,
5722   ["iopf"] = 120154,
5723   ["jopf"] = 120155,
5724   ["kopf"] = 120156,
5725   ["lopf"] = 120157,
5726   ["mopf"] = 120158,
5727   ["nopf"] = 120159,
5728   ["oopf"] = 120160,
5729   ["popf"] = 120161,
5730   ["qopf"] = 120162,
5731   ["ropf"] = 120163,
```

```
  ["sopf"] = 120164,
  ["topf"] = 120165,
  ["uopf"] = 120166,
  ["vopf"] = 120167,
  ["wopf"] = 120168,
  ["xopf"] = 120169,
  ["yopf"] = 120170,
  ["zopf"] = 120171,
}
```

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

```
function entities.dec_entity(s)
  local n = tonumber(s)
  if n == nil then
    return "&#" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.hex_entity(s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#x" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```
function entities.hex_entity_with_x_char(x, s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#" .. x .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.char_entity(s)
  local code_points = character_entities[s]
  if code_points == nil then
    return "&" .. s .. ";"
  end
```

```
5767   if type(code_points) ~= 'table' then
5768     code_points = {code_points}
5769   end
5770   local char_table = {}
5771     for _, code_point in ipairs(code_points) do
5772       table.insert(char_table, unicode.utf8.char(code_point))
5773     end
5774   return table.concat(char_table)
5775 end
```

3.1.3 Plain TEX Writer

This section documents the `writer` object, which implements the routines for producing the TEX output. The object is an amalgamate of the generic, TEX, LATEX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

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

```
5776 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.

```
5777 function M.writer.new(options)
5778   local self = {}
```

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

```
5779   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.

```
5780   self.flatten_inlines = false
```

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

```
5781   local slice_specifiers = {}
5782   for specifier in options.slice:gmatch("[^%s]+") do
```

```
5783      table.insert(slice_specifiers, specifier)
5784    end
5785
5786    if #slice_specifiers == 2 then
5787      self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5788      local slice_begin_type = self.slice_begin:sub(1, 1)
5789      if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5790        self.slice_begin = "^" .. self.slice_begin
5791      end
5792      local slice_end_type = self.slice_end:sub(1, 1)
5793      if slice_end_type ~= "^" and slice_end_type ~= "$" then
5794        self.slice_end = "$" .. self.slice_end
5795      end
5796    elseif #slice_specifiers == 1 then
5797      self.slice_begin = "^" .. slice_specifiers[1]
5798      self.slice_end = "$" .. slice_specifiers[1]
5799    end
5800
5801    self.slice_begin_type = self.slice_begin:sub(1, 1)
5802    self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5803    self.slice_end_type = self.slice_end:sub(1, 1)
5804    self.slice_end_identifier = self.slice_end:sub(2) or ""
5805
5806    if self.slice_begin == "^" and self.slice_end ~= "^" then
5807      self.is_writing = true
5808    else
5809      self.is_writing = false
5810    end
```

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

```
5811    self.space = " "
```

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

```
5812    self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5813    function self.plain(s)
5814      return s
5815    end
```

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

```
5816    function self.paragraph(s)
5817      if not self.is_writing then return "" end
5818      return s
5819    end
```

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

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

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

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

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

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

Define `writer->soft_line_break` as the output format of a soft line break.

```
  self.soft_line_break = function()
    if self.flatten_inlines then return "\n" end
    return "\\markdownRendererSoftLineBreak\n{}"
  end
```

Define `writer->hard_line_break` as the output format of a hard line break.

```
  self.hard_line_break = function()
    if self.flatten_inlines then return "\n" end
    return "\\markdownRendererHardLineBreak\n{}"
  end
```

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

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

Define `writer->thematic_break` as the output format of a thematic break.

```
  function self.thematic_break()
    if not self.is_writing then return "" end
    return "\\markdownRendererThematicBreak{}"
  end
```

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```
  self.escaped_uri_chars = {
```

```
    ["{"] = "\\markdownRendererLeftBrace{}",
    ["}"] = "\\markdownRendererRightBrace{}",
    ["\\"] = "\\markdownRendererBackslash{}",
    ["\r"] = " ",
    ["\n"] = " ",
  }
  self.escaped_minimal_strings = {
    ["^^"] = "\\markdownRendererCircumflex"
          .. "\\markdownRendererCircumflex ",
    ["☒"] = "\\markdownRendererTickedBox{}",
    ["☐"] = "\\markdownRendererHalfTickedBox{}",
    ["▫"] = "\\markdownRendererUntickedBox{}",
    [entities.hex_entity('FFFD')]
      = "\\markdownRendererReplacementCharacter{}",
  }
```

Define table `writer->escaped_strings` containing the mapping from character strings that need to be escaped in typeset content.

```
  self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
  self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
  self.escaped_chars = {
    ["{"] = "\\markdownRendererLeftBrace{}",
    ["}"] = "\\markdownRendererRightBrace{}",
    ["%"] = "\\markdownRendererPercentSign{}",
    ["\\"] = "\\markdownRendererBackslash{}",
    ["#"] = "\\markdownRendererHash{}",
    ["$"] = "\\markdownRendererDollarSign{}",
    ["&"] = "\\markdownRendererAmpersand{}",
    ["_"] = "\\markdownRendererUnderscore{}",
    ["^"] = "\\markdownRendererCircumflex{}",
    ["~"] = "\\markdownRendererTilde{}",
    ["|"] = "\\markdownRendererPipe{}",
    [entities.hex_entity('0000')]
      = "\\markdownRendererReplacementCharacter{}",
  }
```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minima` 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
    self.uri = escape_minimal
    self.infostring = escape_minimal
  else
    self.identifier = escape_programmatic_text
    self.string = escape_typographic_text
    self.uri = escape_programmatic_text
    self.infostring = escape_programmatic_text
  end
```

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

```
  function self.warning(t)
    return {"\\markdownRendererWarning{", self.identifier(t), "}"}
  end
```

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

```
  function self.error(t)
    return {"\\markdownRendererError{", self.identifier(t), "}"}
  end
```

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

```
function self.code(s, attributes)
  if self.flatten_inlines then return s end
  local buf = {}
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererCodeSpanAttributeContextBegin\n")
    table.insert(buf, self.attributes(attributes))
  end
  table.insert(buf,
               {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererCodeSpanAttributeContextEnd{}")
  end
  return buf
end
```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```
function self.link(lab, src, tit, attributes)
  if self.flatten_inlines then return lab end
  local buf = {}
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererLinkAttributeContextBegin\n")
    table.insert(buf, self.attributes(attributes))
  end
  table.insert(buf, {"\\markdownRendererLink{",lab,"}",
                     "{",self.escape(src),"}",
                     "{",self.uri(src),"}",
                     "{",self.string(tit or ""),"}"})
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererLinkAttributeContextEnd{}")
  end
  return buf
end
```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```
function self.image(lab, src, tit, attributes)
  if self.flatten_inlines then return lab end
  local buf = {}
  if attributes ~= nil then
    table.insert(buf,
```

```
                    "\\markdownRendererImageAttributeContextBegin\n")
      table.insert(buf, self.attributes(attributes))
    end
    table.insert(buf, {"\\markdownRendererImage{",lab,"}",
                       "{",self.string(src),"}",
                       "{",self.uri(src),"}",
                       "{",self.string(tit or ""),"}"})
    if attributes ~= nil then
      table.insert(buf,
                    "\\markdownRendererImageAttributeContextEnd{}")
    end
    return buf
  end
```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```
  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
```

```
5988      local buffer = {}
5989      local num = startnum
5990      for _,item in ipairs(items) do
5991        if item ~= "" then
5992          buffer[#buffer + 1] = self.ordereditem(item,num)
5993        end
5994        if num ~= nil and item ~= "" then
5995          num = num + 1
5996        end
5997      end
5998      local contents = util.intersperse(buffer,"\n")
5999      if tight and options.tightLists then
6000        return {"\\markdownRendererOlBeginTight\n",contents,
6001                "\n\\markdownRendererOlEndTight "}
6002      else
6003        return {"\\markdownRendererOlBegin\n",contents,
6004                "\n\\markdownRendererOlEnd "}
6005      end
6006    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.

```
6007    function self.ordereditem(s,num)
6008      if num ~= nil then
6009        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
6010                "\\markdownRendererOlItemEnd "}
6011      else
6012        return {"\\markdownRendererOlItem ",s,
6013                "\\markdownRendererOlItemEnd "}
6014      end
6015    end
```

Define `writer->inline_html_comment` as a function that will transform the contents of an inline HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```
6016    function self.inline_html_comment(contents)
6017      if self.flatten_inlines then return contents end
6018      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
6019    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.

```
6020    function self.inline_html_tag(contents)
6021      if self.flatten_inlines then return contents end
6022      return {"\\markdownRendererInlineHtmlTag{",
6023              self.string(contents),"}"}
```

```
6024    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.

```
6025    function self.block_html_element(s)
6026      if not self.is_writing then return "" end
6027      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6028      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
6029    end
```

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

```
6030    function self.emphasis(s)
6031      if self.flatten_inlines then return s end
6032      return {"\\markdownRendererEmphasis{",s,"}"}
6033    end
```

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

```
6034    function self.tickbox(f)
6035      if f == 1.0 then
6036        return "☒ "
6037      elseif f == 0.0 then
6038        return "▫ "
6039      else
6040        return "⊡ "
6041      end
6042    end
```

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

```
6043    function self.strong(s)
6044      if self.flatten_inlines then return s end
6045      return {"\\markdownRendererStrongEmphasis{",s,"}"}
6046    end
```

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

```
6047    function self.blockquote(s)
6048      if not self.is_writing then return "" end
6049      return {"\\markdownRendererBlockQuoteBegin\n",s,
6050        "\\markdownRendererBlockQuoteEnd "}
6051    end
```

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

```
6052    function self.verbatim(s)
6053      if not self.is_writing then return "" end
```

```
6054      s = s:gsub("\n$", "")
6055      local name = util.cache_verbatim(options.cacheDir, s)
6056      return {"\\markdownRendererInputVerbatim{",name,"}"}
6057    end
```

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

```
6058    function self.document(d)
6059      local buf = {"\\markdownRendererDocumentBegin\n", d}
6060
6061      -- pop all attributes
6062      table.insert(buf, self.pop_attributes())
6063
6064      table.insert(buf, "\\markdownRendererDocumentEnd")
6065
6066      return buf
6067    end
```

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

```
6068    local seen_identifiers = {}
6069    local key_value_regex = "([^= ]+)%s*=%s*(.*)"
6070    local function normalize_attributes(attributes, auto_identifiers)
6071      -- normalize attributes
6072      local normalized_attributes = {}
6073      local has_explicit_identifiers = false
6074      local key, value
6075      for _, attribute in ipairs(attributes or {}) do
6076        if attribute:sub(1, 1) == "#" then
6077          table.insert(normalized_attributes, attribute)
6078          has_explicit_identifiers = true
6079          seen_identifiers[attribute:sub(2)] = true
6080        elseif attribute:sub(1, 1) == "." then
6081          table.insert(normalized_attributes, attribute)
6082        else
6083          key, value = attribute:match(key_value_regex)
6084          if key:lower() == "id" then
6085            table.insert(normalized_attributes, "#" .. value)
6086          elseif key:lower() == "class" then
6087            local classes = {}
6088            for class in value:gmatch("%S+") do
6089              table.insert(classes, class)
6090            end
6091            table.sort(classes)
6092            for _, class in ipairs(classes) do
6093              table.insert(normalized_attributes, "." .. class)
6094            end
6095          else
```

```
          table.insert(normalized_attributes, attribute)
        end
      end
    end

    -- if no explicit identifiers exist, add auto identifiers
    if not has_explicit_identifiers and auto_identifiers ~= nil then
      local seen_auto_identifiers = {}
      for _, auto_identifier in ipairs(auto_identifiers) do
        if seen_auto_identifiers[auto_identifier] == nil then
          seen_auto_identifiers[auto_identifier] = true
          if seen_identifiers[auto_identifier] == nil then
            seen_identifiers[auto_identifier] = true
            table.insert(normalized_attributes,
                         "#" .. auto_identifier)
          else
            local auto_identifier_number = 1
            while true do
              local numbered_auto_identifier = auto_identifier .. "-"
                                               .. auto_identifier_number
              if seen_identifiers[numbered_auto_identifier] == nil then
                seen_identifiers[numbered_auto_identifier] = true
                table.insert(normalized_attributes,
                             "#" .. numbered_auto_identifier)
                break
              end
              auto_identifier_number = auto_identifier_number + 1
            end
          end
        end
      end
    end

    -- sort and deduplicate normalized attributes
    table.sort(normalized_attributes)
    local seen_normalized_attributes = {}
    local deduplicated_normalized_attributes = {}
    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
```

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

```
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`.

```
self.attribute_type_levels = {}
setmetatable(self.attribute_type_levels,
             { __index = function() return 0 end })
```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```
local function apply_attributes()
  local buf = {}
  for i = 1, #self.active_attributes do
    local start_output = self.active_attributes[i][3]
    if start_output ~= nil then
      table.insert(buf, start_output)
    end
  end
```

```
6181      return buf
6182    end
6183
6184    local function tear_down_attributes()
6185      local buf = {}
6186      for i = #self.active_attributes, 1, -1 do
6187        local end_output = self.active_attributes[i][4]
6188        if end_output ~= nil then
6189          table.insert(buf, end_output)
6190        end
6191      end
6192      return buf
6193    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.

```
6194    function self.push_attributes(attribute_type, attributes,
6195                                  start_output, end_output)
6196      local attribute_type_level
6197        = self.attribute_type_levels[attribute_type]
6198      self.attribute_type_levels[attribute_type]
6199        = attribute_type_level + 1
6200
6201      -- index attributes in a hash table for easy lookup
6202      attributes = attributes or {}
6203      for i = 1, #attributes do
6204        attributes[attributes[i]] = true
6205      end
6206
6207      local buf = {}
6208      -- handle slicing
6209      if attributes["#" .. self.slice_end_identifier] ~= nil and
6210         self.slice_end_type == "^" then
6211        if self.is_writing then
6212          table.insert(buf, self.undosep())
6213          table.insert(buf, tear_down_attributes())
6214        end
6215        self.is_writing = false
6216      end
6217      if attributes["#" .. self.slice_begin_identifier] ~= nil and
6218         self.slice_begin_type == "^" then
6219        table.insert(buf, apply_attributes())
6220        self.is_writing = true
6221      end
```

```
6222      if self.is_writing and start_output ~= nil then
6223        table.insert(buf, start_output)
6224      end
6225      table.insert(self.active_attributes,
6226                   {attribute_type, attributes,
6227                    start_output, end_output})
6228      return buf
6229    end
6230
```

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`).

```
6231    function self.pop_attributes(attribute_type)
6232      local buf = {}
6233      -- pop attributes until we find attributes of correct type
6234      -- or until no attributes remain
6235      local current_attribute_type = false
6236      while current_attribute_type ~= attribute_type and
6237            #self.active_attributes > 0 do
6238        local attributes, _, end_output
6239        current_attribute_type, attributes, _, end_output = table.unpack(
6240          self.active_attributes[#self.active_attributes])
6241        local attribute_type_level
6242          = self.attribute_type_levels[current_attribute_type]
6243        self.attribute_type_levels[current_attribute_type]
6244          = attribute_type_level - 1
6245        if self.is_writing and end_output ~= nil then
6246          table.insert(buf, end_output)
6247        end
6248        table.remove(self.active_attributes, #self.active_attributes)
6249        -- handle slicing
6250        if attributes["#" .. self.slice_end_identifier] ~= nil
6251           and self.slice_end_type == "$" then
6252          if self.is_writing then
6253            table.insert(buf, self.undosep())
6254            table.insert(buf, tear_down_attributes())
6255          end
6256          self.is_writing = false
6257        end
6258        if attributes["#" .. self.slice_begin_identifier] ~= nil and
6259           self.slice_begin_type == "$" then
6260          self.is_writing = true
6261          table.insert(buf, apply_attributes())
```

```
      end
    end
    return buf
  end
```

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 = uni_algos.normalize.NFC(normalized_s)
    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 = unicode.utf8.match(char, "%a")
        if is_letter then
          letter_found = true
        else
          goto continue
        end
      end

      -- Remove all non-alphanumeric characters, except underscores,
      -- hyphens, and periods.
      if not unicode.utf8.match(char, "[%w_%-%.%s]") then
        goto continue
      end

      -- Replace all spaces and newlines with hyphens.
      if unicode.utf8.match(char, "[%s\n]") then
        char = "-"
        if prev_space then
          goto continue
        else
          prev_space = true
        end
      else
        -- Convert all alphabetic characters to lowercase.
        char = unicode.utf8.lower(char)
        prev_space = false
      end
```

```
6308
6309        table.insert(buffer, char)
6310
6311        ::continue::
6312      end
6313
6314      if prev_space then
6315        table.remove(buffer)
6316      end
6317
6318      local identifier = #buffer == 0 and "section"
6319                      or table.concat(buffer, "")
6320      return identifier
6321    end
```

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

```
6322    local function create_gfm_auto_identifier(s)
6323      local buffer = {}
6324      local prev_space = false
6325      local letter_found = false
6326      local normalized_s = s
6327      if not options.unicodeNormalization
6328         or options.unicodeNormalizationForm ~= "nfc" then
6329        normalized_s = uni_algos.normalize.NFC(normalized_s)
6330      end
6331
6332      for _, code in utf8.codes(normalized_s) do
6333        local char = utf8.char(code)
6334
6335        -- Remove everything up to the first non-space.
6336        if not letter_found then
6337          local is_letter = unicode.utf8.match(char, "%S")
6338          if is_letter then
6339            letter_found = true
6340          else
6341            goto continue
6342          end
6343        end
6344
6345        -- Remove all non-alphanumeric characters, except underscores
6346        -- and hyphens.
6347        if not unicode.utf8.match(char, "[%w_%-%s]") then
6348          prev_space = false
6349          goto continue
6350        end
6351
```

```
      -- Replace all spaces and newlines with hyphens.
      if unicode.utf8.match(char, "[%s\n]") then
        char = "-"
        if prev_space then
          goto continue
        else
          prev_space = true
        end
      else
        -- Convert all alphabetic characters to lowercase.
        char = unicode.utf8.lower(char)
        prev_space = false
      end

      table.insert(buffer, char)

      ::continue::
    end

    if prev_space then
      table.remove(buffer)
    end

    local identifier = #buffer == 0 and "section"
                    or table.concat(buffer, "")
    return identifier
  end
```

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

```
  self.secbegin_text = "\\markdownRendererSectionBegin\n"
  self.secend_text = "\n\\markdownRendererSectionEnd "
  function self.heading(s, level, attributes)
    local buf = {}
    local flat_text, inlines = table.unpack(s)

    -- 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
```

```
6396        table.insert(buf, self.pop_attributes("heading"))
6397      end
6398
6399      -- construct attributes for the new section
6400      local auto_identifiers = {}
6401      if self.options.autoIdentifiers then
6402        table.insert(auto_identifiers, create_auto_identifier(flat_text))
6403      end
6404      if self.options.gfmAutoIdentifiers then
6405        table.insert(auto_identifiers,
6406                     create_gfm_auto_identifier(flat_text))
6407      end
6408      local normalized_attributes = normalize_attributes(attributes,
6409                                                         auto_identifiers)
6410
6411      -- push attributes for the new section
6412      local start_output = {}
6413      local end_output = {}
6414      table.insert(start_output, self.secbegin_text)
6415      table.insert(end_output, self.secend_text)
6416
6417      table.insert(buf, self.push_attributes("heading",
6418                                             normalized_attributes,
6419                                             start_output,
6420                                             end_output))
6421      assert(self.attribute_type_levels["heading"] == level)
6422
6423      -- render the heading and its attributes
6424      if self.is_writing and #normalized_attributes > 0 then
6425        table.insert(buf,
6426                     "\\markdownRendererHeaderAttributeContextBegin\n")
6427        table.insert(buf, self.attributes(normalized_attributes, false))
6428      end
6429
6430      local cmd
6431      level = level + options.shiftHeadings
6432      if level <= 1 then
6433        cmd = "\\markdownRendererHeadingOne"
6434      elseif level == 2 then
6435        cmd = "\\markdownRendererHeadingTwo"
6436      elseif level == 3 then
6437        cmd = "\\markdownRendererHeadingThree"
6438      elseif level == 4 then
6439        cmd = "\\markdownRendererHeadingFour"
6440      elseif level == 5 then
6441        cmd = "\\markdownRendererHeadingFive"
6442      elseif level >= 6 then
```

```
6443        cmd = "\\markdownRendererHeadingSix"
6444      else
6445        cmd = ""
6446      end
6447      if self.is_writing then
6448        table.insert(buf, {cmd, "{", inlines, "}"})
6449      end
6450
6451      if self.is_writing and #normalized_attributes > 0 then
6452        table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
6453      end
6454
6455      return buf
6456    end
```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```
6457    function self.get_state()
6458      return {
6459        is_writing=self.is_writing,
6460        flatten_inlines=self.flatten_inlines,
6461        active_attributes={table.unpack(self.active_attributes)},
6462      }
6463    end
```

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

```
6464    function self.set_state(s)
6465      local previous_state = self.get_state()
6466      for key, value in pairs(s) do
6467        self[key] = value
6468      end
6469      return previous_state
6470    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`.

```
6471    function self.defer_call(f)
6472      local previous_state = self.get_state()
6473      return function(...)
6474        local state = self.set_state(previous_state)
6475        local return_value = f(...)
6476        self.set_state(state)
6477        return return_value
6478      end
6479    end
6480
```

```
  return self
end
```

3.1.4 Parsers

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

```
local parsers                 = {}
```

3.1.4.1 Basic Parsers

```
parsers.percent               = P("%")
parsers.at                    = P("@")
parsers.comma                 = P(",")
parsers.asterisk              = P("*")
parsers.dash                  = P("-")
parsers.plus                  = P("+")
parsers.underscore            = P("_")
parsers.period                = P(".")
parsers.hash                  = P("#")
parsers.dollar                = P("$")
parsers.ampersand             = P("&")
parsers.backtick              = P("`")
parsers.less                  = P("<")
parsers.more                  = P(">")
parsers.space                 = P(" ")
parsers.squote                = P("'")
parsers.dquote                = P('"')
parsers.lparent               = P("(")
parsers.rparent               = P(")")
parsers.lbracket              = P("[")
parsers.rbracket              = P("]")
parsers.lbrace                = P("{")
parsers.rbrace                = P("}")
parsers.circumflex            = P("^")
parsers.slash                 = P("/")
parsers.equal                 = P("=")
parsers.colon                 = P(":")
parsers.semicolon             = P(";")
parsers.exclamation           = P("!")
parsers.pipe                  = P("|")
parsers.tilde                 = P("~")
parsers.backslash             = P("\\")
parsers.tab                   = P("\t")
parsers.newline               = P("\n")

parsers.digit                 = R("09")
```

```
parsers.hexdigit               = R("09","af","AF")
parsers.letter                 = R("AZ","az")
parsers.alphanumeric           = R("AZ","az","09")
parsers.keyword                = parsers.letter
                               * (parsers.alphanumeric + parsers.dash)^0

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

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

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

3.1.5 Unicode punctuation

This section documents the Unicode punctuation[33] recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

```
(CommonMark Spec, Version 0.31.2 (2024-01-28))
```

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

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

```
  local prefix_trees = {}
  for line in file:lines() do
    local codepoint, major_category = line:match("^(%x+);[^;]*;(%a)")
    if major_category == "P" or major_category == "S" then
      local code = unicode.utf8.char(tonumber(codepoint, 16))
      if prefix_trees[#code] == nil then
        prefix_trees[#code] = {}
      end
      local node = prefix_trees[#code]
      for i = 1, #code do
```

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

```
        local byte = code:sub(i, i)
        if i < #code then
          if node[byte] == nil then
            node[byte] = {}
          end
          node = node[byte]
        else
          table.insert(node, byte)
        end
      end
    end
  end
  assert(file:close())
```

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

```
  local function depth_first_search(node, path, visit, leave)
    visit(node, path)
    for label, child in pairs(node) do
      if type(child) == "table" then
        depth_first_search(child, path .. label, visit, leave)
      else
        visit(child, path)
      end
    end
    leave(node, path)
  end

  parsers.punctuation = {}
  for length, prefix_tree in pairs(prefix_trees) do
    local subparsers = {}
    depth_first_search(prefix_tree, "", function(node, path)
      if type(node) == "table" then
        subparsers[path] = parsers.fail
      else
        assert(type(node) == "string")
        subparsers[path] = subparsers[path] + S(node)
      end
    end, function(_, path)
      if #path > 0 then
        local byte = path:sub(#path, #path)
        local parent_path = path:sub(1, #path-1)
        subparsers[parent_path] = subparsers[parent_path]
                                + S(byte) * subparsers[path]
      else
        parsers.punctuation[length] = subparsers[path]
      end
    end)
    assert(parsers.punctuation[length] ~= nil)
```

```
  end
end)()

parsers.escapable               = parsers.ascii_punctuation
parsers.anyescaped              = parsers.backslash / ""
                                * parsers.escapable
                                + parsers.any

parsers.spacechar               = S("\t ")
parsers.spacing                 = S(" \n\r\t")
parsers.nonspacechar            = parsers.any - parsers.spacing
parsers.optionalspace           = parsers.spacechar^0

parsers.normalchar              = parsers.any - (V("SpecialChar")
                                              + parsers.spacing)
parsers.eof                     = -parsers.any
parsers.nonindentspace          = parsers.space^-3 * - parsers.spacechar
parsers.indent                  = parsers.space^-3 * parsers.tab
                                + parsers.fourspaces / ""
parsers.linechar                = P(1 - parsers.newline)

parsers.blankline               = parsers.optionalspace
                                * parsers.newline / "\n"
parsers.blanklines              = parsers.blankline^0
parsers.skipblanklines          = ( parsers.optionalspace
                                  * parsers.newline)^0
parsers.indentedline            = parsers.indent    /""
                                * C( parsers.linechar^1
                                   * parsers.newline^-1)
parsers.optionallyindentedline = parsers.indent^-1 /""
                                * C( parsers.linechar^1
                                   * parsers.newline^-1)
parsers.sp                      = parsers.spacing^0
parsers.spnl                    = parsers.optionalspace
                                * ( parsers.newline
                                  * parsers.optionalspace)^-1
parsers.line                    = parsers.linechar^0 * parsers.newline
parsers.nonemptyline            = parsers.line - parsers.blankline
```

3.1.5.1 Parsers Used for Indentation

```

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 = {}

  local blank_starter = left_blank_starter(indent_table)

  if current_line_indents == nil then
    current_line_indents = {}
  end

  for index = 1,#indent_table.indents do
    local value = indent_table.indents[index]
    local pattern = decode_pattern(value.name)

    -- match decoded pattern
    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
    if new_indent_info == nil then
```

```
      local blankline_end = lpeg.match(
        Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
      if is_optional or not indent_table.ignore_blockquote_blank
         or not blankline_end then
        return is_optional, new_index, current_trail,
               current_line_indents
      end

      return traverse_indent(s, tonumber(blankline_end.pos),
                             indent_table, is_optional, is_blank,
                             current_line_indents)
    end

    local raw_last_trail = new_indent_info[1]
    local delimiter = new_indent_info[2]
    local raw_new_trail = new_indent_info[3]
    local next_index = new_indent_info[4]

    local space_only = delimiter == ""

    -- check previous trail
    if not space_only and next(current_trail) == nil then
      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
      current_trail = {is_code=sp.is_code, remainder=sp.remainder,
                       total_length=sp.total_length,
                       full_remainder=sp.full_remainder}
    end

    if next(current_trail) ~= nil then
      if not space_only and current_trail.is_code then
        return is_optional, new_index, current_trail,
               current_line_indents
      end
      if current_trail.internal_remainder ~= nil then
        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}

    current_line_indents[#current_line_indents + 1] = new_indent_info
    new_index = next_index
  end

  return true, new_index, current_trail, current_line_indents
end

```

Check if a code trail is expected.

```
local function check_trail(expect_code, is_code)
```

```
  return (expect_code and is_code) or (not expect_code and not is_code)
end
```

Check if the current trail of the `indent_table` would produce code if it is expected `expect_code` or it would not if it is not. If there is no trail, process and check the current spacing `spacing`.

```
local check_trail_joined =
  function(s, i, indent_table, -- luacheck: ignore s i
           spacing, expect_code, omit_remainder)
    local is_code
    local remainder

    if has_trail(indent_table) then
      local trail = indent_table.trail
      is_code = trail.is_code
      if is_code then
        remainder = trail.remainder
      else
        remainder = trail.full_remainder
      end
    else
      local sp = process_starter_spacing(0, spacing, 0, 0)
      is_code = sp.is_code
      if is_code then
        remainder = sp.remainder
      else
        remainder = sp.full_remainder
      end
    end

    local result = check_trail(expect_code, is_code)
    if omit_remainder then
      return result
    end
    return result, remainder
  end
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
local check_trail_length =
  function(s, i, indent_table, -- luacheck: ignore s i
           spacing, min, max)
    local trail

    if has_trail(indent_table) then
      trail = indent_table.trail
    else
```

```
      trail = process_starter_spacing(0, spacing, 0, 0)
    end

    local total_length = trail.total_length
    if total_length == nil then
      return false
    end

    return min <= total_length and total_length <= max
  end

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
local function check_continuation_indentation(s, i, indent_table,
                                              is_optional, is_blank)
  if not has_indents(indent_table) then
    return true
  end

  local passes, new_index, current_trail, current_line_indents =
    traverse_indent(s, i, indent_table, is_optional, is_blank)

  if passes then
    indent_table.current_line_indents = current_line_indents
    indent_table = add_trail(indent_table, current_trail)
    return new_index, indent_table
  end
  return false
end

```

Get name of the last indent from the `indent_table`.

```
local function get_last_indent_name(indent_table)
  if has_indents(indent_table) then
    return indent_table.indents[#indent_table.indents].name
  end
end

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
local function remove_remainder_if_blank(indent_table, remainder)
  if get_last_indent_name(indent_table) == "li" then
    return ""
  end
  return remainder
end
```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```
local check_trail_type =
  function(s, i, -- luacheck: ignore s i
           trail, spacing, trail_type)
    if trail == nil then
      trail = process_starter_spacing(0, spacing, 0, 0)
    end

    if trail_type == "non-code" then
      return check_trail(false, trail.is_code)
    end
    if trail_type == "code" then
      return check_trail(true, trail.is_code)
    end
    if trail_type == "full-code" then
      if (trail.is_code) then
        return i, trail.remainder
      end
      return i, ""
    end
    if trail_type == "full-any" then
      return i, trail.internal_remainder
    end
  end

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
local trail_freezing =
  function(s, i, -- luacheck: ignore s i
           indent_table, is_freezing)
    if is_freezing then
      if indent_table.is_trail_frozen then
        indent_table.trail = indent_table.frozen_trail
      else
        indent_table.frozen_trail = indent_table.trail
        indent_table.is_trail_frozen = true
      end
    else
      indent_table.frozen_trail = nil
      indent_table.is_trail_frozen = false
    end
    return true, indent_table
  end

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```
local check_continuation_indentation_and_trail =
  function (s, i, indent_table, is_optional, is_blank, trail_type,
            reset_rem, omit_remainder)
    if not has_indents(indent_table) then
      local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
                                           * Cp(), s, i)
      local result, remainder = check_trail_type(s, i,
        indent_table.trail, spacing, trail_type)
      if remainder == nil then
        if result then
          return new_index
        end
        return false
      end
      if result then
        return new_index, remainder
      end
      return false
    end

    local passes, new_index, current_trail = traverse_indent(s, i,
      indent_table, is_optional, is_blank)

    if passes then
      local spacing
      if current_trail == nil then
        local newer_spacing, newer_index = lpeg.match(
          C(parsers.spacechar^0) * Cp(), s, i)
        current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
        new_index = newer_index
        spacing = newer_spacing
      else
        spacing = current_trail.remainder
      end
      local result, remainder = check_trail_type(s, new_index,
        current_trail, spacing, trail_type)
      if remainder == nil or omit_remainder then
        if result then
          return new_index
        end
        return false
      end

      if is_blank and reset_rem then
```

```
        remainder = remove_remainder_if_blank(indent_table, remainder)
      end
      if result then
        return new_index, remainder
      end
      return false
    end
    return false
  end

```

The following patterns check whitespace indentation at the start of a block.

```
parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
                         * Cc(false), check_trail_joined)

parsers.check_trail_no_rem = Cmt( Cb("indent_info")
                                * C(parsers.spacechar^0) * Cc(false)
                                * Cc(true), check_trail_joined)

parsers.check_code_trail  = Cmt( Cb("indent_info")
                               * C(parsers.spacechar^0)
                               * Cc(true), check_trail_joined)

parsers.check_trail_length_range  = function(min, max)
  return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
            * Cc(max), check_trail_length)
end

parsers.check_trail_length = function(n)
  return parsers.check_trail_length_range(n, n)
end

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```
parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
                         * Cc(true), trail_freezing), "indent_info")

parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
                            trail_freezing), "indent_info")

```

The following patterns check indentation in continuation lines as defined by the container start.

```
parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
                                   check_continuation_indentation)

parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
```

```
                                        check_continuation_indentation)

parsers.check_minimal_blank_indent
  = Cmt( Cb("indent_info") * Cc(false)
       * Cc(true)
       , check_continuation_indentation)
```

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

```
parsers.check_minimal_indent_and_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("code") * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_full_code_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_minimal_blank_indent_and_any_trail_no_rem =
  Cmt( Cb("indent_info")
     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
     , check_continuation_indentation_and_trail)

parsers.check_optional_indent_and_any_trail =
  Cmt( Cb("indent_info")
     * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

parsers.check_optional_blank_indent_and_any_trail =
```

```
  Cmt( Cb("indent_info")
     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
     , check_continuation_indentation_and_trail)

```

The following patterns specify behaviour around newlines.

```

parsers.spnlc_noexc = parsers.optionalspace
                    * ( parsers.newline
                      * parsers.check_minimal_indent_and_any_trail)^-1

parsers.spnlc = parsers.optionalspace
              * (V("EndlineNoSub"))^-1

parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
                   + parsers.spacechar^1

parsers.only_blank = parsers.spacechar^0
                   * (parsers.newline + parsers.eof)

%   \end{macrocode}
% \begin{figure}
% \hspace*{-0.1\textwidth}
% \begin{minipage}{1.2\textwidth}
% \centering
% \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
% \node[state, initial by diamond, accepting] (noop) {initial};
% \node[state] (odd_backslash) [above right=of noop] {odd backslash};
% \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
% \node[state] (comment) [below=of noop] {comment};
% \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs
% \node[state] (blank_line) [below right=of comment] {blank line};
% \path[->]
% (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\
%        edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
%        edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
% (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$$\drsh
%           edge [bend left=10] node {match $\drsh$} (leading_spaces)
% (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$
%                  edge [bend right=90] node [right] {match \textbackslash} (odd_back
%                  edge [bend left=10] node {match \%} (comment)
%                  edge [bend right=10] node {$\epsilon$} (blank_line)
%                  edge [bend left=10] node [align=center, right=0.3cm] {match [$^\we
% (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
%              edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\\
% (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
%                 edge [bend right=10] node [align=center, above left=-
0.3cm, xshift=0.1cm] {match [$^\wedge$\textbackslash]\\for \%, capture \textbackslash
```

```
% (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
% \end{tikzpicture}
% \caption{A pushdown automaton that recognizes \TeX{} comments}
% \label{fig:commented_line}
% \end{minipage}
% \end{figure}
% \begin{markdown}
%
% The \luamdef{parsers.commented_line}`^1` parser recognizes the regular
% language of \TeX{} comments, see an equivalent finite automaton in Figure
% <#fig:commented_line>.
%
% \end{markdown}
%  \begin{macrocode}
parsers.commented_line_letter  = parsers.linechar
                               + parsers.newline
                               - parsers.backslash
                               - parsers.percent
parsers.commented_line = Cg(Cc(""), "backslashes")
                       * ((#(parsers.commented_line_letter
                          - parsers.newline)
                          * Cb("backslashes")
                          * Cs(parsers.commented_line_letter
                            - parsers.newline)^1  -- initial
                          * Cg(Cc(""), "backslashes"))
                         + #(parsers.backslash * parsers.backslash)
                         * Cg((parsers.backslash  -- even backslash
                              * parsers.backslash)^1, "backslashes")
                         + (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")))^0
                       * (#parsers.percent
                         * Cb("backslashes")
                         / function(backslashes)
                           return string.rep("\\", #backslashes / 2)
                         end
                         * ((parsers.percent  -- comment
                            * parsers.line
```

```
                             * #parsers.blankline) -- blank line
                            / "\n"
                            + parsers.percent  -- comment
                            * parsers.line
                            * parsers.optionalspace)  -- leading spaces
                          + #(parsers.newline)
                          * Cb("backslashes")
                          * C(parsers.newline))

parsers.chunk = parsers.line * (parsers.optionallyindentedline
                                      - parsers.blankline)^0

parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
parsers.attribute_key = (parsers.attribute_key_char
                         - parsers.dash - parsers.digit)
                       * parsers.attribute_key_char^0
parsers.attribute_value = ( (parsers.dquote / "")
                            * (parsers.anyescaped - parsers.dquote)^0
                            * (parsers.dquote / ""))
                          + ( (parsers.squote / "")
                            * (parsers.anyescaped - parsers.squote)^0
                            * (parsers.squote / ""))
                          + ( parsers.anyescaped
                            - parsers.dquote
                            - parsers.rbrace
                            - parsers.space)^0
parsers.attribute_identifier = parsers.attribute_key_char^1
parsers.attribute_classname = parsers.letter
                              * parsers.attribute_key_char^0
parsers.attribute_raw = parsers.attribute_raw_char^1

parsers.attribute = (parsers.dash * Cc(".unnumbered"))
                  + C( parsers.hash
                     * parsers.attribute_identifier)
                  + C( parsers.period
                     * parsers.attribute_classname)
                  + Cs( parsers.attribute_key
                      * parsers.optionalspace
                      * parsers.equal
                      * parsers.optionalspace
                      * parsers.attribute_value)
parsers.attributes = parsers.lbrace
                   * parsers.optionalspace
                   * parsers.attribute
                   * (parsers.spacechar^1
                     * parsers.attribute)^0
```

```
                    * parsers.optionalspace
                    * parsers.rbrace

parsers.raw_attribute = parsers.lbrace
                      * parsers.optionalspace
                      * parsers.equal
                      * C(parsers.attribute_raw)
                      * parsers.optionalspace
                      * parsers.rbrace

-- block followed by 0 or more optionally
-- indented blocks with first line indented.
parsers.indented_blocks = function(bl)
  return Cs( bl
         * ( parsers.blankline^1
           * parsers.indent
           * -parsers.blankline
           * bl)^0
         * (parsers.blankline^1 + parsers.eof) )
end
```

3.1.5.2 Parsers Used for HTML Entities

```
local function repeat_between(pattern, min, max)
  return -pattern^(max + 1) * pattern^min
end

parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
                  * C(repeat_between(parsers.hexdigit, 1, 6))
                  * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                  * C(repeat_between(parsers.digit, 1, 7))
                  * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
                  * parsers.semicolon

parsers.html_entities
  = parsers.hexentity / entities.hex_entity_with_x_char
  + parsers.decentity / entities.dec_entity
  + parsers.tagentity / entities.char_entity
```

3.1.5.3 Parsers Used for Markdown Lists

```
parsers.bullet = function(bullet_char, interrupting)
  local allowed_end
  if interrupting then
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
```

```
    allowed_end = C(parsers.spacechar^1)
                + #(parsers.newline + parsers.eof)
  end
  return parsers.check_trail
        * Ct(C(bullet_char) * Cc(""))
        * allowed_end
end

local function tickbox(interior)
  return parsers.optionalspace * parsers.lbracket
       * interior * parsers.rbracket * parsers.spacechar^1
end

parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)

```

3.1.5.4 Parsers Used for Markdown Code Spans

```
parsers.openticks   = Cg(parsers.backtick^1, "ticks")

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

parsers.closeticks  = Cmt(C(parsers.backtick^1)
                          * Cb("ticks"), captures_equal_length)

parsers.intickschar = (parsers.any - S("\n\r`"))
                    + V("NoSoftLineBreakEndline")
                    + (parsers.backtick^1 - parsers.closeticks)

local function process_inticks(s)
  s = s:gsub("\n", " ")
  s = s:gsub("^ (.*) $", "%1")
  return s
end

parsers.inticks = parsers.openticks
                * C(parsers.space^0)
                * parsers.closeticks
                + parsers.openticks
                * Cs(Cs(parsers.intickschar^0) / process_inticks)
                * parsers.closeticks

```

3.1.5.5 Parsers Used for HTML

```
7462 -- case-insensitive match (we assume s is lowercase)
7463 -- must be single byte encoding
7464 parsers.keyword_exact = function(s)
7465   local parser = P(0)
7466   for i=1,#s do
7467     local c = s:sub(i,i)
7468     local m = c .. upper(c)
7469     parser = parser * S(m)
7470   end
7471   return parser
7472 end
7473
7474 parsers.special_block_keyword =
7475     parsers.keyword_exact("pre") +
7476     parsers.keyword_exact("script") +
7477     parsers.keyword_exact("style") +
7478     parsers.keyword_exact("textarea")
7479
7480 parsers.block_keyword =
7481     parsers.keyword_exact("address") +
7482     parsers.keyword_exact("article") +
7483     parsers.keyword_exact("aside") +
7484     parsers.keyword_exact("base") +
7485     parsers.keyword_exact("basefont") +
7486     parsers.keyword_exact("blockquote") +
7487     parsers.keyword_exact("body") +
7488     parsers.keyword_exact("caption") +
7489     parsers.keyword_exact("center") +
7490     parsers.keyword_exact("col") +
7491     parsers.keyword_exact("colgroup") +
7492     parsers.keyword_exact("dd") +
7493     parsers.keyword_exact("details") +
7494     parsers.keyword_exact("dialog") +
7495     parsers.keyword_exact("dir") +
7496     parsers.keyword_exact("div") +
7497     parsers.keyword_exact("dl") +
7498     parsers.keyword_exact("dt") +
7499     parsers.keyword_exact("fieldset") +
7500     parsers.keyword_exact("figcaption") +
7501     parsers.keyword_exact("figure") +
7502     parsers.keyword_exact("footer") +
7503     parsers.keyword_exact("form") +
7504     parsers.keyword_exact("frame") +
7505     parsers.keyword_exact("frameset") +
7506     parsers.keyword_exact("h1") +
7507     parsers.keyword_exact("h2") +
```

```
    parsers.keyword_exact("h3") +
    parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") +
    parsers.keyword_exact("h6") +
    parsers.keyword_exact("head") +
    parsers.keyword_exact("header") +
    parsers.keyword_exact("hr") +
    parsers.keyword_exact("html") +
    parsers.keyword_exact("iframe") +
    parsers.keyword_exact("legend") +
    parsers.keyword_exact("li") +
    parsers.keyword_exact("link") +
    parsers.keyword_exact("main") +
    parsers.keyword_exact("menu") +
    parsers.keyword_exact("menuitem") +
    parsers.keyword_exact("nav") +
    parsers.keyword_exact("noframes") +
    parsers.keyword_exact("ol") +
    parsers.keyword_exact("optgroup") +
    parsers.keyword_exact("option") +
    parsers.keyword_exact("p") +
    parsers.keyword_exact("param") +
    parsers.keyword_exact("section") +
    parsers.keyword_exact("source") +
    parsers.keyword_exact("summary") +
    parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") +
    parsers.keyword_exact("td") +
    parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("th") +
    parsers.keyword_exact("thead") +
    parsers.keyword_exact("title") +
    parsers.keyword_exact("tr") +
    parsers.keyword_exact("track") +
    parsers.keyword_exact("ul")

-- end conditions
parsers.html_blankline_end_condition
  = parsers.linechar^0
  * ( parsers.newline
    * (parsers.check_minimal_blank_indent_and_any_trail
      * #parsers.blankline
      + parsers.check_minimal_indent_and_any_trail)
    * parsers.linechar^1)^0
  * (parsers.newline^-1 / "")

local function remove_trailing_blank_lines(s)
```

```
  return s:gsub("[\n\r]+%s*$", "")
end

parsers.html_until_end = function(end_marker)
  return Cs(Cs((parsers.newline
           * (parsers.check_minimal_blank_indent_and_any_trail
             * #parsers.blankline
             + parsers.check_minimal_indent_and_any_trail)
           + parsers.linechar - end_marker)^0
           * parsers.linechar^0 * parsers.newline^-1)
         / remove_trailing_blank_lines)
end

-- attributes
parsers.html_attribute_spacing  = parsers.optionalspace
                                * V("NoSoftLineBreakEndline")
                                * parsers.optionalspace
                                + parsers.spacechar^1

parsers.html_attribute_name = ( parsers.letter
                              + parsers.colon
                              + parsers.underscore)
                            * ( parsers.alphanumeric
                              + parsers.colon
                              + parsers.underscore
                            + parsers.period
                            + parsers.dash)^0

parsers.html_attribute_value  = parsers.squote
                              * (parsers.linechar - parsers.squote)^0
                              * parsers.squote
                              + parsers.dquote
                              * (parsers.linechar - parsers.dquote)^0
                              * parsers.dquote
                              + ( parsers.any
                                - parsers.spacechar
                                - parsers.newline
                                - parsers.dquote
                                - parsers.squote
                                - parsers.backtick
                                - parsers.equal
                                - parsers.less
                                - parsers.more)^1

parsers.html_inline_attribute_value = parsers.squote
                                    * (V("NoSoftLineBreakEndline")
                                      + parsers.any
```

```
                                      - parsers.blankline^2
                                      - parsers.squote)^0
                                    * parsers.squote
                                    + parsers.dquote
                                    * (V("NoSoftLineBreakEndline")
                                      + parsers.any
                                      - parsers.blankline^2
                                      - parsers.dquote)^0
                                    * parsers.dquote
                                    + (parsers.any
                                      - parsers.spacechar
                                      - parsers.newline
                                      - parsers.dquote
                                      - parsers.squote
                                      - parsers.backtick
                                      - parsers.equal
                                      - parsers.less
                                      - parsers.more)^1

parsers.html_attribute_value_specification
  = parsers.optionalspace
  * parsers.equal
  * parsers.optionalspace
  * parsers.html_attribute_value

parsers.html_spnl = parsers.optionalspace
                  * (V("NoSoftLineBreakEndline")
                  * parsers.optionalspace)^-1

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.html_comment_end = P("-->")

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

parsers.html_inline_comment = (parsers.html_comment_start / "")
                            * -P(">") * -P("->")
                            * Cs(( V("NoSoftLineBreakEndline")
                                 + 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))

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))

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("<?")
```

```
7696
7697 parsers.html_instruction_end = P("?>")
7698
7699 parsers.html_instruction
7700   = Cs( parsers.html_instruction_start
7701       * parsers.html_until_end(parsers.html_instruction_end))
7702
7703 parsers.html_inline_instruction = parsers.html_instruction_start
7704                                 * Cs( V("NoSoftLineBreakEndline")
7705                                     + parsers.any
7706                                     - parsers.nested_breaking_blank
7707                                     - parsers.html_instruction_end)^0
7708                                 * parsers.html_instruction_end
7709
7710 parsers.html_blankline  = parsers.newline
7711                         * parsers.optionalspace
7712                         * parsers.newline
7713
7714 parsers.html_tag_start = parsers.less
7715
7716 parsers.html_tag_closing_start  = parsers.less
7717                                 * parsers.slash
7718
7719 parsers.html_tag_end  = parsers.html_spnl
7720                       * parsers.more
7721
7722 parsers.html_empty_tag_end  = parsers.html_spnl
7723                             * parsers.slash
7724                             * parsers.more
7725
7726 -- opening tags
7727 parsers.html_any_open_inline_tag  = parsers.html_tag_start
7728                                   * parsers.keyword
7729                                   * parsers.html_attribute^0
7730                                   * parsers.html_tag_end
7731
7732 parsers.html_any_open_tag = parsers.html_tag_start
7733                           * parsers.keyword
7734                           * parsers.html_non_newline_attribute^0
7735                           * parsers.html_tag_end
7736
7737 parsers.html_open_tag = parsers.html_tag_start
7738                       * parsers.block_keyword
7739                       * parsers.html_attribute^0
7740                       * parsers.html_tag_end
7741
7742 parsers.html_open_special_tag = parsers.html_tag_start
```

```
                               * parsers.special_block_keyword
                               * parsers.html_attribute^0
                               * parsers.html_tag_end

-- incomplete tags
parsers.incomplete_tag_following  = parsers.spacechar
                                  + parsers.more
                                  + parsers.slash * parsers.more
                                  + #(parsers.newline + parsers.eof)

parsers.incomplete_special_tag_following = parsers.spacechar
                                         + parsers.more
                                         + #( parsers.newline
                                            + parsers.eof)

parsers.html_incomplete_open_tag  = parsers.html_tag_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

parsers.html_incomplete_open_special_tag
  = parsers.html_tag_start
  * parsers.special_block_keyword
  * parsers.incomplete_special_tag_following

parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
                                  * parsers.block_keyword
                                  * parsers.incomplete_tag_following

parsers.html_incomplete_close_special_tag
  = parsers.html_tag_closing_start
  * parsers.special_block_keyword
  * parsers.incomplete_tag_following

-- closing tags
parsers.html_close_tag  = parsers.html_tag_closing_start
                        * parsers.block_keyword
                        * parsers.html_tag_end

parsers.html_any_close_tag  = parsers.html_tag_closing_start
                            * parsers.keyword
                            * parsers.html_tag_end

parsers.html_close_special_tag = parsers.html_tag_closing_start
                               * parsers.special_block_keyword
                               * parsers.html_tag_end

-- empty tags
```

```
parsers.html_any_empty_inline_tag = parsers.html_tag_start
                                  * parsers.keyword
                                  * parsers.html_attribute^0
                                  * parsers.html_empty_tag_end

parsers.html_any_empty_tag  = parsers.html_tag_start
                            * parsers.keyword
                            * parsers.html_non_newline_attribute^0
                            * parsers.optionalspace
                            * parsers.slash
                            * parsers.more

parsers.html_empty_tag  = parsers.html_tag_start
                        * parsers.block_keyword
                        * parsers.html_attribute^0
                        * parsers.html_empty_tag_end

parsers.html_empty_special_tag  = parsers.html_tag_start
                                * parsers.special_block_keyword
                                * parsers.html_attribute^0
                                * parsers.html_empty_tag_end

parsers.html_incomplete_blocks
  = parsers.html_incomplete_open_tag
  + parsers.html_incomplete_open_special_tag
  + parsers.html_incomplete_close_tag

-- parse special html blocks
parsers.html_blankline_ending_special_block_opening
  = ( parsers.html_close_special_tag
    + parsers.html_empty_special_tag)
  * #( parsers.optionalspace
     * (parsers.newline + parsers.eof))

parsers.html_blankline_ending_special_block
  = parsers.html_blankline_ending_special_block_opening
  * parsers.html_blankline_end_condition

parsers.html_special_block_opening
  = parsers.html_incomplete_open_special_tag
  - parsers.html_empty_special_tag

parsers.html_closing_special_block
  = parsers.html_special_block_opening
  * parsers.html_until_end(parsers.html_close_special_tag)

parsers.html_special_block
```

```
  = parsers.html_blankline_ending_special_block
  + parsers.html_closing_special_block

-- parse html blocks
parsers.html_block_opening  = parsers.html_incomplete_open_tag
                            + parsers.html_incomplete_close_tag

parsers.html_block  = parsers.html_block_opening
                    * parsers.html_blankline_end_condition

-- parse any html blocks
parsers.html_any_block_opening
  = ( parsers.html_any_open_tag
    + parsers.html_any_close_tag
    + parsers.html_any_empty_tag)
  * #(parsers.optionalspace * (parsers.newline + parsers.eof))

parsers.html_any_block  = parsers.html_any_block_opening
                        * parsers.html_blankline_end_condition

parsers.html_inline_comment_full  = parsers.html_comment_start
                                  * -P(">") * -P("->")
                                  * Cs(( V("NoSoftLineBreakEndline")
                                       + parsers.any - P("--")
                                       - parsers.nested_breaking_blank
                                       - parsers.html_comment_end)^0)
                                  * parsers.html_comment_end

parsers.html_inline_tags  = parsers.html_inline_comment_full
                          + parsers.html_any_empty_inline_tag
                          + parsers.html_inline_instruction
                          + parsers.html_inline_cdatasection
                          + parsers.html_inline_declaration
                          + parsers.html_any_open_inline_tag
                          + parsers.html_any_close_tag

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```
parsers.urlchar = parsers.anyescaped
                - parsers.newline
                - parsers.more

parsers.auto_link_scheme_part = parsers.alphanumeric
                              + parsers.plus
                              + parsers.period
                              + parsers.dash
```

```
7881
7882 parsers.auto_link_scheme  = parsers.letter
7883                           * parsers.auto_link_scheme_part
7884                           * parsers.auto_link_scheme_part^-30
7885
7886 parsers.absolute_uri  = parsers.auto_link_scheme * parsers.colon
7887                       * ( parsers.any - parsers.spacing
7888                         - parsers.less - parsers.more)^0
7889
7890 parsers.printable_characters = S(".!#$%&'*+/=?^_`{|}~-")
7891
7892 parsers.email_address_local_part_char = parsers.alphanumeric
7893                                       + parsers.printable_characters
7894
7895 parsers.email_address_local_part
7896   = parsers.email_address_local_part_char^1
7897
7898 parsers.email_address_dns_label = parsers.alphanumeric
7899                                 * ( parsers.alphanumeric
7900                                   + parsers.dash)^-62
7901                                 * B(parsers.alphanumeric)
7902
7903 parsers.email_address_domain  = parsers.email_address_dns_label
7904                               * ( parsers.period
7905                                 * parsers.email_address_dns_label)^0
7906
7907 parsers.email_address = parsers.email_address_local_part
7908                       * parsers.at
7909                       * parsers.email_address_domain
7910
7911 parsers.auto_link_url = parsers.less
7912                       * C(parsers.absolute_uri)
7913                       * parsers.more
7914
7915 parsers.auto_link_email = parsers.less
7916                         * C(parsers.email_address)
7917                         * parsers.more
7918
7919 parsers.auto_link_relative_reference = parsers.less
7920                                      * C(parsers.urlchar^1)
7921                                      * parsers.more
7922
7923 parsers.autolink  = parsers.auto_link_url
7924                   + parsers.auto_link_email
7925
7926 -- content in balanced brackets, parentheses, or quotes:
7927 parsers.bracketed   = P{ parsers.lbracket
```

```
                        * (( parsers.backslash / "" * parsers.rbracket
                           + parsers.any - (parsers.lbracket
                                          + parsers.rbracket
                                          + parsers.blankline^2)
                           ) + V(1))^0
                        * parsers.rbracket }

parsers.inparens    = P{ parsers.lparent
                        * ((parsers.anyescaped - (parsers.lparent
                                                + parsers.rparent
                                                + parsers.blankline^2)
                           ) + V(1))^0
                        * parsers.rparent }

parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
                        * ((parsers.anyescaped - (parsers.squote
                                                + parsers.blankline^2)
                           ) + V(1))^0
                        * parsers.squote }

parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
                        * ((parsers.anyescaped - (parsers.dquote
                                                + parsers.blankline^2)
                           ) + V(1))^0
                        * parsers.dquote }

parsers.link_text  = parsers.lbracket
                     * Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + parsers.autolink
                         + V("InlineHtml")
                         + ( parsers.backslash * parsers.backslash)
                         + ( parsers.backslash
                           * ( parsers.lbracket
                             + parsers.rbracket)
                           + V("NoSoftLineBreakSpace")
                           + V("NoSoftLineBreakEndline")
                           + (parsers.any
                             - ( parsers.newline
                               + parsers.lbracket
                               + parsers.rbracket
                               + parsers.blankline^2))))^0)
                     * parsers.rbracket

parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
                        * #( ( parsers.any
```

```
                              - parsers.rbracket)^-999
                           * parsers.rbracket)
                       * Cs((parsers.alphanumeric^1
                           + parsers.inticks
                           + parsers.autolink
                           + V("InlineHtml")
                           + ( parsers.backslash * parsers.backslash)
                           + ( parsers.backslash
                             * ( parsers.lbracket
                               + parsers.rbracket)
                             + V("NoSoftLineBreakSpace")
                             + V("NoSoftLineBreakEndline")
                             + (parsers.any
                               - ( parsers.newline
                                 + parsers.lbracket
                                 + parsers.rbracket
                                 + parsers.blankline^2))))^1)

parsers.link_label  = parsers.lbracket
                    * parsers.link_label_body
                    * parsers.rbracket

parsers.inparens_url  = P{ parsers.lparent
                         * ((parsers.anyescaped - (parsers.lparent
                                                 + parsers.rparent
                                                 + parsers.spacing)
                            ) + V(1))^0
                         * parsers.rparent }

-- 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("NoSoftLineBreakSpace")
                         + V("NoSoftLineBreakEndline")
                         + ( parsers.anyescaped
```

```
                            - parsers.newline
                            - parsers.squote
                            - parsers.blankline^2))^0)
                      * parsers.squote

parsers.title_d     = parsers.dquote
                      * Cs((parsers.html_entities
                           + V("NoSoftLineBreakSpace")
                           + V("NoSoftLineBreakEndline")
                           + ( parsers.anyescaped
                             - parsers.newline
                             - parsers.dquote
                             - parsers.blankline^2))^0)
                      * parsers.dquote

parsers.title_p     = parsers.lparent
                      * Cs((parsers.html_entities
                           + V("NoSoftLineBreakSpace")
                           + 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("")
```

3.1.5.7 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)
```

3.1.5.8 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

```

3.1.5.9 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)
```

3.1.5.10 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*⟩.

```
8104 M.reader = {}
8105 function M.reader.new(writer, options)
8106   local self = {}
```

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

```
8107   self.writer = writer
8108   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.

```
8109   self.parsers = {}
8110   (function(parsers)
8111     setmetatable(self.parsers, {
8112       __index = function (_, key)
8113         return parsers[key]
8114       end
8115     })
8116   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.

```
8117   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.

```
8118   function self.normalize_tag(tag)
8119     tag = util.rope_to_string(tag)
8120     tag = tag:gsub("[ \n\r\t]+", " ")
8121     tag = tag:gsub("^ ", ""):gsub(" $", "")
8122     tag = uni_algos.case.casefold(tag, true, false)
8123     return tag
```

```
8124  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.

```
8125  local function iterlines(s, f)
8126    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8127    return util.rope_to_string(rope)
8128  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.

```
8129  if options.preserveTabs then
8130    self.expandtabs = function(s) return s end
8131  else
8132    self.expandtabs = function(s)
8133                        if s:find("\t") then
8134                          return iterlines(s, util.expand_tabs_in_line)
8135                        else
8136                          return s
8137                        end
8138                      end
8139  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.

```
8140  self.parser_functions = {}
8141  self.create_parser = function(name, grammar, toplevel)
8142    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.

```
8143      if toplevel and options.stripIndent then
8144          local min_prefix_length, min_prefix = nil, ''
8145          str = iterlines(str, function(line)
8146              if lpeg.match(parsers.nonemptyline, line) == nil then
8147                  return line
8148              end
8149              line = util.expand_tabs_in_line(line)
8150              local prefix = lpeg.match(C(parsers.optionalspace), line)
```

```
8151              local prefix_length = #prefix
8152              local is_shorter = min_prefix_length == nil
8153              if not is_shorter then
8154                is_shorter = prefix_length < min_prefix_length
8155              end
8156              if is_shorter then
8157                min_prefix_length, min_prefix = prefix_length, prefix
8158              end
8159              return line
8160            end)
8161            str = str:gsub('^' .. min_prefix, '')
8162        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.

```
8163        if toplevel and (options.texComments or options.hybrid) then
8164          str = lpeg.match(Ct(parsers.commented_line^1), str)
8165          str = util.rope_to_string(str)
8166        end
8167        local res = lpeg.match(grammar(), str)
8168        if res == nil then
8169          return writer.error(format("%s failed on:\n%s",
8170                                     name, str:sub(1,20)))
8171        else
8172          return res
8173        end
8174      end
8175    end
8176
8177    self.create_parser("parse_blocks",
8178                       function()
8179                         return parsers.blocks
8180                       end, true)
8181
8182    self.create_parser("parse_blocks_nested",
8183                       function()
8184                         return parsers.blocks_nested
8185                       end, false)
8186
8187    self.create_parser("parse_inlines",
8188                       function()
8189                         return parsers.inlines
8190                       end, false)
8191
8192    self.create_parser("parse_inlines_no_inline_note",
8193                       function()
```

```
                      return parsers.inlines_no_inline_note
                    end, false)

  self.create_parser("parse_inlines_no_html",
                    function()
                      return parsers.inlines_no_html
                    end, false)

  self.create_parser("parse_inlines_nbsp",
                    function()
                      return parsers.inlines_nbsp
                    end, false)
  self.create_parser("parse_inlines_no_link_or_emphasis",
                   function()
                     return parsers.inlines_no_link_or_emphasis
                   end, false)
```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```
  parsers.minimally_indented_blankline
    = parsers.check_minimal_indent * (parsers.blankline / "")

  parsers.minimally_indented_block
    = parsers.check_minimal_indent * V("Block")

  parsers.minimally_indented_block_or_paragraph
    = parsers.check_minimal_indent * V("BlockOrParagraph")

  parsers.minimally_indented_paragraph
    = parsers.check_minimal_indent * V("Paragraph")

  parsers.minimally_indented_plain
    = parsers.check_minimal_indent * V("Plain")

  parsers.minimally_indented_par_or_plain
    = parsers.minimally_indented_paragraph
    + parsers.minimally_indented_plain

  parsers.minimally_indented_par_or_plain_no_blank
    = parsers.minimally_indented_par_or_plain
    - parsers.minimally_indented_blankline

  parsers.minimally_indented_ref
    = parsers.check_minimal_indent * V("Reference")

  parsers.minimally_indented_blank
```

```
  = parsers.check_minimal_indent * V("Blank")

parsers.conditionally_indented_blankline
  = parsers.check_minimal_blank_indent * (parsers.blankline / "")

parsers.minimally_indented_ref_or_block
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block
  - parsers.minimally_indented_blankline

parsers.minimally_indented_ref_or_block_or_par
  = parsers.minimally_indented_ref
  + parsers.minimally_indented_block_or_paragraph
  - parsers.minimally_indented_blankline
```

The following pattern parses the properly indented content that follows the initial container start.

```
function parsers.separator_loop(separated_block, paragraph,
                                block_separator, paragraph_separator)
  return  separated_block
        + block_separator
          * paragraph
          * separated_block
        + paragraph_separator
        * paragraph
end

function parsers.create_loop_body_pair(separated_block, paragraph,
                                       block_separator,
                                       paragraph_separator)
  return {
    block = parsers.separator_loop(separated_block, paragraph,
                                   block_separator, block_separator),
    par = parsers.separator_loop(separated_block, paragraph,
                                 block_separator, paragraph_separator)
  }
end

parsers.block_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + ( blank^2 / writer.paragraphsep
          + blank^0 / writer.interblocksep
          )
end
```

```
parsers.par_sep_group = function(blank)
  return  blank^0 * parsers.eof
        + blank^0 / writer.paragraphsep
end

parsers.sep_group_no_output = function(blank)
  return  blank^0 * parsers.eof
        + blank^0
end

parsers.content_blank = parsers.minimally_indented_blankline

parsers.ref_or_block_separated
  = parsers.sep_group_no_output(parsers.content_blank)
  * ( parsers.minimally_indented_ref
    - parsers.content_blank)
  + parsers.block_sep_group(parsers.content_blank)
  * ( parsers.minimally_indented_block
    - parsers.content_blank)

parsers.loop_body_pair  =
  parsers.create_loop_body_pair(
    parsers.ref_or_block_separated,
    parsers.minimally_indented_par_or_plain_no_blank,
    parsers.block_sep_group(parsers.content_blank),
    parsers.par_sep_group(parsers.content_blank))

parsers.content_loop  = ( V("Block")
                        * parsers.loop_body_pair.block^0
                        + (V("Paragraph") + V("Plain"))
                        * parsers.ref_or_block_separated
                        * parsers.loop_body_pair.block^0
                        + (V("Paragraph") + V("Plain"))
                        * parsers.loop_body_pair.par^0)
                      * parsers.content_blank^0

parsers.indented_content = function()
  return  Ct( (V("Reference") + (parsers.blankline / ""))
            * parsers.content_blank^0
            * parsers.check_minimal_indent
            * parsers.content_loop
            + (V("Reference") + (parsers.blankline / ""))
            * parsers.content_blank^0
            + parsers.content_loop)
end

parsers.add_indent = function(pattern, name, breakable)
```

```
8328    return  Cg(Cmt( Cb("indent_info")
8329                  * Ct(pattern)
8330                  * ( #parsers.linechar  -- check if starter is blank
8331                    * Cc(false) + Cc(true))
8332                  * Cc(name)
8333                  * Cc(breakable),
8334              process_starter_indent), "indent_info")
8335  end
8336
```

3.1.6.4 Parsers Used for Markdown Lists (local)

```
8337  if options.hashEnumerators then
8338    parsers.dig = parsers.digit + parsers.hash
8339  else
8340    parsers.dig = parsers.digit
8341  end
8342
8343  parsers.enumerator = function(delimiter_type, interrupting)
8344    local delimiter_range
8345    local allowed_end
8346    if interrupting then
8347      delimiter_range = P("1")
8348      allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8349    else
8350      delimiter_range = parsers.dig * parsers.dig^-8
8351      allowed_end = C(parsers.spacechar^1)
8352                  + #(parsers.newline + parsers.eof)
8353    end
8354
8355    return parsers.check_trail
8356             * Ct(C(delimiter_range) * C(delimiter_type))
8357             * allowed_end
8358  end
8359
8360  parsers.starter = parsers.bullet(parsers.dash)
8361                  + parsers.bullet(parsers.asterisk)
8362                  + parsers.bullet(parsers.plus)
8363                  + parsers.enumerator(parsers.period)
8364                  + parsers.enumerator(parsers.rparent)
8365
```

3.1.6.5 Parsers Used for Blockquotes (local)

```
8366  parsers.blockquote_start
8367    = parsers.check_trail
8368    * C(parsers.more)
8369    * C(parsers.spacechar^0)
```

```
8370
8371  parsers.blockquote_body
8372    = parsers.add_indent(parsers.blockquote_start, "bq", true)
8373    * parsers.indented_content()
8374    * remove_indent("bq")
8375
8376  if not options.breakableBlockquotes then
8377    parsers.blockquote_body
8378      = parsers.add_indent(parsers.blockquote_start, "bq", false)
8379      * parsers.indented_content()
8380      * remove_indent("bq")
8381  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.

```
8382  local function parse_content_part(content_part)
8383    local rope = util.rope_to_string(content_part)
8384    local parsed
8385      = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8386    parsed.indent_info = nil
8387    return parsed
8388  end
8389
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8390  local collect_emphasis_content =
8391    function(t, opening_index, closing_index)
8392      local content = {}
8393
8394      local content_part = {}
8395      for i = opening_index, closing_index do
8396        local value = t[i]
8397
8398        if value.rendered ~= nil then
8399          content[#content + 1] = parse_content_part(content_part)
8400          content_part = {}
8401          content[#content + 1] = value.rendered
8402          value.rendered = nil
8403        else
8404          if value.type == "delimiter"
8405              and value.element == "emphasis" then
8406            if value.is_active then
8407              content_part[#content_part + 1]
8408                = string.rep(value.character, value.current_count)
8409            end
```

```
          else
            content_part[#content_part + 1] = value.content
          end
          value.content = ''
          value.is_active = false
        end
      end

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

      return content
    end

```

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

```
  local function fill_emph(t, opening_index, closing_index)
    local content
      = collect_emphasis_content(t, opening_index + 1,
                                 closing_index - 1)
    t[opening_index + 1].is_active = true
    t[opening_index + 1].rendered = writer.emphasis(content)
  end

```

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

```
  local function fill_strong(t, opening_index, closing_index)
    local content
      = collect_emphasis_content(t, opening_index + 1,
                                 closing_index - 1)
    t[opening_index + 1].is_active = true
    t[opening_index + 1].rendered = writer.strong(content)
  end

```

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

```
  local function breaks_three_rule(opening_delimiter, closing_delimiter)
    return ( opening_delimiter.is_closing
          or closing_delimiter.is_opening)
       and (( opening_delimiter.original_count
            + closing_delimiter.original_count) % 3 == 0)
       and ( opening_delimiter.original_count % 3 ~= 0
          or closing_delimiter.original_count % 3 ~= 0)
  end
```

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
  end

  local cont = lpeg.R("\128\191") -- continuation byte

```

Match a UTF-8 character of byte length `n`.

```
  local function utf8_by_byte_count(n)
    if (n == 1) then
      return lpeg.R("\0\127")
    end
    if (n == 2) then
      return lpeg.R("\194\223") * cont
    end
    if (n == 3) then
      return lpeg.R("\224\239") * cont * cont
    end
    if (n == 4) then
      return lpeg.R("\240\244") * cont * cont * cont
    end
  end
```

Check if a there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```
  local function check_unicode_type(s, i, start_pos, end_pos, chartype)
    local c
    local char_length
    for pos = start_pos, end_pos, 1 do
      if (start_pos < 0) then
        char_length = -pos
      else
        char_length = pos + 1
      end

      if (chartype == "punctuation") then
        if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
          return i
        end
      else
        c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
        if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
          return i
        end
      end
```

```
8580    end
8581  end
8582
8583  local function check_preceding_unicode_punctuation(s, i)
8584    return check_unicode_type(s, i, -4, -1, "punctuation")
8585  end
8586
8587  local function check_preceding_unicode_whitespace(s, i)
8588    return check_unicode_type(s, i, -4, -1, "%s")
8589  end
8590
8591  local function check_following_unicode_punctuation(s, i)
8592    return check_unicode_type(s, i, 0, 3, "punctuation")
8593  end
8594
8595  local function check_following_unicode_whitespace(s, i)
8596    return check_unicode_type(s, i, 0, 3, "%s")
8597  end
8598
8599  parsers.unicode_preceding_punctuation
8600    = B(parsers.escapable)
8601    + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8602
8603  parsers.unicode_preceding_whitespace
8604    = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8605
8606  parsers.unicode_following_punctuation
8607    = #parsers.escapable
8608    + Cmt(parsers.succeed, check_following_unicode_punctuation)
8609
8610  parsers.unicode_following_whitespace
8611    = Cmt(parsers.succeed, check_following_unicode_whitespace)
8612
8613  parsers.delimiter_run = function(character)
8614    return  (B(parsers.backslash * character) + -B(character))
8615          * character^1
8616          * -#character
8617  end
8618
8619  parsers.left_flanking_delimiter_run = function(character)
8620    return  (B( parsers.any)
8621              * ( parsers.unicode_preceding_punctuation
8622                + parsers.unicode_preceding_whitespace)
8623             + -B(parsers.any))
8624            * parsers.delimiter_run(character)
8625            * parsers.unicode_following_punctuation
8626            + parsers.delimiter_run(character)
```

```
8627            * -#( parsers.unicode_following_punctuation
8628                + parsers.unicode_following_whitespace
8629                + parsers.eof)
8630  end
8631
8632  parsers.right_flanking_delimiter_run = function(character)
8633    return  parsers.unicode_preceding_punctuation
8634          * parsers.delimiter_run(character)
8635          * ( parsers.unicode_following_punctuation
8636            + parsers.unicode_following_whitespace
8637            + parsers.eof)
8638          + (B(parsers.any)
8639            * -( parsers.unicode_preceding_punctuation
8640               + parsers.unicode_preceding_whitespace))
8641          * parsers.delimiter_run(character)
8642  end
8643
8644  if options.underscores then
8645    parsers.emph_start
8646      = parsers.left_flanking_delimiter_run(parsers.asterisk)
8647      + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
8648        + ( parsers.unicode_preceding_punctuation
8649          * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8650      * parsers.left_flanking_delimiter_run(parsers.underscore)
8651
8652    parsers.emph_end
8653      = parsers.right_flanking_delimiter_run(parsers.asterisk)
8654      + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
8655        + #( parsers.left_flanking_delimiter_run(parsers.underscore)
8656           * parsers.unicode_following_punctuation))
8657      * parsers.right_flanking_delimiter_run(parsers.underscore)
8658  else
8659    parsers.emph_start
8660      = parsers.left_flanking_delimiter_run(parsers.asterisk)
8661
8662    parsers.emph_end
8663      = parsers.right_flanking_delimiter_run(parsers.asterisk)
8664  end
8665
8666  parsers.emph_capturing_open_and_close
8667    = #parsers.emph_start * #parsers.emph_end
8668    * Ct( Cg(Cc("delimiter"), "type")
8669        * Cg(Cc("emphasis"), "element")
8670        * Cg(C(parsers.emph_start), "content")
8671        * Cg(Cc(true), "is_opening")
8672        * Cg(Cc(true), "is_closing"))
8673
```

```
parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
                                * Cg(Cc("emphasis"), "element")
                                * Cg(C(parsers.emph_start), "content")
                                * Cg(Cc(true), "is_opening")
                                * Cg(Cc(false), "is_closing"))

parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
                                 * Cg(Cc("emphasis"), "element")
                                 * Cg(C(parsers.emph_end), "content")
                                 * Cg(Cc(false), "is_opening")
                                 * Cg(Cc(true), "is_closing"))

parsers.emph_open_or_close  = parsers.emph_capturing_open_and_close
                            + parsers.emph_capturing_open
                            + parsers.emph_capturing_close

parsers.emph_open = parsers.emph_capturing_open_and_close
                  + parsers.emph_capturing_open

parsers.emph_close  = parsers.emph_capturing_open_and_close
                    + parsers.emph_capturing_close

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
-- List of references defined in the document
local references

-- List of note references defined in the document
parsers.rawnotes = {}

```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```
function self.register_link(_, tag, url, title,
                            attributes)
  local normalized_tag = self.normalize_tag(tag)
    if references[normalized_tag] == nil then
      references[normalized_tag] = {
        url = url,
        title = title,
        attributes = attributes
      }
    end
  return ""
end

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
function self.lookup_reference(tag)
  return references[self.normalize_tag(tag)]
end

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```
function self.lookup_note_reference(tag)
  return parsers.rawnotes[self.normalize_tag(tag)]
end

parsers.title_s_direct_ref  = parsers.squote
                            * Cs((parsers.html_entities
                                + ( parsers.anyescaped
                                  - parsers.squote
                                  - parsers.blankline^2))^0)
                            * parsers.squote

parsers.title_d_direct_ref  = parsers.dquote
                            * Cs((parsers.html_entities
                                + ( parsers.anyescaped
                                  - parsers.dquote
                                  - parsers.blankline^2))^0)
                            * parsers.dquote

parsers.title_p_direct_ref  = parsers.lparent
                            * Cs((parsers.html_entities
                                + ( parsers.anyescaped
                                  - parsers.lparent
                                  - parsers.rparent
                                  - parsers.blankline^2))^0)
                            * parsers.rparent

parsers.title_direct_ref  = parsers.title_s_direct_ref
                          + parsers.title_d_direct_ref
                          + parsers.title_p_direct_ref

parsers.inline_direct_ref_inside  = parsers.lparent * parsers.spnl
                                  * Cg(parsers.url + Cc(""), "url")
                                  * parsers.spnl
                                  * Cg( parsers.title_direct_ref
                                      + Cc(""), "title")
                                  * parsers.spnl * parsers.rparent

parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
                          * Cg(parsers.url + Cc(""), "url")
```

```
                            * parsers.spnlc
                            * Cg(parsers.title + Cc(""), "title")
                            * parsers.spnlc * parsers.rparent

parsers.empty_link  = parsers.lbracket
                    * parsers.rbracket

parsers.inline_link = parsers.link_text
                    * parsers.inline_direct_ref

parsers.full_link = parsers.link_text
                  * parsers.link_label

parsers.shortcut_link = parsers.link_label
                      * -(parsers.empty_link + parsers.link_label)

parsers.collapsed_link  = parsers.link_label
                        * parsers.empty_link

parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
                      * Cg(Cc("inline"), "link_type")
                      + #(parsers.exclamation * parsers.full_link)
                      * Cg(Cc("full"), "link_type")
                      + #( parsers.exclamation
                         * parsers.collapsed_link)
                      * Cg(Cc("collapsed"), "link_type")
                      + #(parsers.exclamation * parsers.shortcut_link)
                      * Cg(Cc("shortcut"), "link_type")
                      + #(parsers.exclamation * parsers.empty_link)
                      * Cg(Cc("empty"), "link_type")

parsers.link_opening  = #parsers.inline_link
                      * Cg(Cc("inline"), "link_type")
                      + #parsers.full_link
                      * Cg(Cc("full"), "link_type")
                      + #parsers.collapsed_link
                      * Cg(Cc("collapsed"), "link_type")
                      + #parsers.shortcut_link
                      * Cg(Cc("shortcut"), "link_type")
                      + #parsers.empty_link
                      * Cg(Cc("empty_link"), "link_type")
                      + #parsers.link_text
                      * Cg(Cc("link_text"), "link_type")

parsers.note_opening  = #(parsers.circumflex * parsers.link_text)
                      * Cg(Cc("note_inline"), "link_type")

```

```
parsers.raw_note_opening  = #( parsers.lbracket
                               * parsers.circumflex
                               * parsers.link_label_body
                               * parsers.rbracket)
                             * Cg(Cc("raw_note"), "link_type")

local inline_note_element = Cg(Cc("note"), "element")
                            * parsers.note_opening
                            * Cg( parsers.circumflex
                                * parsers.lbracket, "content")

local image_element = Cg(Cc("image"), "element")
                      * parsers.image_opening
                      * Cg( parsers.exclamation
                          * parsers.lbracket, "content")

local note_element  = Cg(Cc("note"), "element")
                      * parsers.raw_note_opening
                      * Cg( parsers.lbracket
                          * parsers.circumflex, "content")

local link_element  = Cg(Cc("link"), "element")
                      * parsers.link_opening
                      * Cg(parsers.lbracket, "content")

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

opening_elements = opening_elements + link_element

parsers.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
                                  * Cg(Cc(true), "is_opening")
                                  * Cg(Cc(false), "is_closing")
                                  * opening_elements)

parsers.link_image_closing  = Ct( Cg(Cc("delimiter"), "type")
                                  * Cg(Cc("link"), "element")
                                  * Cg(Cc(false), "is_opening")
```

```
                                    * Cg(Cc(true), "is_closing")
                                    * ( Cg(Cc(true), "is_direct")
                                      * Cg( parsers.rbracket
                                          * #parsers.inline_direct_ref,
                                           "content")
                                      + Cg(Cc(false), "is_direct")
                                      * Cg(parsers.rbracket, "content")))

parsers.link_image_open_or_close  = parsers.link_image_opening
                                    + parsers.link_image_closing

if options.html then
  parsers.link_emph_precedence  = parsers.inticks
                                  + parsers.autolink
                                  + parsers.html_inline_tags
else
  parsers.link_emph_precedence  = parsers.inticks
                                  + parsers.autolink
end

parsers.link_and_emph_endline = parsers.newline
                                * ((parsers.check_minimal_indent
                                  * -V("EndlineExceptions")
                                  + parsers.check_optional_indent
                                  * -V("EndlineExceptions")
                                  * -parsers.starter) / "")
                                * parsers.spacechar^0 / "\n"

parsers.link_and_emph_content
  = Ct( Cg(Cc("content"), "type")
      * Cg(Cs(( parsers.link_emph_precedence
              + parsers.backslash * parsers.any
              + parsers.link_and_emph_endline
              + (parsers.linechar
                - parsers.blankline^2
                - parsers.link_image_open_or_close
                - parsers.emph_open_or_close))^0), "content"))

parsers.link_and_emph_table
  = (parsers.link_image_opening + parsers.emph_open)
  * parsers.link_and_emph_content
  * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
    * parsers.link_and_emph_content)^1
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local function collect_link_content(t, opening_index, closing_index)
  local content = {}
  for i = opening_index, closing_index do
    content[#content + 1] = t[i].content
  end
  return util.rope_to_string(content)
end

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
local function find_link_opener(t, bottom_index, latest_index)
  for i = latest_index, bottom_index, -1 do
    local value = t[i]
    if value.type == "delimiter" and
       value.is_opening and
       ( value.element == "link"
      or value.element == "image"
      or value.element == "note")
       and not value.removed then
      if value.is_active then
        return i
      end
      value.removed = true
      return nil
    end
  end
end

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
local function find_next_link_closing_index(t, latest_index)
  for i = latest_index, #t do
    local value = t[i]
    if value.is_closing and
       value.element == "link" and
       not value.removed then
      return i
    end
  end
end

```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```
local function disable_previous_link_openers(t, opening_index)
  if t[opening_index].element == "image" then
```

```
      return
    end

    for i = opening_index, 1, -1 do
      local value = t[i]
      if value.is_active and
         value.type == "delimiter" and
         value.is_opening and
         value.element == "link" then
        value.is_active = false
      end
    end
  end

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
  local function disable_range(t, opening_index, closing_index)
    for i = opening_index, closing_index do
      local value = t[i]
      if value.is_active then
        value.is_active = false
        if value.type == "delimiter" then
          value.removed = true
        end
      end
    end
  end

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
  local delete_parsed_content_in_range =
    function(t, opening_index, closing_index)
      for i = opening_index, closing_index do
        t[i].rendered = nil
      end
    end

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
  local function empty_content_in_range(t, opening_index, closing_index)
    for i = opening_index, closing_index do
      t[i].content = ''
    end
  end

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
local function join_attributes(reference_attributes, own_attributes)
  local merged_attributes = {}
  for _, attribute in ipairs(reference_attributes or {}) do
    table.insert(merged_attributes, attribute)
  end
  for _, attribute in ipairs(own_attributes or {}) do
    table.insert(merged_attributes, attribute)
  end
  if next(merged_attributes) == nil then
    merged_attributes = nil
  end
  return merged_attributes
end

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
local render_link_or_image =
  function(t, opening_index, closing_index, content_end_index,
           reference)
    process_emphasis(t, opening_index, content_end_index)
    local mapped = collect_emphasis_content(t, opening_index + 1,
                                            content_end_index - 1)

    local rendered = {}
    if (t[opening_index].element == "link") then
      rendered = writer.link(mapped, reference.url,
                             reference.title, reference.attributes)
    end

    if (t[opening_index].element == "image") then
      rendered = writer.image(mapped, reference.url, reference.title,
                              reference.attributes)
    end

    if (t[opening_index].element == "note") then
      if (t[opening_index].link_type == "note_inline") then
        rendered = writer.note(mapped)
      end
      if (t[opening_index].link_type == "raw_note") then
        rendered = writer.note(reference)
      end
    end

    t[opening_index].rendered = rendered
```

```
      delete_parsed_content_in_range(t, opening_index + 1,
                                     closing_index)
      empty_content_in_range(t, opening_index, closing_index)
      disable_previous_link_openers(t, opening_index)
      disable_range(t, opening_index, closing_index)
    end

```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```
  local resolve_inline_following_content =
    function(t, closing_index, match_reference, match_link_attributes)
      local content = ""
      for i = closing_index + 1, #t do
        content = content .. t[i].content
      end

      local matching_content = parsers.succeed

      if match_reference then
        matching_content = matching_content
                         * parsers.inline_direct_ref_inside
      end

      if match_link_attributes then
        matching_content = matching_content
                         * Cg(Ct(parsers.attributes^-1), "attributes")
      end

      local matched = lpeg.match(Ct( matching_content
                                   * Cg(Cp(), "end_position")), content)

      local matched_count = matched.end_position - 1
      for i = closing_index + 1, #t do
        local value = t[i]

        local chars_left = matched_count
        matched_count = matched_count - #value.content

        if matched_count <= 0 then
          value.content = value.content:sub(chars_left + 1)
          break
        end

        value.content = ''
        value.is_active = false
```

```
      end

      local attributes = matched.attributes
      if attributes == nil or next(attributes) == nil then
        attributes = nil
      end

      return {
        url = matched.url or "",
        title = matched.title or "",
        attributes = attributes
      }
    end

```

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.

```
  local function resolve_inline_link(t, opening_index, closing_index)
    local inline_content
      = resolve_inline_following_content(t, closing_index, true,
                                         t.match_link_attributes)
    render_link_or_image(t, opening_index, closing_index,
                         closing_index, inline_content)
  end

```

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

```
  local resolve_note_inline_link =
    function(t, opening_index, closing_index)
      local inline_content
        = resolve_inline_following_content(t, closing_index,
                                           false, false)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, inline_content)
    end

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
  local function resolve_shortcut_link(t, opening_index, closing_index)
    local content
      = collect_link_content(t, opening_index + 1, closing_index - 1)
    local r = self.lookup_reference(content)

    if r then
```

```
      local inline_content
        = resolve_inline_following_content(t, closing_index, false,
                                           t.match_link_attributes)
      r.attributes
        = join_attributes(r.attributes, inline_content.attributes)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, r)
    end
  end

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```
  local function resolve_raw_note_link(t, opening_index, closing_index)
    local content
      = collect_link_content(t, opening_index + 1, closing_index - 1)
    local r = self.lookup_note_reference(content)

    if r then
      local parsed_ref = self.parser_functions.parse_blocks_nested(r)
      render_link_or_image(t, opening_index, closing_index,
                           closing_index, parsed_ref)
    end
  end

```

Resolve a full link `[a][b]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```
  local function resolve_full_link(t, opening_index, closing_index)
    local next_link_closing_index
      = find_next_link_closing_index(t, closing_index + 4)
    local next_link_content
      = collect_link_content(t, closing_index + 3,
                             next_link_closing_index - 1)
    local r = self.lookup_reference(next_link_content)

    if r then
      local inline_content
        = resolve_inline_following_content(t, next_link_closing_index,
                                           false,
                                           t.match_link_attributes)
      r.attributes
        = join_attributes(r.attributes, inline_content.attributes)
      render_link_or_image(t, opening_index, next_link_closing_index,
                           closing_index, r)
    end
  end

```

Resolve a collapsed link `[a][]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```
local function resolve_collapsed_link(t, opening_index, closing_index)
  local next_link_closing_index
    = find_next_link_closing_index(t, closing_index + 4)
  local content
    = collect_link_content(t, opening_index + 1, closing_index - 1)
  local r = self.lookup_reference(content)

  if r then
    local inline_content
      = resolve_inline_following_content(t, closing_index, false,
                                         t.match_link_attributes)
    r.attributes
      = join_attributes(r.attributes, inline_content.attributes)
    render_link_or_image(t, opening_index, next_link_closing_index,
                         closing_index, r)
  end
end

```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```
local function process_links_and_emphasis(t)
  for _,value in ipairs(t) do
    value.is_active = true
  end

  for i,value in ipairs(t) do
    if not value.is_closing
        or value.type ~= "delimiter"
        or not ( value.element == "link"
              or value.element == "image"
              or value.element == "note")
        or value.removed then
      goto continue
    end

    local opener_position = find_link_opener(t, 1, i - 1)
    if (opener_position == nil) then
      goto continue
    end

    local opening_delimiter = t[opener_position]
```

```
9177        opening_delimiter.removed = true
9178
9179        local link_type = opening_delimiter.link_type
9180
9181        if (link_type == "inline") then
9182          resolve_inline_link(t, opener_position, i)
9183        end
9184        if (link_type == "shortcut") then
9185          resolve_shortcut_link(t, opener_position, i)
9186        end
9187        if (link_type == "full") then
9188          resolve_full_link(t, opener_position, i)
9189        end
9190        if (link_type == "collapsed") then
9191          resolve_collapsed_link(t, opener_position, i)
9192        end
9193        if (link_type == "note_inline") then
9194          resolve_note_inline_link(t, opener_position, i)
9195        end
9196        if (link_type == "raw_note") then
9197          resolve_raw_note_link(t, opener_position, i)
9198        end
9199
9200        ::continue::
9201      end
9202
9203      t[#t].content = t[#t].content:gsub("%s*$","")
9204
9205      process_emphasis(t, 1, #t)
9206      local final_result = collect_emphasis_content(t, 1, #t)
9207      return final_result
9208    end
9209
9210    function self.defer_link_and_emphasis_processing(delimiter_table)
9211      return writer.defer_call(function()
9212        return process_links_and_emphasis(delimiter_table)
9213      end)
9214    end
9215
```

3.1.6.8 Inline Elements (local)

```
9216    parsers.Str      = ( parsers.normalchar
9217                       * (parsers.normalchar + parsers.at)^0)
9218                     / writer.string
9219
9220    parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
```

```
                  / writer.string

parsers.Ellipsis = P("...") / writer.ellipsis

parsers.Smart    = parsers.Ellipsis

parsers.Code     = parsers.inticks / writer.code

if options.blankBeforeBlockquote then
  parsers.bqstart = parsers.fail
else
  parsers.bqstart = parsers.blockquote_start
end

if options.blankBeforeHeading then
  parsers.headerstart = parsers.fail
else
  parsers.headerstart = parsers.atx_heading
end

if options.blankBeforeList then
  parsers.interrupting_bullets = parsers.fail
  parsers.interrupting_enumerators = parsers.fail
else
  parsers.interrupting_bullets
    = parsers.bullet(parsers.dash, true)
    + parsers.bullet(parsers.asterisk, true)
    + parsers.bullet(parsers.plus, true)

  parsers.interrupting_enumerators
    = parsers.enumerator(parsers.period, true)
    + parsers.enumerator(parsers.rparent, true)
end

if options.html then
  parsers.html_interrupting
    = parsers.check_trail
    * ( parsers.html_incomplete_open_tag
      + parsers.html_incomplete_close_tag
      + parsers.html_incomplete_open_special_tag
      + parsers.html_comment_start
      + parsers.html_cdatasection_start
      + parsers.html_declaration_start
      + parsers.html_instruction_start
      - parsers.html_close_special_tag
      - parsers.html_empty_special_tag)
else
```

```
  parsers.html_interrupting = parsers.fail
end

parsers.EndlineExceptions
                   = parsers.blankline -- paragraph break
                   + parsers.eof       -- end of document
                   + parsers.bqstart
                   + parsers.thematic_break_lines
                   + parsers.interrupting_bullets
                   + parsers.interrupting_enumerators
                   + parsers.headerstart
                   + parsers.html_interrupting

parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions

parsers.endline = parsers.newline
                * (parsers.check_minimal_indent
                  * -V("EndlineExceptions")
                  + parsers.check_optional_indent
                  * -V("EndlineExceptions")
                  * -parsers.starter) / function(_) return end
                * parsers.spacechar^0

parsers.Endline = parsers.endline
                / writer.soft_line_break

parsers.EndlineNoSub = parsers.endline

parsers.NoSoftLineBreakEndline
                      = parsers.newline
                      * (parsers.check_minimal_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        + parsers.check_optional_indent
                        * -V("NoSoftLineBreakEndlineExceptions")
                        * -parsers.starter)
                      * parsers.spacechar^0
                      / writer.space

parsers.EndlineBreak = parsers.backslash * parsers.Endline
                                         / writer.hard_line_break

parsers.OptionalIndent
                   = parsers.spacechar^1 / writer.space

parsers.Space      = parsers.spacechar^2 * parsers.Endline
                                         / writer.hard_line_break
                   + parsers.spacechar^1
```

```
                    * parsers.Endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.Endline
                                          / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs

  parsers.NoSoftLineBreakSpace
                    = parsers.spacechar^2 * parsers.Endline
                                          / writer.hard_line_break
                    + parsers.spacechar^1
                    * parsers.Endline^-1
                    * parsers.eof / self.expandtabs
                    + parsers.spacechar^1 * parsers.Endline
                                          / writer.soft_line_break
                    + parsers.spacechar^1
                    * -parsers.newline / self.expandtabs

  parsers.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

  parsers.AutoLinkEmail
                      = parsers.auto_link_email
                      / self.auto_link_email

  parsers.AutoLinkRelativeReference
                      = parsers.auto_link_relative_reference
                      / self.auto_link_url

  parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
                      / self.defer_link_and_emphasis_processing

  parsers.EscapedChar = parsers.backslash
                      * C(parsers.escapable) / writer.string

  parsers.InlineHtml = Cs(parsers.html_inline_comment)
                     / writer.inline_html_comment
                     + Cs(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)
                       / writer.inline_html_tag

  parsers.HtmlEntity = parsers.html_entities / writer.string
```

3.1.6.9 Block Elements (local)

```
  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

  parsers.indented_non_blank_line = parsers.indentedline
                                  - parsers.blankline
```

```
9398
9399  parsers.Verbatim
9400    = Cs( parsers.check_code_trail
9401        * (parsers.line - parsers.blankline)
9402        * (( parsers.check_minimal_blank_indent_and_full_code_trail
9403           * parsers.blankline)^0
9404          * ( (parsers.check_minimal_indent / "")
9405            * parsers.check_code_trail
9406            * (parsers.line - parsers.blankline))^1)^0)
9407    / self.expandtabs / writer.verbatim
9408
9409  parsers.Blockquote    = parsers.blockquote_body
9410                        / writer.blockquote
9411
9412  parsers.ThematicBreak = parsers.thematic_break_lines
9413                         / writer.thematic_break
9414
9415  parsers.Reference     = parsers.define_reference_parser
9416                        / self.register_link
9417
9418  parsers.Paragraph     = parsers.freeze_trail
9419                        * (Ct((parsers.Inline)^1)
9420                        * (parsers.newline + parsers.eof)
9421                        * parsers.unfreeze_trail
9422                        / writer.paragraph)
9423
9424  parsers.Plain         = parsers.nonindentspace * Ct(parsers.Inline^1)
9425                        / writer.plain
```

3.1.6.10 Lists (local)

```
9426
9427  if options.taskLists then
9428    parsers.tickbox = ( parsers.ticked_box
9429                      + parsers.halfticked_box
9430                      + parsers.unticked_box
9431                      ) / writer.tickbox
9432  else
9433     parsers.tickbox = parsers.fail
9434  end
9435
9436  parsers.list_blank = parsers.conditionally_indented_blankline
9437
9438  parsers.ref_or_block_list_separated
9439    = parsers.sep_group_no_output(parsers.list_blank)
9440    * parsers.minimally_indented_ref
9441    + parsers.block_sep_group(parsers.list_blank)
```

```
    * parsers.minimally_indented_block

  parsers.ref_or_block_non_separated
    = parsers.minimally_indented_ref
    + (parsers.succeed / writer.interblocksep)
    * parsers.minimally_indented_block
    - parsers.minimally_indented_blankline

  parsers.tight_list_loop_body_pair =
    parsers.create_loop_body_pair(
      parsers.ref_or_block_non_separated,
      parsers.minimally_indented_par_or_plain_no_blank,
      (parsers.succeed / writer.interblocksep),
      (parsers.succeed / writer.paragraphsep))

  parsers.loose_list_loop_body_pair =
    parsers.create_loop_body_pair(
      parsers.ref_or_block_list_separated,
      parsers.minimally_indented_par_or_plain,
      parsers.block_sep_group(parsers.list_blank),
      parsers.par_sep_group(parsers.list_blank))

  parsers.tight_list_content_loop
    = V("Block")
    * parsers.tight_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_non_separated
    * parsers.tight_list_loop_body_pair.block^0
    +  (V("Paragraph") + V("Plain"))
    * parsers.tight_list_loop_body_pair.par^0

  parsers.loose_list_content_loop
    = V("Block")
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.ref_or_block_list_separated
    * parsers.loose_list_loop_body_pair.block^0
    + (V("Paragraph") + V("Plain"))
    * parsers.loose_list_loop_body_pair.par^0

  parsers.list_item_tightness_condition
    = -#( parsers.list_blank^0
        * parsers.minimally_indented_ref_or_block_or_par)
    * remove_indent("li")
    + remove_indent("li")
    * parsers.fail
```

```
parsers.indented_content_tight
  = Ct( (parsers.blankline / "")
      * #parsers.list_blank
      * remove_indent("li")
      + ( (V("Reference") + (parsers.blankline / ""))
        * parsers.check_minimal_indent
        * parsers.tight_list_content_loop
        + (V("Reference") + (parsers.blankline / ""))
        + (parsers.tickbox^-1 / writer.escape)
        * parsers.tight_list_content_loop
        )
      * parsers.list_item_tightness_condition)

parsers.indented_content_loose
  = Ct( (parsers.blankline / "")
      * #parsers.list_blank
      + ( (V("Reference") + (parsers.blankline / ""))
        * 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)
  return  -parsers.ThematicBreak
          * parsers.add_indent(starter, "li")
          * parsers.indented_content_tight
end

parsers.LooseListItem = function(starter)
  return  -parsers.ThematicBreak
          * parsers.add_indent(starter, "li")
          * parsers.indented_content_loose
          * remove_indent("li")
end

parsers.BulletListOfType = function(bullet_type)
  local bullet = parsers.bullet(bullet_type)
  return  ( Ct( parsers.TightListItem(bullet)
              * ( (parsers.check_minimal_indent / "")
                * parsers.TightListItem(bullet)
                )^0
            )
            * Cc(true)
            * -#( (parsers.list_blank^0 / "")
                * parsers.check_minimal_indent
                * (bullet - parsers.ThematicBreak)
```

```
              )
              + Ct( parsers.LooseListItem(bullet)
                  * ( (parsers.list_blank^0 / "")
                    * (parsers.check_minimal_indent / "")
                    * parsers.LooseListItem(bullet)
                    )^0
              )
              * Cc(false)
            ) / writer.bulletlist
end

parsers.BulletList = parsers.BulletListOfType(parsers.dash)
                   + parsers.BulletListOfType(parsers.asterisk)
                   + parsers.BulletListOfType(parsers.plus)

local function ordered_list(items,tight,starter)
  local startnum = starter[2][1]
  if options.startNumber then
    startnum = tonumber(startnum) or 1  -- fallback for '#'
    if startnum ~= nil then
      startnum = math.floor(startnum)
    end
  else
    startnum = nil
  end
  return writer.orderedlist(items,tight,startnum)
end

parsers.OrderedListOfType = function(delimiter_type)
  local enumerator = parsers.enumerator(delimiter_type)
  return  Cg(enumerator, "listtype")
        * (Ct( parsers.TightListItem(Cb("listtype"))
             * ( (parsers.check_minimal_indent / "")
               * parsers.TightListItem(enumerator))^0)
        * Cc(true)
        * -#((parsers.list_blank^0 / "")
            * parsers.check_minimal_indent * enumerator)
        + Ct( parsers.LooseListItem(Cb("listtype"))
            * ((parsers.list_blank^0 / "")
              * (parsers.check_minimal_indent / "")
              * parsers.LooseListItem(enumerator))^0)
        * Cc(false)
        ) * Ct(Cb("listtype")) / ordered_list
end

parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
                    + parsers.OrderedListOfType(parsers.rparent)
```

3.1.6.11 Blank (local)

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

3.1.6.12 Headings (local)

```
function parsers.parse_heading_text(s)
  local inlines = self.parser_functions.parse_inlines(s)
  local flatten_inlines = self.writer.flatten_inlines
  self.writer.flatten_inlines = true
  local flat_text = self.parser_functions.parse_inlines(s)
  flat_text = util.rope_to_string(flat_text)
  self.writer.flatten_inlines = flatten_inlines
  return {flat_text, inlines}
end

-- parse atx header
parsers.AtxHeading = 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))
                     / strip_atx_end
                     / 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.

```
  local current_extension_name = nil
  self.insert_pattern = function(selector, pattern, pattern_name)
    assert(pattern_name == nil or type(pattern_name) == "string")
    local _, _, lhs, pos, rhs
      = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
    assert(lhs ~= nil,
      [[Expected selector in form ]]
      .. [["LHS (before|after|instead of) RHS", not "]]
      .. selector .. [["]])
    assert(walkable_syntax[lhs] ~= nil,
      [[Rule ]] .. lhs
      .. [[ -> ... does not exist in markdown grammar]])
    assert(pos == "before" or pos == "after" or pos == "instead of",
      [[Expected positional specifier "before", "after", ]]
      .. [[or "instead of", not "]]
      .. pos .. [["]])
```

```
      local rule = walkable_syntax[lhs]
      local index = nil
      for current_index, current_rhs in ipairs(rule) do
        if type(current_rhs) == "string" and current_rhs == rhs then
          index = current_index
          if pos == "after" then
            index = index + 1
          end
          break
        end
      end
      assert(index ~= nil,
        [[Rule ]] .. lhs .. [[ -> ]] .. rhs
          .. [[ does not exist in markdown grammar]])
      local accountable_pattern
      if current_extension_name then
        accountable_pattern
          = {pattern, current_extension_name, pattern_name}
      else
        assert(type(pattern) == "string",
          [[reader->insert_pattern() was called outside ]]
          .. [[an extension with ]]
          .. [[a PEG pattern instead of a rule name]])
        accountable_pattern = pattern
      end
      if pos == "instead of" then
        rule[index] = accountable_pattern
      else
        table.insert(rule, index, accountable_pattern)
      end
    end
```

Create a local `syntax` hash table that stores those rules of the PEG grammar of markdown that can't be represented as an ordered choice of terminal symbols.

```
    local syntax =
      { "Blocks",

        Blocks = V("InitializeState")
               * V("ExpectedJekyllData")
               * V("Blank")^0
```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```
               * ( V("Block")
                 * ( V("Blank")^0 * parsers.eof
                   + ( V("Blank")^2 / writer.paragraphsep
                     + V("Blank")^0 / writer.interblocksep
```

```
                    )
                  )
                + ( V("Paragraph") + V("Plain") )
                * ( V("Blank")^0 * parsers.eof
                  + ( V("Blank")^2 / writer.paragraphsep
                    + V("Blank")^0 / writer.interblocksep
                    )
                  )
                * V("Block")
                * ( V("Blank")^0 * parsers.eof
                  + ( V("Blank")^2 / writer.paragraphsep
                    + V("Blank")^0 / writer.interblocksep
                    )
                  )
                + ( V("Paragraph") + V("Plain") )
                * ( V("Blank")^0 * parsers.eof
                  + V("Blank")^0 / writer.paragraphsep
                  )
                )^0,

        ExpectedJekyllData = parsers.succeed,

        Blank              = parsers.Blank,
        Reference          = parsers.Reference,

        Blockquote         = parsers.Blockquote,
        Verbatim           = parsers.Verbatim,
        ThematicBreak      = parsers.ThematicBreak,
        BulletList         = parsers.BulletList,
        OrderedList        = parsers.OrderedList,
        DisplayHtml        = parsers.DisplayHtml,
        Heading            = parsers.Heading,
        Paragraph          = parsers.Paragraph,
        Plain              = parsers.Plain,

        EndlineExceptions  = parsers.EndlineExceptions,
        NoSoftLineBreakEndlineExceptions
                           = parsers.NoSoftLineBreakEndlineExceptions,

        Str                = parsers.Str,
        Space              = parsers.Space,
        NoSoftLineBreakSpace
                           = parsers.NoSoftLineBreakSpace,
        OptionalIndent     = parsers.OptionalIndent,
        Endline            = parsers.Endline,
        EndlineNoSub       = parsers.EndlineNoSub,
        NoSoftLineBreakEndline
```

```
                          = parsers.NoSoftLineBreakEndline,
      EndlineBreak        = parsers.EndlineBreak,
      LinkAndEmph         = parsers.LinkAndEmph,
      Code                = parsers.Code,
      AutoLinkUrl         = parsers.AutoLinkUrl,
      AutoLinkEmail       = parsers.AutoLinkEmail,
      AutoLinkRelativeReference
                          = parsers.AutoLinkRelativeReference,
      InlineHtml          = parsers.InlineHtml,
      HtmlEntity          = parsers.HtmlEntity,
      EscapedChar         = parsers.EscapedChar,
      Smart               = parsers.Smart,
      Symbol              = parsers.Symbol,
      SpecialChar         = parsers.fail,
      InitializeState     = parsers.succeed,
    }
```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a function that accepts the current PEG pattern in `walkable_syntax[`left-hand side terminal symbol`]` if defined or `nil` otherwise and returns a PEG pattern that will (re)define `walkable_syntax[`left-hand side terminal symbol`]`.

```
  self.update_rule = function(rule_name, get_pattern)
    assert(current_extension_name ~= nil)
    assert(syntax[rule_name] ~= nil,
      [[Rule ]] .. rule_name
      .. [[ -> ... does not exist in markdown grammar]])
    local previous_pattern
    local extension_name
    if walkable_syntax[rule_name] then
      local previous_accountable_pattern
        = walkable_syntax[rule_name][1]
      previous_pattern = previous_accountable_pattern[1]
      extension_name
        = previous_accountable_pattern[2]
        .. ", " .. current_extension_name
    else
      previous_pattern = nil
      extension_name = current_extension_name
    end
    local pattern
```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define `walkable_syntax[`left-hand side terminal symbol`]` unless it has been previously defined.

```
function(previous_pattern)
  assert(previous_pattern == nil)
  return pattern
end
```

```
9780        if type(get_pattern) == "function" then
9781          pattern = get_pattern(previous_pattern)
9782        else
9783          assert(previous_pattern == nil,
9784                 [[Rule ]] .. rule_name ..
9785                 [[ has already been updated by ]] .. extension_name)
9786          pattern = get_pattern
9787        end
9788        local accountable_pattern = { pattern, extension_name, rule_name }
9789        walkable_syntax[rule_name] = { accountable_pattern }
9790      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.

```
9791      local special_characters = {}
9792      self.add_special_character = function(c)
9793        table.insert(special_characters, c)
9794        syntax.SpecialChar = S(table.concat(special_characters, ""))
9795      end
9796
9797      self.add_special_character("*")
9798      self.add_special_character("[")
9799      self.add_special_character("]")
9800      self.add_special_character("<")
9801      self.add_special_character("!")
9802      self.add_special_character("\\")
```

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

```
9803      self.initialize_named_group = function(name, value)
9804        local pattern = Ct("")
9805        if value ~= nil then
9806          pattern = pattern / value
9807        end
9808        syntax.InitializeState = syntax.InitializeState
9809                                 * Cg(pattern, name)
9810      end
```

Add a named group for indentation.

```
9811      self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
    for _, extension in ipairs(extensions) do
      current_extension_name = extension.name
      extension.extend_writer(writer)
      extension.extend_reader(self)
    end
    current_extension_name = nil
```

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

```
    if options.debugExtensions then
      local sorted_lhs = {}
      for lhs, _ in pairs(walkable_syntax) do
        table.insert(sorted_lhs, lhs)
      end
      table.sort(sorted_lhs)

      local output_lines = {"{"}
      for lhs_index, lhs in ipairs(sorted_lhs) do
        local encoded_lhs = util.encode_json_string(lhs)
        table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
        local rule = walkable_syntax[lhs]
        for rhs_index, rhs in ipairs(rule) do
          local human_readable_rhs
          if type(rhs) == "string" then
            human_readable_rhs = rhs
          else
            local pattern_name
            if rhs[3] then
              pattern_name = rhs[3]
            else
              pattern_name = "Anonymous Pattern"
            end
            local extension_name = rhs[2]
            human_readable_rhs = pattern_name .. [[ (]]
                                 .. extension_name .. [[)]]
          end
          local encoded_rhs
            = util.encode_json_string(human_readable_rhs)
          local output_line = [[        ]] .. encoded_rhs
          if rhs_index < #rule then
            output_line = output_line .. ","
          end
          table.insert(output_lines, output_line)
        end
        local output_line = "    ]"
        if lhs_index < #sorted_lhs then
```

```
9855            output_line = output_line .. ","
9856          end
9857          table.insert(output_lines, output_line)
9858        end
9859        table.insert(output_lines, "}")
9860
9861        local output = table.concat(output_lines, "\n")
9862        local output_filename = options.debugExtensionsFileName
9863        local output_file = assert(io.open(output_filename, "w"),
9864          [[Could not open file "]] .. output_filename
9865          .. [[" for writing]])
9866        assert(output_file:write(output))
9867        assert(output_file:close())
9868      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`.

```
9869      for lhs, rule in pairs(walkable_syntax) do
9870        syntax[lhs] = parsers.fail
9871        for _, rhs in ipairs(rule) do
9872          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.

```
9873          if type(rhs) == "string" then
9874            pattern = V(rhs)
9875          else
9876            pattern = rhs[1]
9877            if type(pattern) == "string" then
9878              pattern = V(pattern)
9879            end
9880          end
9881          syntax[lhs] = syntax[lhs] + pattern
9882        end
9883      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.

```
9884      if options.underscores then
9885        self.add_special_character("_")
9886      end
9887
9888      if not options.codeSpans then
```

```
9889        syntax.Code = parsers.fail
9890      else
9891        self.add_special_character("`")
9892      end
9893
9894      if not options.html then
9895        syntax.DisplayHtml = parsers.fail
9896        syntax.InlineHtml = parsers.fail
9897        syntax.HtmlEntity  = parsers.fail
9898      else
9899        self.add_special_character("&")
9900      end
9901
9902      if options.preserveTabs then
9903        options.stripIndent = false
9904      end
9905
9906      if not options.smartEllipses then
9907        syntax.Smart = parsers.fail
9908      else
9909        self.add_special_character(".")
9910      end
9911
9912      if not options.relativeReferences then
9913        syntax.AutoLinkRelativeReference = parsers.fail
9914      end
9915
9916      if options.contentLevel == "inline" then
9917        syntax[1] = "Inlines"
9918        syntax.Inlines = V("InitializeState")
9919                         * parsers.Inline^0
9920                         * ( parsers.spacing^0
9921                           * parsers.eof / "")
9922        syntax.Space = parsers.Space + parsers.blankline / writer.space
9923      end
9924
9925      local blocks_nested_t = util.table_copy(syntax)
9926      blocks_nested_t.ExpectedJekyllData = parsers.succeed
9927      parsers.blocks_nested = Ct(blocks_nested_t)
9928
9929      parsers.blocks = Ct(syntax)
9930
9931      local inlines_t = util.table_copy(syntax)
9932      inlines_t[1] = "Inlines"
9933      inlines_t.Inlines = V("InitializeState")
9934                         * parsers.Inline^0
9935                         * ( 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_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)
```

Unicode-normalize the input.

```
      if options.unicodeNormalization then
        local form = options.unicodeNormalizationForm
        if form == "nfc" then
          input = uni_algos.normalize.NFC(input)
        elseif form == "nfd" then
          input = uni_algos.normalize.NFD(input)
        elseif form == "nfkc" then
          input = uni_algos.normalize.NFKC(input)
        elseif form == "nfkd" then
          input = uni_algos.normalize.NFKD(input)
        else
          return writer.error(
            format("Unknown normalization form %s", form))
        end
      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.

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

Clear the table of references.

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

```
      local undosep_start, undosep_end
      local potential_secend_start, secend_start
      local potential_sep_start, sep_start
      while true do
        -- find a `writer->undosep`
        undosep_start, undosep_end
          = output:find(writer.undosep_text, 1, true)
        if undosep_start == nil then break end
        -- skip any preceding section ends
        secend_start = undosep_start
        while true do
          potential_secend_start = secend_start - #writer.secend_text
          if potential_secend_start < 1
             or output:sub(potential_secend_start,
                           secend_start - 1) ~= writer.secend_text
             then
            break
          end
          secend_start = potential_secend_start
        end
        -- find an immediately preceding
        -- block element / paragraph separator
        sep_start = secend_start
        potential_sep_start = sep_start - #writer.interblocksep_text
        if potential_sep_start >= 1
           and output:sub(potential_sep_start,
                          sep_start - 1) == writer.interblocksep_text
           then
          sep_start = potential_sep_start
        else
          potential_sep_start = sep_start - #writer.paragraphsep_text
          if potential_sep_start >= 1
             and output:sub(potential_sep_start,
                            sep_start - 1) == writer.paragraphsep_text
             then
```

```
            sep_start = potential_sep_start
          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 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

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

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

```
      function self.span(s, attr)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererBracketedSpanAttributeContextBegin",
                self.attributes(attr),
                s,
                "\\markdownRendererBracketedSpanAttributeContextEnd{}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local span_label  = parsers.lbracket
```

```
                            * (Cs((parsers.alphanumeric^1
                                + parsers.inticks
                                + parsers.autolink
                                + V("InlineHtml")
                                + ( parsers.backslash * parsers.backslash)
                                + ( parsers.backslash
                                  * (parsers.lbracket + parsers.rbracket)
                                  + V("Space") + V("Endline")
                                  + (parsers.any
                                    - ( parsers.newline
                                      + parsers.lbracket
                                      + parsers.rbracket
                                      + parsers.blankline^2))))^1)
                            / self.parser_functions.parse_inlines)
                            * parsers.rbracket

      local Span = span_label
                 * Ct(parsers.attributes)
                 / writer.span

      self.insert_pattern("Inline before LinkAndEmph",
                          Span, "Span")
    end
  }
end
```

3.1.7.2 Citations

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

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

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, the citations should be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.
- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.

- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```
10078       function self.citations(text_cites, cites)
10079         local buffer = {}
10080         if self.flatten_inlines then
10081           for _,cite in ipairs(cites) do
10082             if cite.prenote then
10083               table.insert(buffer, {cite.prenote, " "})
10084             end
10085             table.insert(buffer, cite.name)
10086             if cite.postnote then
10087               table.insert(buffer, {" ", cite.postnote})
10088             end
10089           end
10090         else
10091           table.insert(buffer,
10092                        {"\\markdownRenderer",
10093                         text_cites and "TextCite" or "Cite",
10094                         "{", #cites, "}"})
10095           for _,cite in ipairs(cites) do
10096             table.insert(buffer,
10097                          {cite.suppress_author and "-" or "+", "{",
10098                           cite.prenote or "", "}{",
10099                           cite.postnote or "", "}{", cite.name, "}"})
10100           end
10101         end
10102         return buffer
10103       end
10104     end, extend_reader = function(self)
10105       local parsers = self.parsers
10106       local writer = self.writer
10107
10108       local citation_chars
10109                     = parsers.alphanumeric
10110                     + S("#$%&-+<>~/_")
10111
10112       local citation_name
10113                     = Cs(parsers.dash^-1) * parsers.at
10114                     * Cs(citation_chars
10115                         * ((( citation_chars
10116                             + parsers.internal_punctuation
10117                             - parsers.comma - parsers.semicolon)
10118                            * -#(( parsers.internal_punctuation
10119                                 - parsers.comma
10120                                 - 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.

```
10256  local languages_json = (function()
10257    local base, prev, curr
10258    for _, pathname in ipairs{kpse.lookup(language_map,
10259                                          {all=true})} do
10260      local file = io.open(pathname, "r")
10261      if not file then goto continue end
10262      local input = assert(file:read("*a"))
10263      assert(file:close())
10264      local json = input:gsub('("[^\n]-"):','[%1]=')
10265      curr = load("_ENV = {}; return "..json)()
10266      if type(curr) == "table" then
10267        if base == nil then
10268          base = curr
10269        else
10270          setmetatable(prev, { __index = curr })
10271        end
10272        prev = curr
10273      end
10274      ::continue::
10275    end
10276    return base or {}
10277  end)()
10278
10279  return {
10280    name = "built-in content_blocks syntax extension",
10281    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.

```
10282      function self.contentblock(src,suf,type,tit)
10283        if not self.is_writing then return "" end
10284        src = src.."."..suf
10285        suf = suf:lower()
10286        if type == "onlineimage" then
10287          return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
10288                  "{",self.string(src),"}",
10289                  "{",self.uri(src),"}",
10290                  "{",self.string(tit or ""),"}"}
10291        elseif languages_json[suf] then
10292          return {"\\markdownRendererContentBlockCode{",suf,"}",
10293                  "{",self.string(languages_json[suf]),"}",
```

```
                "{",self.string(src),"}",
                "{",self.uri(src),"}",
                "{",self.string(tit or ""),"}"}
      else
        return {"\\markdownRendererContentBlock{",suf,"}",
                "{",self.string(src),"}",
                "{",self.uri(src),"}",
                "{",self.string(tit or ""),"}"}
      end
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local contentblock_tail
                = parsers.optionaltitle
                * (parsers.newline + parsers.eof)

    -- case insensitive online image suffix:
    local onlineimagesuffix
                = (function(...)
                    local parser = nil
                    for _, suffix in ipairs({...}) do
                      local pattern=nil
                      for i=1,#suffix do
                        local char=suffix:sub(i,i)
                        char = S(char:lower()..char:upper())
                        if pattern == nil then
                          pattern = char
                        else
                          pattern = pattern * char
                        end
                      end
                      if parser == nil then
                        parser = pattern
                      else
                        parser = parser + pattern
                      end
                    end
                    return parser
                  end)("png", "jpg", "jpeg", "gif", "tif", "tiff")

    -- online image url for iA Writer content blocks with
    -- mandatory suffix, allowing nested brackets:
    local onlineimageurl
                = (parsers.less
                  * Cs((parsers.anyescaped
```

```
10341                           - parsers.more
10342                           - parsers.spacing
10343                           - #(parsers.period
10344                              * onlineimagesuffix
10345                              * parsers.more
10346                              * contentblock_tail))^0)
10347                       * parsers.period
10348                       * Cs(onlineimagesuffix)
10349                       * parsers.more
10350                       + (Cs((parsers.inparens
10351                            + (parsers.anyescaped
10352                              - parsers.spacing
10353                              - parsers.rparent
10354                              - #(parsers.period
10355                                 * onlineimagesuffix
10356                                 * contentblock_tail)))^0)
10357                         * parsers.period
10358                         * Cs(onlineimagesuffix))
10359                       ) * Cc("onlineimage")
10360
10361       -- filename for iA Writer content blocks with mandatory suffix:
10362       local localfilepath
10363                     = parsers.slash
10364                     * Cs((parsers.anyescaped
10365                          - parsers.tab
10366                          - parsers.newline
10367                          - #(parsers.period
10368                             * parsers.alphanumeric^1
10369                             * contentblock_tail))^1)
10370                     * parsers.period
10371                     * Cs(parsers.alphanumeric^1)
10372                     * Cc("localfile")
10373
10374       local ContentBlock
10375                     = parsers.check_trail_no_rem
10376                     * (localfilepath + onlineimageurl)
10377                     * contentblock_tail
10378                     / writer.contentblock
10379
10380       self.insert_pattern("Block before Blockquote",
10381                           ContentBlock, "ContentBlock")
10382     end
10383   }
10384 end
```

3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

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

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

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1]
            = {"\\markdownRendererDlDefinitionBegin ",def,
               "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

      local defstart
        = parsers.check_trail_length(0) * defstartchar
        * #parsers.spacing
        * (parsers.tab + parsers.space^-3)
```

```
      + parsers.check_trail_length(1)
      * defstartchar * #parsers.spacing
      * (parsers.tab + parsers.space^-2)
      + parsers.check_trail_length(2)
      * defstartchar * #parsers.spacing
      * (parsers.tab + parsers.space^-1)
      + parsers.check_trail_length(3)
      * defstartchar * #parsers.spacing

    local indented_line
      = (parsers.check_minimal_indent / "")
      * parsers.check_code_trail * parsers.line

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

    local dlchunk = Cs(parsers.line * (indented_line - blank)^0)

    local indented_blocks = function(bl)
      return Cs( bl
            * (blank^1 * (parsers.check_minimal_indent / "")
              * parsers.check_code_trail * -parsers.blankline * bl)^0
            * (blank^1 + parsers.eof))
    end

    local function definition_list_item(term, defs, _)
      return { term = self.parser_functions.parse_inlines(term),
               definitions = defs }
    end

    local DefinitionListItemLoose
      = C(parsers.line) * blank^0
      * Ct((parsers.check_minimal_indent * (defstart
           * indented_blocks(dlchunk)
           / self.parser_functions.parse_blocks_nested))^1)
      * Cc(false) / definition_list_item

    local DefinitionListItemTight
      = C(parsers.line)
      * Ct((parsers.check_minimal_indent * (defstart * dlchunk
           / self.parser_functions.parse_blocks_nested))^1)
      * Cc(true) / definition_list_item

    local DefinitionList
      = ( Ct(DefinitionListItemLoose^1) * Cc(false)
        + Ct(DefinitionListItemTight^1)
```

```
          * (blank^0
            * -DefinitionListItemLoose * Cc(true))
          ) / writer.definitionlist

      self.insert_pattern("Block after Heading",
                          DefinitionList, "DefinitionList")
    end
  }
end
```

3.1.7.5 Fancy Lists

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

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

```

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.

```
      function self.fancylist(items,tight,startnum,numstyle,numdelim)
        if not self.is_writing then return "" end
        local buffer = {}
        local num = startnum
```

```
10490         for _,item in ipairs(items) do
10491           if item ~= "" then
10492             buffer[#buffer + 1] = self.fancyitem(item,num)
10493           end
10494           if num ~= nil and item ~= "" then
10495             num = num + 1
10496           end
10497         end
10498         local contents = util.intersperse(buffer,"\n")
10499         if tight and options.tightLists then
10500           return {"\\markdownRendererFancyOlBeginTight{",
10501                   numstyle,"}{",numdelim,"}",contents,
10502                   "\n\\markdownRendererFancyOlEndTight "}
10503         else
10504           return {"\\markdownRendererFancyOlBegin{",
10505                   numstyle,"}{",numdelim,"}",contents,
10506                   "\n\\markdownRendererFancyOlEnd "}
10507         end
10508       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.

```
10509       function self.fancyitem(s,num)
10510         if num ~= nil then
10511           return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
10512                   "\\markdownRendererFancyOlItemEnd "}
10513         else
10514           return {"\\markdownRendererFancyOlItem ",s,
10515                   "\\markdownRendererFancyOlItemEnd "}
10516         end
10517       end
10518     end, extend_reader = function(self)
10519       local parsers = self.parsers
10520       local options = self.options
10521       local writer = self.writer
10522
10523       local function combine_markers_and_delims(markers, delims)
10524         local markers_table = {}
10525         for _,marker in ipairs(markers) do
10526           local start_marker
10527           local continuation_marker
10528           if type(marker) == "table" then
10529             start_marker = marker[1]
10530             continuation_marker = marker[2]
10531           else
10532             start_marker = marker
```

```
10533             continuation_marker = marker
10534           end
10535           for _,delim in ipairs(delims) do
10536             table.insert(markers_table,
10537                          {start_marker, continuation_marker, delim})
10538           end
10539         end
10540         return markers_table
10541       end
10542
10543       local function join_table_with_func(func, markers_table)
10544         local pattern = func(table.unpack(markers_table[1]))
10545         for i = 2, #markers_table do
10546           pattern = pattern + func(table.unpack(markers_table[i]))
10547         end
10548         return pattern
10549       end
10550
10551       local lowercase_letter_marker = R("az")
10552       local uppercase_letter_marker = R("AZ")
10553
10554       local roman_marker = function(chars)
10555         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10556         local l, x, v, i
10557           = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
10558         return  m^-3
10559                 * (c*m + c*d + d^-1 * c^-3)
10560                 * (x*c + x*l + l^-1 * x^-3)
10561                 * (i*x + i*v + v^-1 * i^-3)
10562       end
10563
10564       local lowercase_roman_marker
10565         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10566       local uppercase_roman_marker
10567         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10568
10569       local lowercase_opening_roman_marker  = P("i")
10570       local uppercase_opening_roman_marker  = P("I")
10571
10572       local digit_marker = parsers.dig * parsers.dig^-8
10573
10574       local markers = {
10575         {lowercase_opening_roman_marker, lowercase_roman_marker},
10576         {uppercase_opening_roman_marker, uppercase_roman_marker},
10577         lowercase_letter_marker,
10578         uppercase_letter_marker,
10579         lowercase_roman_marker,
```

```
10580         uppercase_roman_marker,
10581         digit_marker
10582       }
10583
10584       local delims = {
10585         parsers.period,
10586         parsers.rparent
10587       }
10588
10589       local markers_table = combine_markers_and_delims(markers, delims)
10590
10591       local function enumerator(start_marker, _,
10592                                 delimiter_type, interrupting)
10593         local delimiter_range
10594         local allowed_end
10595         if interrupting then
10596           delimiter_range = P("1")
10597           allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10598         else
10599           delimiter_range = start_marker
10600           allowed_end = C(parsers.spacechar^1)
10601                       + #(parsers.newline + parsers.eof)
10602         end
10603
10604         return parsers.check_trail
10605                 * Ct(C(delimiter_range) * C(delimiter_type))
10606                 * allowed_end
10607       end
10608
10609       local starter = join_table_with_func(enumerator, markers_table)
10610
10611       local TightListItem = function(starter)
10612         return  parsers.add_indent(starter, "li")
10613                 * parsers.indented_content_tight
10614       end
10615
10616       local LooseListItem = function(starter)
10617         return  parsers.add_indent(starter, "li")
10618                 * parsers.indented_content_loose
10619                 * remove_indent("li")
10620       end
10621
10622       local function roman2number(roman)
10623         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10624                          ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10625         local numeral = 0
10626
```

```
10627         local i = 1
10628         local len = string.len(roman)
10629         while i < len do
10630           local z1, z2 = romans[ string.sub(roman, i, i) ],
10631                          romans[ string.sub(roman, i+1, i+1) ]
10632           if z1 < z2 then
10633               numeral = numeral + (z2 - z1)
10634               i = i + 2
10635           else
10636               numeral = numeral + z1
10637               i = i + 1
10638           end
10639         end
10640         if i <= len then
10641           numeral = numeral + romans[ string.sub(roman,i,i) ]
10642         end
10643         return numeral
10644       end
10645
10646       local function sniffstyle(numstr, delimend)
10647         local numdelim
10648         if delimend == ")" then
10649           numdelim = "OneParen"
10650         elseif delimend == "." then
10651           numdelim = "Period"
10652         else
10653           numdelim = "Default"
10654         end
10655
10656         local num
10657         num = numstr:match("^([I])$")
10658         if num then
10659           return roman2number(num), "UpperRoman", numdelim
10660         end
10661         num = numstr:match("^([i])$")
10662         if num then
10663           return roman2number(string.upper(num)), "LowerRoman", numdelim
10664         end
10665         num = numstr:match("^([A-Z])$")
10666         if num then
10667           return string.byte(num) - string.byte("A") + 1,
10668                  "UpperAlpha", numdelim
10669         end
10670         num = numstr:match("^([a-z])$")
10671         if num then
10672           return string.byte(num) - string.byte("a") + 1,
10673                  "LowerAlpha", numdelim
```

```
10674         end
10675         num = numstr:match("^([IVXLCDM]+)")
10676         if num then
10677           return roman2number(num), "UpperRoman", numdelim
10678         end
10679         num = numstr:match("^([ivxlcdm]+)")
10680         if num then
10681           return roman2number(string.upper(num)), "LowerRoman", numdelim
10682         end
10683         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10684       end
10685
10686       local function fancylist(items,tight,start)
10687         local startnum, numstyle, numdelim
10688           = sniffstyle(start[2][1], start[2][2])
10689         return writer.fancylist(items,tight,
10690                                 options.startNumber and startnum or 1,
10691                                 numstyle or "Decimal",
10692                                 numdelim or "Default")
10693       end
10694
10695       local FancyListOfType
10696         = function(start_marker, continuation_marker, delimiter_type)
10697           local enumerator_start
10698             = enumerator(start_marker, continuation_marker,
10699                          delimiter_type)
10700           local enumerator_cont
10701             = enumerator(continuation_marker, continuation_marker,
10702                          delimiter_type)
10703           return Cg(enumerator_start, "listtype")
10704                * (Ct( TightListItem(Cb("listtype"))
10705                    * ((parsers.check_minimal_indent / "")
10706                    * TightListItem(enumerator_cont))^0)
10707                * Cc(true)
10708                * -#((parsers.conditionally_indented_blankline^0 / "")
10709                    * parsers.check_minimal_indent * enumerator_cont)
10710                + Ct( LooseListItem(Cb("listtype"))
10711                    * ((parsers.conditionally_indented_blankline^0 / "")
10712                      * (parsers.check_minimal_indent / "")
10713                      * LooseListItem(enumerator_cont))^0)
10714                * Cc(false)
10715                ) * Ct(Cb("listtype")) / fancylist
10716         end
10717
10718       local FancyList
10719         = join_table_with_func(FancyListOfType, markers_table)
10720
```

```
      local Endline   = parsers.newline
                      * (parsers.check_minimal_indent
                        * -parsers.EndlineExceptions
                        + parsers.check_optional_indent
                        * -parsers.EndlineExceptions
                        * -starter)
                      * parsers.spacechar^0
                      / writer.soft_line_break

      self.update_rule("OrderedList", FancyList)
      self.update_rule("Endline", Endline)
    end
  }
end
```

3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```
M.extensions.fenced_code = function(blank_before_code_fence,
                                    allow_attributes,
                                    allow_raw_blocks)
  return {
    name = "built-in fenced_code syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.fencedCode(s, i, attr)
        if not self.is_writing then return "" end
        s = s:gsub("\n$", "")
        local buf = {}
        if attr ~= nil then
          table.insert(buf,
            {"\\markdownRendererFencedCodeAttributeContextBegin",
             self.attributes(attr)})
        end
        local name = util.cache_verbatim(options.cacheDir, s)
        table.insert(buf,
```

```
10754           {"\\markdownRendererInputFencedCode{",
10755            name,"}{",self.string(i),"}{",self.infostring(i),"}"})
10756         if attr ~= nil then
10757           table.insert(buf,
10758             "\\markdownRendererFencedCodeAttributeContextEnd{}")
10759         end
10760         return buf
10761       end
10762
```

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

```
10763       if allow_raw_blocks then
10764         function self.rawBlock(s, attr)
10765           if not self.is_writing then return "" end
10766           s = s:gsub("\n$", "")
10767           local name = util.cache_verbatim(options.cacheDir, s)
10768           return {"\\markdownRendererInputRawBlock{",
10769                   name,"}{", self.string(attr),"}"}
10770         end
10771       end
10772     end, extend_reader = function(self)
10773       local parsers = self.parsers
10774       local writer = self.writer
10775
10776       local function captures_geq_length(_,i,a,b)
10777         return #a >= #b and i
10778       end
10779
10780       local function strip_enclosing_whitespaces(str)
10781         return str:gsub("^%s*(.-)%s*$", "%1")
10782       end
10783
10784       local tilde_infostring = Cs(Cs((V("HtmlEntity")
10785                                       + parsers.anyescaped
10786                                       - parsers.newline)^0)
10787                                  / strip_enclosing_whitespaces)
10788
10789       local backtick_infostring
10790         = Cs( Cs((V("HtmlEntity")
10791             + ( -#(parsers.backslash * parsers.backtick)
10792               * parsers.anyescaped)
10793               - parsers.newline
10794               - parsers.backtick)^0)
10795         / strip_enclosing_whitespaces)
10796
10797       local fenceindent
```

```
10798
10799       local function has_trail(indent_table)
10800         return indent_table ~= nil and
10801           indent_table.trail ~= nil and
10802           next(indent_table.trail) ~= nil
10803       end
10804
10805       local function has_indents(indent_table)
10806         return indent_table ~= nil and
10807           indent_table.indents ~= nil and
10808           next(indent_table.indents) ~= nil
10809       end
10810
10811       local function get_last_indent_name(indent_table)
10812         if has_indents(indent_table) then
10813           return indent_table.indents[#indent_table.indents].name
10814         end
10815       end
10816
10817       local count_fenced_start_indent =
10818         function(_, _, indent_table, trail)
10819           local last_indent_name = get_last_indent_name(indent_table)
10820           fenceindent = 0
10821           if last_indent_name ~= "li" then
10822             fenceindent = #trail
10823           end
10824           return true
10825         end
10826
10827       local fencehead = function(char, infostring)
10828         return Cmt( Cb("indent_info")
10829                   * parsers.check_trail, count_fenced_start_indent)
10830              * Cg(char^3, "fencelength")
10831              * parsers.optionalspace
10832              * infostring
10833              * (parsers.newline + parsers.eof)
10834       end
10835
10836       local fencetail = function(char)
10837         return parsers.check_trail_no_rem
10838              * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10839              * parsers.optionalspace * (parsers.newline + parsers.eof)
10840              + parsers.eof
10841       end
10842
10843       local process_fenced_line =
10844         function(s, i, -- luacheck: ignore s i
```

```
10845                 indent_table, line_content, is_blank)
10846           local remainder = ""
10847           if has_trail(indent_table) then
10848             remainder = indent_table.trail.internal_remainder
10849           end
10850
10851           if is_blank
10852              and get_last_indent_name(indent_table) == "li" then
10853             remainder = ""
10854           end
10855
10856           local str = remainder .. line_content
10857           local index = 1
10858           local remaining = fenceindent
10859
10860           while true do
10861             local c = str:sub(index, index)
10862             if c == " " and remaining > 0 then
10863               remaining = remaining - 1
10864               index = index + 1
10865             elseif c == "\t" and remaining > 3 then
10866               remaining = remaining - 4
10867               index = index + 1
10868             else
10869               break
10870             end
10871           end
10872
10873           return true, str:sub(index)
10874         end
10875
10876       local fencedline = function(char)
10877         return Cmt( Cb("indent_info")
10878                   * C(parsers.line - fencetail(char))
10879                   * Cc(false), process_fenced_line)
10880       end
10881
10882       local blankfencedline
10883         = Cmt( Cb("indent_info")
10884              * C(parsers.blankline)
10885              * Cc(true), process_fenced_line)
10886
10887       local TildeFencedCode
10888         = fencehead(parsers.tilde, tilde_infostring)
10889         * Cs(( (parsers.check_minimal_blank_indent / "")
10890              * blankfencedline
10891              + ( parsers.check_minimal_indent / "")
```

```
            * fencedline(parsers.tilde))^0)
      * ( (parsers.check_minimal_indent / "")
        * fencetail(parsers.tilde) + parsers.succeed)

    local BacktickFencedCode
           = fencehead(parsers.backtick, backtick_infostring)
           * Cs(( (parsers.check_minimal_blank_indent / "")
                * blankfencedline
                + (parsers.check_minimal_indent / "")
                * fencedline(parsers.backtick))^0)
           * ( (parsers.check_minimal_indent / "")
             * fencetail(parsers.backtick) + parsers.succeed)

    local infostring_with_attributes
                      = Ct(C((parsers.linechar
                            - ( parsers.optionalspace
                              * parsers.attributes))^0)
                           * parsers.optionalspace
                           * Ct(parsers.attributes))

    local FencedCode
      = ((TildeFencedCode + BacktickFencedCode)
      / function(infostring, code)
        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()
```

```
10973         return self.pop_attributes("div")
10974       end
10975     end, extend_reader = function(self)
10976       local parsers = self.parsers
10977       local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
10978       local fenced_div_infostring
10979                               = C((parsers.linechar
10980                                  - ( parsers.spacechar^1
10981                                    * parsers.colon^1))^1)
10982
10983       local fenced_div_begin = parsers.nonindentspace
10984                              * parsers.colon^3
10985                              * parsers.optionalspace
10986                              * fenced_div_infostring
10987                              * ( parsers.spacechar^1
10988                                * parsers.colon^1)^0
10989                              * parsers.optionalspace
10990                              * (parsers.newline + parsers.eof)
10991
10992       local fenced_div_end = parsers.nonindentspace
10993                            * parsers.colon^3
10994                            * parsers.optionalspace
10995                            * (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.

```
10996       self.initialize_named_group("fenced_div_level", "0")
10997       self.initialize_named_group("fenced_div_num_opening_indents")
10998
10999       local function increment_div_level()
11000         local push_indent_table =
11001           function(s, i, indent_table, -- luacheck: ignore s i
11002                    fenced_div_num_opening_indents, fenced_div_level)
11003             fenced_div_level = tonumber(fenced_div_level) + 1
11004             local num_opening_indents = 0
11005             if indent_table.indents ~= nil then
11006               num_opening_indents = #indent_table.indents
11007             end
11008             fenced_div_num_opening_indents[fenced_div_level]
11009               = num_opening_indents
11010             return true, fenced_div_num_opening_indents
```

```
11011           end
11012
11013         local increment_level =
11014           function(s, i, fenced_div_level) -- luacheck: ignore s i
11015             fenced_div_level = tonumber(fenced_div_level) + 1
11016             return true, tostring(fenced_div_level)
11017           end
11018
11019         return Cg( Cmt( Cb("indent_info")
11020                       * Cb("fenced_div_num_opening_indents")
11021                       * Cb("fenced_div_level"), push_indent_table)
11022                  , "fenced_div_num_opening_indents")
11023              * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11024                  , "fenced_div_level")
11025       end
11026
11027       local function decrement_div_level()
11028         local pop_indent_table =
11029           function(s, i, -- luacheck: ignore s i
11030                    fenced_div_indent_table, fenced_div_level)
11031             fenced_div_level = tonumber(fenced_div_level)
11032             fenced_div_indent_table[fenced_div_level] = nil
11033             return true, tostring(fenced_div_level - 1)
11034           end
11035
11036         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11037                       * Cb("fenced_div_level"), pop_indent_table)
11038                  , "fenced_div_level")
11039       end
11040
11041
11042       local non_fenced_div_block
11043         = parsers.check_minimal_indent * V("Block")
11044         - parsers.check_minimal_indent_and_trail * fenced_div_end
11045
11046       local non_fenced_div_paragraph
11047         = parsers.check_minimal_indent * V("Paragraph")
11048         - parsers.check_minimal_indent_and_trail * fenced_div_end
11049
11050       local blank = parsers.minimally_indented_blank
11051
11052       local block_separated = parsers.block_sep_group(blank)
11053                             * non_fenced_div_block
11054
11055       local loop_body_pair
11056         = parsers.create_loop_body_pair(block_separated,
11057                                         non_fenced_div_paragraph,
```

```
                                        parsers.block_sep_group(blank),
                                        parsers.par_sep_group(blank))

    local content_loop  = ( non_fenced_div_block
                          * loop_body_pair.block^0
                          + non_fenced_div_paragraph
                          * block_separated
                          * loop_body_pair.block^0
                          + non_fenced_div_paragraph
                          * loop_body_pair.par^0)
                        * blank^0

    local FencedDiv = fenced_div_begin
                    / function (infostring)
                        local attr
                          = lpeg.match(Ct(parsers.attributes),
                                       infostring)
                        if attr == nil then
                          attr = {"." .. infostring}
                        end
                        return attr
                      end
                    / writer.div_begin
                    * increment_div_level()
                    * parsers.skipblanklines
                    * Ct(content_loop)
                    * parsers.minimally_indented_blank^0
                    * parsers.check_minimal_indent_and_trail
                    * fenced_div_end
                    * decrement_div_level()
                    * (Cc("") / writer.div_end)

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

    self.add_special_character(":")
```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```
    local function is_inside_div()
      local check_div_level =
        function(s, i, fenced_div_level) -- luacheck: ignore s i
          fenced_div_level = tonumber(fenced_div_level)
          return fenced_div_level > 0
        end
```

```
11101
11102         return Cmt(Cb("fenced_div_level"), check_div_level)
11103       end
11104
11105       local function check_indent()
11106         local compare_indent =
11107           function(s, i, indent_table, -- luacheck: ignore s i
11108                    fenced_div_num_opening_indents, fenced_div_level)
11109             fenced_div_level = tonumber(fenced_div_level)
11110             local num_current_indents
11111               = ( indent_table.current_line_indents ~= nil and
11112                   #indent_table.current_line_indents) or 0
11113             local num_opening_indents
11114               = fenced_div_num_opening_indents[fenced_div_level]
11115             return num_current_indents == num_opening_indents
11116           end
11117
11118         return Cmt( Cb("indent_info")
11119                   * Cb("fenced_div_num_opening_indents")
11120                   * Cb("fenced_div_level"), compare_indent)
11121       end
11122
11123       local fencestart = is_inside_div()
11124                        * fenced_div_end
11125                        * check_indent()
11126
11127       if not blank_before_div_fence then
11128         self.update_rule("EndlineExceptions", function(previous_pattern)
11129           if previous_pattern == nil then
11130             previous_pattern = parsers.EndlineExceptions
11131           end
11132           return previous_pattern + fencestart
11133         end)
11134       end
11135     end
11136   }
11137 end
```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```
11138 M.extensions.header_attributes = function()
11139   return {
11140     name = "built-in header_attributes syntax extension",
11141     extend_writer = function()
11142     end, extend_reader = function(self)
```

```
11143       local parsers = self.parsers
11144       local writer = self.writer
11145
11146       local function strip_atx_end(s)
11147         return s:gsub("%s+#*%s*$","")
11148       end
11149
11150       local AtxHeading = Cg(parsers.heading_start, "level")
11151                        * parsers.optionalspace
11152                        * (C(((parsers.linechar
11153                               - (parsers.attributes
11154                                 * parsers.optionalspace
11155                                 * parsers.newline))
11156                              * (parsers.linechar
11157                                - parsers.lbrace)^0)^1)
11158                              / strip_atx_end
11159                              / parsers.parse_heading_text)
11160                        * Cg(Ct(parsers.newline
11161                               + (parsers.attributes
11162                                 * parsers.optionalspace
11163                                 * parsers.newline)), "attributes")
11164                        * Cb("level")
11165                        * Cb("attributes")
11166                        / writer.heading
11167
11168       local function strip_trailing_spaces(s)
11169         return s:gsub("%s*$","")
11170       end
11171
11172       local heading_line  = (parsers.linechar
11173                             - (parsers.attributes
11174                               * parsers.optionalspace
11175                               * parsers.newline))^1
11176                           - parsers.thematic_break_lines
11177
11178       local heading_text
11179         = heading_line
11180         * ( (V("Endline") / "\n")
11181           * (heading_line - parsers.heading_level))^0
11182         * parsers.newline^-1
11183
11184       local SetextHeading
11185         = parsers.freeze_trail * parsers.check_trail_no_rem
11186         * #(heading_text
11187            * (parsers.attributes
11188              * parsers.optionalspace
11189              * parsers.newline)^-1
```

```
            * parsers.check_minimal_indent
            * parsers.check_trail
            * parsers.heading_level)
        * Cs(heading_text) / strip_trailing_spaces
        / parsers.parse_heading_text
        * Cg(Ct((parsers.attributes
              * parsers.optionalspace
              * parsers.newline)^-1), "attributes")
        * parsers.check_minimal_indent_and_trail * parsers.heading_level
        * Cb("attributes")
        * parsers.newline
        * parsers.unfreeze_trail
        / writer.heading

      local Heading = AtxHeading + SetextHeading
      self.update_rule("Heading", Heading)
    end
  }
end
```

3.1.7.9 Inline Code Attributes

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

```
M.extensions.inline_code_attributes = function()
  return {
    name = "built-in inline_code_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local writer = self.writer

      local CodeWithAttributes = parsers.inticks
                               * Ct(parsers.attributes)
                               / writer.code

      self.insert_pattern("Inline before Code",
                          CodeWithAttributes,
                          "CodeWithAttributes")
    end
  }
end
```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
M.extensions.line_blocks = function()
```

```
  return {
    name = "built-in line_blocks syntax extension",
    extend_writer = function(self)
```

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```
      function self.lineblock(lines)
        if not self.is_writing then return "" end
        local buffer = {}
        for i = 1, #lines - 1 do
          buffer[#buffer + 1] = { lines[i], self.hard_line_break }
        end
        buffer[#buffer + 1] = lines[#lines]

        return {"\\markdownRendererLineBlockBegin\n"
                  ,buffer,
                  "\n\\markdownRendererLineBlockEnd "}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      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
                 * (parsers.blankline/"")^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,
```

```
11350           "AutoLinkRelativeReferenceWithAttributes")
11351
11352       end
11353
11354     end
11355   }
11356 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.

```
11357 M.extensions.notes = function(notes, inline_notes)
11358   assert(notes or inline_notes)
11359   return {
11360     name = "built-in notes syntax extension",
11361     extend_writer = function(self)
```

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

```
11362       function self.note(s)
11363         if self.flatten_inlines then return "" end
11364         return {"\\markdownRendererNote{",s,"}"}
11365       end
11366     end, extend_reader = function(self)
11367       local parsers = self.parsers
11368       local writer = self.writer
11369
11370       local rawnotes = parsers.rawnotes
11371
11372       if inline_notes then
11373         local InlineNote
11374           = parsers.circumflex
11375           * ( parsers.link_label
11376             / self.parser_functions.parse_inlines_no_inline_note)
11377           / writer.note
11378
11379         self.insert_pattern("Inline after LinkAndEmph",
11380                             InlineNote, "InlineNote")
11381       end
11382       if notes then
11383         local function strip_first_char(s)
11384           return s:sub(2)
11385         end
11386
```

```
11387         local RawNoteRef
11388                       = #(parsers.lbracket * parsers.circumflex)
11389                       * parsers.link_label / strip_first_char
11390
11391         -- like indirect_link
11392         local function lookup_note(ref)
11393           return writer.defer_call(function()
11394             local found = rawnotes[self.normalize_tag(ref)]
11395             if found then
11396               return writer.note(
11397                 self.parser_functions.parse_blocks_nested(found))
11398             else
11399               return {"[",
11400                 self.parser_functions.parse_inlines("^" .. ref), "]"}
11401             end
11402           end)
11403         end
11404
11405         local function register_note(ref,rawnote)
11406           local normalized_tag = self.normalize_tag(ref)
11407           if rawnotes[normalized_tag] == nil then
11408             rawnotes[normalized_tag] = rawnote
11409           end
11410           return ""
11411         end
11412
11413         local NoteRef = RawNoteRef / lookup_note
11414
11415         local optionally_indented_line
11416           = parsers.check_optional_indent_and_any_trail * parsers.line
11417
11418         local blank
11419           = parsers.check_optional_blank_indent_and_any_trail
11420           * parsers.optionalspace * parsers.newline
11421
11422         local chunk
11423           = Cs(parsers.line
11424           * (optionally_indented_line - blank)^0)
11425
11426         local indented_blocks = function(bl)
11427           return Cs( bl
11428                 * ( blank^1 * (parsers.check_optional_indent / "")
11429                   * parsers.check_code_trail
11430                   * -parsers.blankline * bl)^0)
11431         end
11432
11433         local NoteBlock
```

```
                    = parsers.check_trail_no_rem
                    * RawNoteRef * parsers.colon
                    * parsers.spnlc * indented_blocks(chunk)
                    / register_note

      local Reference = NoteBlock + parsers.Reference

      self.update_rule("Reference", Reference)
      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
    return rectangular_rows
  end

  local function pipe_table_row(allow_empty_first_column
                                , nonempty_column
                                , column_separator
                                , column)
    local row_beginning
    if allow_empty_first_column then
```

```
      row_beginning = -- empty first column
                      #(parsers.spacechar^4
                        * column_separator)
                    * parsers.optionalspace
                    * column
                    * parsers.optionalspace
                    -- non-empty first column
                    + parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
    else
      row_beginning = parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
    end

    return Ct(row_beginning
              * (-- single column with no leading pipes
                 #(column_separator
                   * parsers.optionalspace
                   * parsers.newline)
                 * column_separator
                 * parsers.optionalspace
                 -- single column with leading pipes or
                 -- more than a single column
                 + (column_separator
                   * parsers.optionalspace
                   * column
                   * parsers.optionalspace)^1
                 * (column_separator
                   * parsers.optionalspace)^-1))
  end

  return {
    name = "built-in pipe_tables syntax extension",
    extend_writer = function(self)
```

Define `writer->table` as a function that will transform an input table to the output format, where `rows` is a sequence of columns and a column is a sequence of cell texts.

```
      function self.table(rows, caption, attributes)
        if not self.is_writing then return "" end
        local buffer = {}
        if attributes ~= nil then
          table.insert(buffer,
                       "\\markdownRendererTableAttributeContextBegin\n")
          table.insert(buffer, self.attributes(attributes))
```

```
        end
        table.insert(buffer,
                     {"\\markdownRendererTable{",
                      caption or "", "}{", #rows - 1, "}{",
                      #rows[1], "}"})
        local temp = rows[2] -- put alignments on the first row
        rows[2] = rows[1]
        rows[1] = temp
        for i, row in ipairs(rows) do
          table.insert(buffer, "{")
          for _, column in ipairs(row) do
            if i > 1 then -- do not use braces for alignments
              table.insert(buffer, "{")
            end
            table.insert(buffer, column)
            if i > 1 then
              table.insert(buffer, "}")
            end
          end
          table.insert(buffer, "}")
        end
        if attributes ~= nil then
          table.insert(buffer,
                       "\\markdownRendererTableAttributeContextEnd{}")
        end
        return buffer
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local table_hline_separator = parsers.pipe + parsers.plus

      local table_hline_column = (parsers.dash
                                  - #(parsers.dash
                                     * (parsers.spacechar
                                       + table_hline_separator
                                       + parsers.newline)))^1
                               * (parsers.colon * Cc("r")
                                 + parsers.dash * Cc("d"))
                               + parsers.colon
                               * (parsers.dash
                                 - #(parsers.dash
                                    * (parsers.spacechar
                                      + table_hline_separator
                                      + parsers.newline)))^1
                               * (parsers.colon * Cc("c")
```

```
                                + parsers.dash * Cc("l"))

    local table_hline = pipe_table_row(false
                                      , table_hline_column
                                      , table_hline_separator
                                      , table_hline_column)

    local table_caption_beginning
      = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
        * parsers.optionalspace * parsers.newline)^0
      * parsers.check_minimal_indent_and_trail
      * (P("Table")^-1 * parsers.colon)
      * parsers.optionalspace

    local function strip_trailing_spaces(s)
      return s:gsub("%s*$","")
    end

    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)
```

```
                              / 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.

```
11650       function self.rawInline(s, attr)
11651         if not self.is_writing then return "" end
11652         if self.flatten_inlines then return s end
11653         local name = util.cache_verbatim(options.cacheDir, s)
11654         return {"\\markdownRendererInputRawInline{",
11655                 name,"}{", self.string(attr),"}"}
11656       end
11657     end, extend_reader = function(self)
11658       local writer = self.writer
11659
11660       local RawInline = parsers.inticks
11661                       * parsers.raw_attribute
11662                       / writer.rawInline
11663
11664       self.insert_pattern("Inline before Code",
11665                           RawInline, "RawInline")
11666     end
11667   }
11668 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
11669 M.extensions.strike_through = function()
11670   return {
11671     name = "built-in strike_through syntax extension",
11672     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.

```
11673       function self.strike_through(s)
11674         if self.flatten_inlines then return s end
11675         return {"\\markdownRendererStrikeThrough{",s,"}"}
11676       end
11677     end, extend_reader = function(self)
11678       local parsers = self.parsers
11679       local writer = self.writer
11680
11681       local StrikeThrough = (
11682         parsers.between(parsers.Inline, parsers.doubletildes,
11683                         parsers.doubletildes)
11684       ) / writer.strike_through
11685
11686       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)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererSubscript{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Subscript = (
        parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
      ) / writer.subscript

      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.

```
11720       function self.superscript(s)
11721         if self.flatten_inlines then return s end
11722         return {"\\markdownRendererSuperscript{",s,"}"}
11723       end
11724     end, extend_reader = function(self)
11725       local parsers = self.parsers
11726       local writer = self.writer
11727
11728       local Superscript = (
11729         parsers.between(parsers.Str, parsers.circumflex,
11730                         parsers.circumflex)
11731       ) / writer.superscript
11732
11733       self.insert_pattern("Inline after LinkAndEmph",
11734                           Superscript, "Superscript")
11735
11736       self.add_special_character("^")
11737     end
11738   }
11739 end
```

3.1.7.19 TEX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
11740 M.extensions.tex_math = function(tex_math_dollars,
11741                                   tex_math_single_backslash,
11742                                   tex_math_double_backslash)
11743   return {
11744     name = "built-in tex_math syntax extension",
11745     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.

```
11746       function self.display_math(s)
11747         if self.flatten_inlines then return s end
11748         return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
11749       end
```

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

```
11750       function self.inline_math(s)
11751         if self.flatten_inlines then return s end
11752         return {"\\markdownRendererInlineMath{",self.math(s),"}"}
11753       end
```

```
11754     end, extend_reader = function(self)
11755       local parsers = self.parsers
11756       local writer = self.writer
11757
11758       local function between(p, starter, ender)
11759         return (starter * Cs(p * (p - ender)^0) * ender)
11760       end
11761
11762       local function strip_preceding_whitespaces(str)
11763         return str:gsub("^%s*(.-)$", "%1")
11764       end
11765
11766       local allowed_before_closing
11767         = B( parsers.backslash * parsers.any
11768            + parsers.any * (parsers.any - parsers.backslash))
11769
11770       local allowed_before_closing_no_space
11771         = B( parsers.backslash * parsers.any
11772            + parsers.any * (parsers.nonspacechar - parsers.backslash))
11773
```

The following patterns implement the Pandoc dollar math syntax extension.

```
11774       local dollar_math_content
11775         = (parsers.newline * (parsers.check_optional_indent / "")
11776         + parsers.backslash^-1
11777         * parsers.linechar)
11778         - parsers.blankline^2
11779         - parsers.dollar
11780
11781       local inline_math_opening_dollars = parsers.dollar
11782                                         * #(parsers.nonspacechar)
11783
11784       local inline_math_closing_dollars
11785         = allowed_before_closing_no_space
11786         * parsers.dollar
11787         * -#(parsers.digit)
11788
11789       local inline_math_dollars = between(Cs( dollar_math_content),
11790                                           inline_math_opening_dollars,
11791                                           inline_math_closing_dollars)
11792
11793       local display_math_opening_dollars  = parsers.dollar
11794                                           * parsers.dollar
11795
11796       local display_math_closing_dollars  = parsers.dollar
11797                                           * parsers.dollar
11798
11799       local display_math_dollars = between(Cs( dollar_math_content),
```

```
                                          display_math_opening_dollars,
                                          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 / writer.display_math
                  + inline_math / writer.inline_math

    self.insert_pattern("Inline after LinkAndEmph",
                        TexMath, "TexMath")

    if tex_math_dollars then
```

```
11887         self.add_special_character("$")
11888       end
11889
11890       if tex_math_single_backslash or tex_math_double_backslash then
11891         self.add_special_character("\\")
11892         self.add_special_character("[")
11893         self.add_special_character("]")
11894         self.add_special_character(")")
11895         self.add_special_character("(")
11896       end
11897     end
11898   }
11899 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.

```
11900 M.extensions.jekyll_data = function(expect_jekyll_data,
11901                                       ensure_jekyll_data)
11902   return {
11903     name = "built-in jekyll_data syntax extension",
11904     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.

```
11905       function self.jekyllData(d, t, p)
11906         if not self.is_writing then return "" end
11907
11908         local buf = {}
11909
11910         local keys = {}
11911         for k, _ in pairs(d) do
11912           table.insert(keys, k)
11913         end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
11914         table.sort(keys, function(first, second)
```

```
      if type(first) ~= type(second) then
        return type(first) < type(second)
      else
        return first < second
      end
    end)

    if not p then
      table.insert(buf, "\\markdownRendererJekyllDataBegin")
    end

    local is_sequence = false
    if #d > 0 and #d == #keys then
      for i=1, #d do
        if d[i] == nil then
          goto not_a_sequence
        end
      end
      is_sequence = true
    end
    ::not_a_sequence::

    if is_sequence then
      table.insert(buf,
        "\\markdownRendererJekyllDataSequenceBegin{")
      table.insert(buf, self.identifier(p or "null"))
      table.insert(buf, "}{")
      table.insert(buf, #keys)
      table.insert(buf, "}")
    else
      table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
      table.insert(buf, self.identifier(p or "null"))
      table.insert(buf, "}{")
      table.insert(buf, #keys)
      table.insert(buf, "}")
    end

    for _, k in ipairs(keys) do
      local v = d[k]
      local typ = type(v)
      k = tostring(k or "null")
      if typ == "table" and next(v) ~= nil then
        table.insert(
          buf,
          self.jekyllData(v, t, k)
        )
      else
```

```
11962             k = self.identifier(k)
11963             v = tostring(v)
11964             if typ == "boolean" then
11965               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11966               table.insert(buf, k)
11967               table.insert(buf, "}{")
11968               table.insert(buf, v)
11969               table.insert(buf, "}")
11970             elseif typ == "number" then
11971               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11972               table.insert(buf, k)
11973               table.insert(buf, "}{")
11974               table.insert(buf, v)
11975               table.insert(buf, "}")
11976             elseif typ == "string" then
11977               table.insert(buf,
11978                 "\\markdownRendererJekyllDataProgrammaticString{")
11979               table.insert(buf, k)
11980               table.insert(buf, "}{")
11981               table.insert(buf, self.identifier(v))
11982               table.insert(buf, "}")
11983               table.insert(buf,
11984                 "\\markdownRendererJekyllDataTypographicString{")
11985               table.insert(buf, k)
11986               table.insert(buf, "}{")
11987               table.insert(buf, t(v))
11988               table.insert(buf, "}")
11989             elseif typ == "table" then
11990               table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11991               table.insert(buf, k)
11992               table.insert(buf, "}")
11993             else
11994               local error = self.error(format(
11995                 "Unexpected type %s for value of "
11996                 .. "YAML key %s", typ, k))
11997               table.insert(buf, error)
11998             end
11999           end
12000         end
12001
12002         if is_sequence then
12003           table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12004         else
12005           table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12006         end
12007
12008         if not p then
```

```
          table.insert(buf, "\\markdownRendererJekyllDataEnd")
        end

        return buf
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local JekyllData
        = Cmt( C((parsers.line - P("---") - P("..."))^0)
             , function(s, i, text) -- luacheck: ignore s i
                 local data
                 local ran_ok, _ = pcall(function()
                   -- TODO: Use `require("tinyyaml")` in TeX Live 2023
                   local tinyyaml = require("markdown-tinyyaml")
                   data = tinyyaml.parse(text, {timestamps=false})
                 end)
                 if ran_ok and data ~= nil then
                   return true, writer.jekyllData(data, function(s)
                     return self.parser_functions.parse_blocks_nested(s)
                   end, nil)
                 else
                   return false
                 end
               end
             )

      local UnexpectedJekyllData
        = P("---")
        * parsers.blankline / 0
        -- if followed by blank, it's thematic break
        * #(-parsers.blankline)
        * JekyllData
        * (P("---") + P("..."))

      local ExpectedJekyllData
        = ( P("---")
          * parsers.blankline / 0
          -- if followed by blank, it's thematic break
          * #(-parsers.blankline)
          )^-1
        * JekyllData
        * (P("---") + P("..."))^-1

      if ensure_jekyll_data then
        ExpectedJekyllData = ExpectedJekyllData
```

```
                              * parsers.eof
      else
        ExpectedJekyllData = ( ExpectedJekyllData
                               * (V("Blank")^0 / writer.interblocksep)
                               )^-1
      end

      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 own function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

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

function M.new(options)
```

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

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

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```
  local parser_convert = nil
  return function(input)
    local function convert(input)
      if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
        local parser = require("markdown-parser")
        if metadata.version ~= parser.metadata.version then
          warn("markdown.lua " .. metadata.version .. " used with " ..
               "markdown-parser.lua " .. parser.metadata.version .. ".")
```

```
12087         end
12088         parser_convert = parser.new(options)
12089       end
12090       return parser_convert(input)
12091     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).

```
12092     local output
12093     if options.eagerCache or options.finalizeCache then
12094       local salt = util.salt(options)
12095       local name = util.cache(options.cacheDir, input, salt, convert,
12096                               ".md.tex")
12097       output = [[\input{]] .. name .. [[}\relax]]
```

Otherwise, return the result of the conversion directly.

```
12098     else
12099       output = convert(input)
12100     end
```

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

```
12101     if options.finalizeCache then
12102       local file, mode
12103       if options.frozenCacheCounter > 0 then
12104         mode = "a"
12105       else
12106         mode = "w"
12107       end
12108       file = assert(io.open(options.frozenCacheFileName, mode),
12109         [[Could not open file "]] .. options.frozenCacheFileName
12110         .. [[" for writing]])
12111       assert(file:write(
12112         [[\expandafter\global\expandafter\def\csname ]]
12113         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12114         .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
12115       assert(file:close())
12116     end
12117     return output
12118   end
12119 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` table.

```
  options = options or {}
  setmetatable(options, { __index = function (_, key)
    return defaultOptions[key] 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.bracketedSpans then
    local bracketed_spans_extension = M.extensions.bracketed_spans()
    table.insert(extensions, bracketed_spans_extension)
  end

  if options.contentBlocks then
    local content_blocks_extension = M.extensions.content_blocks(
      options.contentBlocksLanguageMap)
    table.insert(extensions, content_blocks_extension)
  end

  if options.definitionLists then
    local definition_lists_extension = M.extensions.definition_lists(
      options.tightLists)
```

```
12156     table.insert(extensions, definition_lists_extension)
12157   end
12158
12159   if options.fencedCode then
12160     local fenced_code_extension = M.extensions.fenced_code(
12161       options.blankBeforeCodeFence,
12162       options.fencedCodeAttributes,
12163       options.rawAttribute)
12164     table.insert(extensions, fenced_code_extension)
12165   end
12166
12167   if options.fencedDivs then
12168     local fenced_div_extension = M.extensions.fenced_divs(
12169       options.blankBeforeDivFence)
12170     table.insert(extensions, fenced_div_extension)
12171   end
12172
12173   if options.headerAttributes then
12174     local header_attributes_extension = M.extensions.header_attributes()
12175     table.insert(extensions, header_attributes_extension)
12176   end
12177
12178   if options.inlineCodeAttributes then
12179     local inline_code_attributes_extension =
12180       M.extensions.inline_code_attributes()
12181     table.insert(extensions, inline_code_attributes_extension)
12182   end
12183
12184   if options.jekyllData then
12185     local jekyll_data_extension = M.extensions.jekyll_data(
12186       options.expectJekyllData, options.ensureJekyllData)
12187     table.insert(extensions, jekyll_data_extension)
12188   end
12189
12190   if options.linkAttributes then
12191     local link_attributes_extension =
12192       M.extensions.link_attributes()
12193     table.insert(extensions, link_attributes_extension)
12194   end
12195
12196   if options.lineBlocks then
12197     local line_block_extension = M.extensions.line_blocks()
12198     table.insert(extensions, line_block_extension)
12199   end
12200
12201   if options.mark then
12202     local mark_extension = M.extensions.mark()
```

```
12203     table.insert(extensions, mark_extension)
12204   end
12205
12206   if options.pipeTables then
12207     local pipe_tables_extension = M.extensions.pipe_tables(
12208       options.tableCaptions, options.tableAttributes)
12209     table.insert(extensions, pipe_tables_extension)
12210   end
12211
12212   if options.rawAttribute then
12213     local raw_inline_extension = M.extensions.raw_inline()
12214     table.insert(extensions, raw_inline_extension)
12215   end
12216
12217   if options.strikeThrough then
12218     local strike_through_extension = M.extensions.strike_through()
12219     table.insert(extensions, strike_through_extension)
12220   end
12221
12222   if options.subscripts then
12223     local subscript_extension = M.extensions.subscripts()
12224     table.insert(extensions, subscript_extension)
12225   end
12226
12227   if options.superscripts then
12228     local superscript_extension = M.extensions.superscripts()
12229     table.insert(extensions, superscript_extension)
12230   end
12231
12232   if options.texMathDollars or
12233      options.texMathSingleBackslash or
12234      options.texMathDoubleBackslash then
12235     local tex_math_extension = M.extensions.tex_math(
12236       options.texMathDollars,
12237       options.texMathSingleBackslash,
12238       options.texMathDoubleBackslash)
12239     table.insert(extensions, tex_math_extension)
12240   end
12241
12242   if options.notes or options.inlineNotes then
12243     local notes_extension = M.extensions.notes(
12244       options.notes, options.inlineNotes)
12245     table.insert(extensions, notes_extension)
12246   end
12247
12248   if options.citations then
12249     local citations_extension
```

```
      = M.extensions.citations(options.citationNbsps)
    table.insert(extensions, citations_extension)
  end

  if options.fancyLists then
    local fancy_lists_extension = M.extensions.fancy_lists()
    table.insert(extensions, fancy_lists_extension)
  end
```

Apply user-defined syntax extensions based on `options.extensions`.

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

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

```
      local pathname = assert(kpse.find_file(filename),
        [[Could not locate user-defined syntax extension "]]
        .. filename)
      local input_file = assert(io.open(pathname, "r"),
        [[Could not open user-defined syntax extension "]]
        .. pathname .. [[" for reading]])
      local input = assert(input_file:read("*a"))
      assert(input_file:close())
      local user_extension, err = load([[
        local sandbox = {}
        setmetatable(sandbox, {__index = _G})
        _ENV = sandbox
      ]] .. input)()
      assert(user_extension,
        [[Failed to compile user-defined syntax extension "]]
        .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
      assert(user_extension.api_version ~= nil,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory field "api_version"]])
      assert(type(user_extension.api_version) == "number",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "api_version" of type "]]
        .. type(user_extension.api_version)
        .. [[" but "number" was expected]])
      assert(user_extension.api_version > 0
         and user_extension.api_version
          <= metadata.user_extension_api_version,
        [[User-defined syntax extension "]] .. pathname
        .. [[" uses syntax extension API version "]]
        .. user_extension.api_version .. [[ but markdown.lua ]]
        .. metadata.version .. [[ uses API version ]]
        .. metadata.user_extension_api_version
```

```
12292         .. [[, which is incompatible]])
12293
12294       assert(user_extension.grammar_version ~= nil,
12295         [[User-defined syntax extension "]] .. pathname
12296         .. [[" does not specify mandatory field "grammar_version"]])
12297       assert(type(user_extension.grammar_version) == "number",
12298         [[User-defined syntax extension "]] .. pathname
12299         .. [[" specifies field "grammar_version" of type "]]
12300         .. type(user_extension.grammar_version)
12301         .. [[" but "number" was expected]])
12302       assert(user_extension.grammar_version == metadata.grammar_version,
12303         [[User-defined syntax extension "]] .. pathname
12304         .. [[" uses grammar version "]]
12305         .. user_extension.grammar_version
12306         .. [[ but markdown.lua ]] .. metadata.version
12307         .. [[ uses grammar version ]] .. metadata.grammar_version
12308         .. [[, which is incompatible]])
12309
12310       assert(user_extension.finalize_grammar ~= nil,
12311         [[User-defined syntax extension "]] .. pathname
12312         .. [[" does not specify mandatory "finalize_grammar" field]])
12313       assert(type(user_extension.finalize_grammar) == "function",
12314         [[User-defined syntax extension "]] .. pathname
12315         .. [[" specifies field "finalize_grammar" of type "]]
12316         .. type(user_extension.finalize_grammar)
12317         .. [[" 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.)

```
12318       local extension = {
12319         name = [[user-defined "]] .. pathname .. [[" syntax extension]],
12320         extend_reader = user_extension.finalize_grammar,
12321         extend_writer = function() end,
12322       }
12323       return extension
12324     end)(user_extension_filename)
12325     table.insert(extensions, user_extension)
12326   end
```

Produce a conversion function from markdown to plain TEX.

```
12327   local writer = M.writer.new(options)
12328   local reader = M.reader.new(writer, options)
12329   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`.

```
12330   collectgarbage("collect")
```

Update the singleton cache.

```
  if options.singletonCache then
    local singletonCacheOptions = {}
    for k, v in pairs(options) do
      singletonCacheOptions[k] = v
    end
    setmetatable(singletonCacheOptions,
      { __index = function (_, key)
        return defaultOptions[key] end })
    singletonCache.options = singletonCacheOptions
    singletonCache.convert = convert
  end
```

Return the conversion function from markdown to plain TEX.

```
  return convert
end

return M
```

3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```

local input
if input_filename then
  local input_file = assert(io.open(input_filename, "r"),
    [[Could not open file "]] .. input_filename .. [[" for reading]])
  input = assert(input_file:read("*a"))
  assert(input_file:close())
else
  input = assert(io.read("*a"))
end

```

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

```
local lfs = require("lfs")
if options.cacheDir and not lfs.isdir(options.cacheDir) then
  assert(lfs.mkdir(options["cacheDir"]))
end
```

If Kpathsea has not been loaded before or if LuaTEX has not yet been initialized, configure Kpathsea on top of loading it.

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

```
    kpse.set_program_name("luatex")
  end
end)()
local md = require("markdown")
```

Since we are loading the rest of the Lua implementation dynamically, check that both the `markdown` module and the command line implementation are the same version.

```
if metadata.version ~= md.metadata.version then
  warn("markdown-cli.lua " .. metadata.version .. " used with " ..
       "markdown.lua " .. md.metadata.version .. ".")
end
local convert = md.new(options)
local output = convert(input)

if output_filename then
  local output_file = assert(io.open(output_filename, "w"),
    [[Could not open file "]] .. output_filename .. [[" for writing]])
  assert(output_file:write(output))
  assert(output_file:close())
else
  assert(io.write(output))
end
```

Remove the `options.cacheDir` directory if it is empty.

```
if options.cacheDir then
  lfs.rmdir(options["cacheDir"])
end
```

3.2 Plain TEX Implementation

The plain TEX implementation provides macros for the interfacing between TEX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain TEX exposed by the plain TEX interface (see Section 2.2).

3.2.1 Logging Facilities

```
\ExplSyntaxOn
\cs_if_free:NT
  \markdownInfo
  {
    \cs_new:Npn
      \markdownInfo #1
      {
        \msg_info:nne
          { markdown }
          { generic-message }
```

```
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownWarning
  {
    \cs_new:Npn
      \markdownWarning #1
      {
        \msg_warning:nne
          { markdown }
          { generic-message }
          { #1 }
      }
  }
\cs_if_free:NT
  \markdownError
  {
    \cs_new:Npn
      \markdownError #1 #2
      {
        \msg_error:nnee
          { markdown }
          { generic-message-with-help-text }
          { #1 }
          { #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { generic-message }
  { #1 }
\msg_new:nnnn
  { markdown }
  { generic-message-with-help-text }
  { #1 }
  { #2 }
\cs_generate_variant:Nn
  \msg_info:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_warning:nnn
  { nne }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnee }
\ExplSyntaxOff
```

3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain TEX themes provided with the Markdown package.

```
\ExplSyntaxOn
\prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
\cs_new:Nn
  \@@_plain_tex_load_theme:nn
  {
    \prop_get:NnNTF
      \g_@@_plain_tex_loaded_themes_linenos_prop
      { #1 }
      \l_tmpa_tl
      {
        \msg_warning:nnnV
          { markdown }
          { repeatedly-loaded-plain-tex-theme }
          { #1 }
          \l_tmpa_tl
      }
      {
        \msg_info:nnn
          { markdown }
          { loading-plain-tex-theme }
          { #1 }
        \prop_gput:Nnx
          \g_@@_plain_tex_loaded_themes_linenos_prop
          { #1 }
          { \tex_the:D \tex_inputlineno:D }
        \file_input:n
          { markdown theme #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-plain-tex-theme }
  { Loading~plain~TeX~Markdown~theme~#1 }
\msg_new:nnn
  { markdown }
  { repeatedly-loaded-plain-tex-theme }
  {
    Plain~TeX~Markdown~theme~#1~was~previously~
    loaded~on~line~#2,~not~loading~it~again
  }
\cs_generate_variant:Nn
  \prop_gput:Nnn
```

```
  { Nnx }
\cs_gset_eq:NN
  \@@_load_theme:nn
  \@@_plain_tex_load_theme:nn
\cs_generate_variant:Nn
  \@@_load_theme:nn
  { nV }
```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as LaTeX and ConTeXt.

```
\cs_new:Npn
  \markdownLoadPlainTeXTheme
  {
```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```
    \tl_set:NV
      \l_tmpa_tl
      \g_@@_current_theme_tl
    \tl_reverse:N
      \l_tmpa_tl
    \tl_set:Ne
      \l_tmpb_tl
      {
        \tl_tail:V
          \l_tmpa_tl
      }
    \tl_reverse:N
      \l_tmpb_tl
```

Next, we munge the theme name.

```
    \str_set:NV
      \l_tmpa_str
      \l_tmpb_tl
    \str_replace_all:Nnn
      \l_tmpa_str
      { / }
      { _ }
```

Finally, we load the plain TeX theme.

```
    \@@_plain_tex_load_theme:VV
      \l_tmpb_tl
      \l_tmpa_str
  }
\cs_generate_variant:Nn
  \tl_set:Nn
  { Ne }
```

```
\cs_generate_variant:Nn
  \@@_plain_tex_load_theme:nn
  { VV }
\ExplSyntaxOff
```

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

```
\markdownSetup {
  rendererPrototypes = {
    tilde = {~},
  },
}
```

The `witiko/markdown/defaults` plain TEX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererParagraphSeparatorPrototype{%
  \markdownRendererInterblockSeparator}%
\def\markdownRendererHardLineBreakPrototype{\hfil\break}%
\def\markdownRendererSoftLineBreakPrototype{ }%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`\{}%
\def\markdownRendererRightBracePrototype{\char`\}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\&}%
\def\markdownRendererUnderscorePrototype{\char`_}%
\def\markdownRendererHashPrototype{\char`\#}%
\def\markdownRendererCircumflexPrototype{\char`^}%
\def\markdownRendererBackslashPrototype{\char`\\}%
\def\markdownRendererTildePrototype{\char`~}%
\def\markdownRendererPipePrototype{|}%
\def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
\def\markdownRendererLinkPrototype#1#2#3#4{#2}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \markdownInput{#3}}%
\def\markdownRendererContentBlockOnlineImagePrototype{%
  \markdownRendererImage}%
\def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
  \markdownRendererInputFencedCode{#3}{#2}{#2}}%
\def\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
```

```
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererFancyOlBeginPrototype#1#2{%
  \markdownRendererOlBegin}%
\def\markdownRendererFancyOlBeginTightPrototype#1#2{%
  \markdownRendererOlBeginTight}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
\def\markdownRendererFancyOlItemWithNumberPrototype{%
  \markdownRendererOlItemWithNumber}%
\def\markdownRendererFancyOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
\def\markdownRendererFancyOlEndTightPrototype{%
  \markdownRendererOlEndTight}%
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{#1}%
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{\par}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{{\it#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
\def\markdownRendererLineBlockEndPrototype{\endgroup}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input#1\relax{}}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2#3{%
  \markdownRendererInputVerbatim{#1}}%
\def\markdownRendererHeadingOnePrototype#1{#1}%
\def\markdownRendererHeadingTwoPrototype#1{#1}%
\def\markdownRendererHeadingThreePrototype#1{#1}%
\def\markdownRendererHeadingFourPrototype#1{#1}%
\def\markdownRendererHeadingFivePrototype#1{#1}%
\def\markdownRendererHeadingSixPrototype#1{#1}%
\def\markdownRendererThematicBreakPrototype{}%
```

```
\def\markdownRendererNotePrototype#1{#1}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
\def\markdownRendererTickedBoxPrototype{[X]}%
\def\markdownRendererHalfTickedBoxPrototype{[/]}%
\def\markdownRendererUntickedBoxPrototype{[ ]}%
\def\markdownRendererStrikeThroughPrototype#1{#1}%
\def\markdownRendererSuperscriptPrototype#1{#1}%
\def\markdownRendererSubscriptPrototype#1{#1}%
\def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
\def\markdownRendererInlineMathPrototype#1{$#1$}%
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextBeginPrototype
  {
    \group_begin:
    \color_group_begin:
  }
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextEndPrototype
  {
    \color_group_end:
    \group_end:
  }
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset:Npn
  \markdownRendererReplacementCharacterPrototype
  { \codepoint_str_generate:n { fffd } }
\ExplSyntaxOff
\def\markdownRendererSectionBeginPrototype{}%
```

```
\def\markdownRendererSectionEndPrototype{}%
\let\markdownRendererWarningPrototype\markdownWarning
\let\markdownRendererErrorPrototype\markdownError
```

3.2.3.1 Raw Attributes

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

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_plain_tex_default_input_raw_inline:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline:nn
      { #1 }
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

3.2.3.2 YAML Metadata Renderer Prototypes

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

\c_@@_jekyll_data_sequence_tl The currently traversed branch of the YAML document contains a sequence at depth p.

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

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

```
\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 \markdown_jekyll_data_push_address_segment:n macro.

```
\seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
\cs_new:Nn \markdown_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 `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```
\tl_new:N  \g_@@_jekyll_data_wildcard_absolute_address_tl
\tl_new:N  \g_@@_jekyll_data_wildcard_relative_address_tl
\cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
  {
    \seq_pop_left:NN #1 \l_tmpa_tl
    \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
    \seq_put_left:NV #1 \l_tmpa_tl
  }
\cs_new:Nn \markdown_jekyll_data_update_address_tls:
  {
    \markdown_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 `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```
\cs_new:Nn \markdown_jekyll_data_push:nN
  {
    \markdown_jekyll_data_push_address_segment:n
      { #1 }
    \seq_put_right:NV
     \g_@@_jekyll_data_datatypes_seq
     #2
    \markdown_jekyll_data_update_address_tls:
  }
\cs_new:Nn \markdown_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
    \markdown_jekyll_data_update_address_tls:
  }
```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```
\cs_new:Nn \markdown_jekyll_data_set_keyval:nn
  {
    \keys_set_known:nn
      { markdown/jekyllData }
      { { #1 } = { #2 } }
  }
\cs_generate_variant:Nn
  \markdown_jekyll_data_set_keyval:nn
  { Vn }
\cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
  {
    \markdown_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_scalar_tl
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_absolute_address_tl
      { #2 }
```

```
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_relative_address_tl
      { #2 }
    \markdown_jekyll_data_pop:
  }
```

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

```
\def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_sequence_tl
}
\def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_mapping_tl
}
\def\markdownRendererJekyllDataSequenceEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataMappingEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataBooleanPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\def\markdownRendererJekyllDataEmptyPrototype#1{}
\def\markdownRendererJekyllDataNumberPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```
\def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
\def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\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
  {
    \ExplSyntaxOff
    \@@_if_option:nF
      { noDefaults }
      {
        \@@_setup:n
          {theme = witiko/markdown/defaults}
      }
    \ExplSyntaxOn
  }
\ExplSyntaxOff
```

3.2.4 Lua Snippets

After the \markdownPrepareLuaOptions macro has been fully expanded, the \markdownLuaOptions macro will expands to a Lua table that contains the plain TEX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```
\ExplSyntaxOn
\tl_new:N \g_@@_formatted_lua_options_tl
\cs_new:Nn \@@_format_lua_options:
  {
    \tl_gclear:N
      \g_@@_formatted_lua_options_tl
    \seq_map_function:NN
      \g_@@_lua_options_seq
      \@@_format_lua_option:n
  }
\cs_new:Nn \@@_format_lua_option:n
  {
    \@@_typecheck_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \bool_case_true:nF
      {
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
```

```
          \c_@@_option_type_boolean_tl ||
        \str_if_eq_p:VV
          \l_tmpa_tl
          \c_@@_option_type_number_tl ||
        \str_if_eq_p:VV
          \l_tmpa_tl
          \c_@@_option_type_counter_tl
      }
        {
          \@@_get_option_value:nN
            { #1 }
            \l_tmpa_tl
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { #1~=~  \l_tmpa_tl   ,~ }
        }
      {
        \str_if_eq_p:VV
          \l_tmpa_tl
          \c_@@_option_type_clist_tl
      }
        {
          \@@_get_option_value:nN
            { #1 }
            \l_tmpa_tl
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { #1~=~\c_left_brace_str }
          \clist_map_inline:Vn
            \l_tmpa_tl
            {
              \tl_gput_right:Nx
                \g_@@_formatted_lua_options_tl
                { "##1" ,~ }
            }
          \tl_gput_right:Nx
            \g_@@_formatted_lua_options_tl
            { \c_right_brace_str ,~ }
        }
    }
    {
      \@@_get_option_value:nN
        { #1 }
        \l_tmpa_tl
      \tl_gput_right:Nx
        \g_@@_formatted_lua_options_tl
        { #1~=~ " \l_tmpa_tl " ,~ }
```

```
      }
  }
\cs_generate_variant:Nn
  \clist_map_inline:nn
  { Vn }
\let\markdownPrepareLuaOptions=\@@_format_lua_options:
\def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
\ExplSyntaxOff
```

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.

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

First, ensure that the `cacheDir` directory exists.

```
  local lfs = require("lfs")
  local cacheDir = "\markdownOptionCacheDir"
  if not lfs.isdir(cacheDir) then
    assert(lfs.mkdir(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(\markdownLuaOptions)
}%
```

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.

```
  lfs.rmdir(cacheDir)
}%
```

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 2ε `\filecontents` macro to plain TeX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

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

```
    |markdownIfOption{frozenCache}{}{@
      |immediate|openout|markdownOutputFileStream@
        |markdownOptionInputTempFileName|relax@
      |markdownInfo{@
        Buffering block-level markdown input into the temporary @
        input file "|markdownOptionInputTempFileName" and scanning @
        for the closing token sequence "#1"}@
  }@
```

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

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

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

```
    |def|markdownReadAndConvertStripPercentSign##1{@
      |markdownIfOption{stripPercentSigns}{@
        |if##1%@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
        |else@
          |expandafter|expandafter|expandafter@
```

```
            |markdownReadAndConvertProcessLine@
            |expandafter|expandafter|expandafter##1@
        |fi@
      }{@
        |expandafter@
          |markdownReadAndConvertProcessLine@
          |expandafter##1@
      }@
    }@
```

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

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

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

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain TEX, `\input` the result of the conversion, and expand the ending control sequence.

```
        |def^^M{@
          |markdownInfo{The ending token sequence was found}@
          |markdownIfOption{frozenCache}{}{@
            |immediate|closeout|markdownOutputFileStream@
          }@
          |endgroup@
          |markdownInput{@
            |markdownOptionOutputDir@
            /|markdownOptionInputTempFileName@
          }@
          #2}@
      |fi@
```

Repeat with the next line.

```
      ^^M}@
```

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

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

```
    |def^^I{|markdownReadAndConvertTab}@
```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```
    |catcode`|^^M=13@
    |def^^M##1^^M{@
      |def^^M####1^^M{@
        |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
      ^^M}@
    ^^M}@
```

Reset the character categories back to the former state.

```
|endgroup
```

Use the lt3luabridge library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownLuaExecute
  #1
  {
    \int_compare:nNnT
      { \g_luabridge_method_int }
      =
      { \c_luabridge_method_shell_int }
      {
        \sys_if_shell_unrestricted:F
          {
            \sys_if_shell:TF
              {
                \msg_error:nn
                  { markdown }
                  { restricted-shell-access }
              }
              {
                \msg_error:nn
                  { markdown }
                  { disabled-shell-access }
              }
          }
      }
    \str_gset:NV
      \g_luabridge_output_dirname_str
      \markdownOptionOutputDir
    \luabridge_now:e
      { #1 }
  }
```

```
\cs_generate_variant:Nn
  \msg_new:nnnn
  { nnnV }
\tl_set:Nn
  \l_tmpa_tl
  {
    You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
    --enable-write18~flag,~or~write~shell_escape=t~in~the~
    texmf.cnf~file.
  }
\msg_new:nnnV
  { markdown }
  { restricted-shell-access }
  { Shell~escape~is~restricted }
  \l_tmpa_tl
\msg_new:nnnV
  { markdown }
  { disabled-shell-access }
  { Shell~escape~is~disabled }
  \l_tmpa_tl
\ExplSyntaxOff
```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```
\ExplSyntaxOn
\tl_new:N
  \g_@@_after_markinline_tl
\tl_gset:Nn
  \g_@@_after_markinline_tl
  { \unskip }
\cs_new:Npn
  \markinline
  {
```

Locally change the category of the special plain TEX characters to *other* in order to prevent unwanted interpretation of the input markdown text as TEX code.

```
    \group_begin:
    \cctab_select:N
      \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
    \@@_if_option:nF
      { frozenCache }
      {
        \immediate
```

```
        \openout
        \markdownOutputFileStream
        \markdownOptionInputTempFileName
        \relax
      \msg_info:nne
        { markdown }
        { buffering-markinline }
        { \markdownOptionInputTempFileName }
    }
```

Peek ahead and extract the inline markdown text.

```
    \peek_regex_replace_once:nnF
      { { (.*?) } }
      {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
        \c { @@_if_option:nF }
          \cB { frozenCache \cE }
          \cB {
            \c { immediate }
              \c { write }
              \c { markdownOutputFileStream }
              \cB { \1 \cE }
            \c { immediate }
              \c { closeout }
              \c { markdownOutputFileStream }
          \cE }
```

Reset the category codes and `\input` the result of the conversion.

```
        \c { group_end: }
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { contentLevel = inline \cE }
        \c { markdownInput }
          \cB {
            \c { markdownOptionOutputDir } /
            \c { markdownOptionInputTempFileName }
          \cE }
        \c { group_end: }
        \c { tl_use:N }
          \c { g_@@_after_markinline_tl }
      }
      {
        \msg_error:nn
          { markdown }
          { markinline-peek-failure }
        \group_end:
```

```
        \tl_use:N
          \g_@@_after_markinline_tl
      }
  }
\msg_new:nnn
  { markdown }
  { buffering-markinline }
  { Buffering~inline~markdown~input~into~
    the~temporary~input~file~"#1". }
\msg_new:nnnn
  { markdown }
  { markinline-peek-failure }
  { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
  { The~macro~should~be~followed~by~inline~
    markdown~text~in~curly~braces }
\ExplSyntaxOff
```

3.2.7 Typesetting Markdown

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

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownInput
  #1
  {
    \@@_if_option:nTF
      { frozenCache }
      {
        \markdownInputRaw
          { #1 }
      }
      {
```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On LATEX, this also includes the directories specified in `\input@path`.

```
        \tl_set:Nx
          \l_tmpa_tl
          { #1 }
        \file_get_full_name:VNTF
          \l_tmpa_tl
          \l_tmpb_tl
          {
            \exp_args:NV
              \markdownInputRaw
```

```
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { markdown-file-does-not-exist }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { markdown-file-does-not-exist }
  {
    Markdown~file~#1~does~not~exist
  }
\ExplSyntaxOff
\begingroup
```

Swap the category code of the backslash symbol and the pipe symbol, so that we may use the backslash symbol freely inside the Lua code. Furthermore, use the ampersand symbol to specify parameters.

```
  \catcode`|=0%
  \catcode`\\=12%
  \catcode`|&=6%
  |gdef|markdownInputRaw#1{%
```

Change the category code of the percent sign (%) to other, so that a user of the hybrid Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |begingroup
    |catcode`|%=12
```

Furthermore, also change the category code of the hash sign (#) to other, so that it's safe to tokenize the plain TeX output without mistaking hash signs with TeX's parameter numbers.

```
    |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding \markdownFrozenCache⟨*number*⟩ macro, and increment frozenCacheCounter.

```
    |markdownIfOption{frozenCache}{%
      |ifnum|markdownOptionFrozenCacheCounter=0|relax
        |markdownInfo{Reading frozen cache from
          "|markdownOptionFrozenCacheFileName"}%
        |input|markdownOptionFrozenCacheFileName|relax
      |fi
      |markdownInfo{Including markdown document number
        "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
```

```
      |csname markdownFrozenCache%
        |the|markdownOptionFrozenCacheCounter|endcsname
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }{%
      |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as LaTeXMk to track changes to the markdown document.

```
      |openin|markdownInputFileStream&1
      |closein|markdownInputFileStream
      |markdownPrepareLuaOptions
      |markdownLuaExecute{%
        |markdownPrepare
        local file = assert(io.open("&1", "r"),
          [[Could not open file "&1" for reading]])
        local input = assert(file:read("*a"))
        assert(file:close())
        print(convert(input))
        |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 [12, Section 9]. As a consequence, we can directly reuse the existing plain TeX implementation.

```
\def\markdownVersionSpace{ }%
```

```
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain TEX implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the LATEX interface (see Section 2.3.2).

```
\ExplSyntaxOn
\cs_gset_eq:NN
  \markinlinePlainTeX
  \markinline
\cs_gset:Npn
  \markinline
  {
    \peek_regex_replace_once:nn
      { ( \[ (.*?) \] ) ? }
      {
```

Apply the options locally.

```
        \c { group_begin: }
        \c { @@_setup:n }
          \cB { \2 \cE }
        \c { tl_put_right:Nn }
          \c { g_@@_after_markinline_tl }
          \cB { \c { group_end: } \cE }
        \c { markinlinePlainTeX }
      }
  }
\ExplSyntaxOff
```

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

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

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

```
\ExplSyntaxOn
\renewenvironment
```

```
13229   { markdown }
13230   {
```

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.

```
13231       \group_begin:
13232       \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
13233       \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
13234       \peek_regex_replace_once:nnF
13235         { \ *\[\r*([^]]*)\][^\r]* }
13236         {
```

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.

```
13237           \c { group_end: }
13238           \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the \markdownSetup macro.

```
        \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the \markdownReadAndConvert macro process the rest of the LaTeX environment.

We also make provision for using the \markdown command as a part of a different LaTeX environment as follows:

```
\newenvironment{foo}%
               {code before \markdown[some, options]}%
               {\markdownEnd code after}
```

```
        \c { exp_args:NV }
          \c { markdownReadAndConvert@ }
          \c { @currenvir }
      }
      {
        \group_end:
        \exp_args:NV
          \markdownReadAndConvert@
          \@currenvir
      }
  }
  { \markdownEnd }
\renewenvironment
  { markdown* }
  [ 1 ]
  {
    \msg_warning:nnn
      { markdown }
      { latex-markdown-star-deprecated }
      { #1 }
    \@@_setup:n
      { #1 }
    \markdownReadAndConvert@
      { markdown* }
  }
  { \markdownEnd }
\msg_new:nnn
  { markdown }
  { latex-markdown-star-deprecated }
  {
    The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
    be~removed~in~the~next~major~version~of~the~Markdown~package.
  }
\cs_generate_variant:Nn
```

```
  \@@_setup:n
  { V }
\ExplSyntaxOff
\begingroup
```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left ({) and right brace (}) with the less-than (<) and greater-than (>) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```
  \catcode`\|=0\catcode`\<=1\catcode`\>=2%
  \catcode`\\=12|catcode`|{=12|catcode`|}=12%
  |gdef|markdownReadAndConvert@#1<%
    |markdownReadAndConvert<\end{#1}>%
                           <|end<#1>>>%
|endgroup
```

3.3.2 Options

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

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

3.3.3 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
\cs_gset:Nn
  \@@_load_theme:nn
  {
```

If the Markdown package has already been loaded, determine whether 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 does, end with an error, since we cannot load LATEX themes after the preamble. Otherwise, try loading a plain TEX theme instead.

```
        \file_if_exist:nTF
          { markdown theme #2.sty }
          {
            \msg_error:nnn
              { markdown }
```

```
              { latex-theme-after-preamble }
              { #1 }
          }
          {
            \@@_plain_tex_load_theme:nn
              { #1 }
              { #2 }
          }
      \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.

```
        \file_if_exist:nTF
          { markdown theme #2.sty }
          {
            \msg_info:nnn
              { markdown }
              { loading-latex-theme }
              { #1 }
            \RequirePackage
              { markdown theme #2 }
          }
          {
            \@@_plain_tex_load_theme:nn
              { #1 }
              { #2 }
          }
      \fi
    \else
```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```
      \msg_info:nnn
        { markdown }
        { theme-loading-postponed }
        { #1 }
      \AtEndOfPackage
        {
          \@@_load_theme:nn
            { #1 }
            { #2 }
        }
    \fi
  }
\msg_new:nnn
  { markdown }
  { theme-loading-postponed }
  {
```

```
    Postponing~loading~Markdown~theme~#1~until~
    Markdown~package~has~finished~loading
  }
\msg_new:nnn
  { markdown }
  { loading-latex-theme }
  { Loading~LaTeX~Markdown~theme~#1 }
\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
\ExplSyntaxOff
```

The witiko/dot theme enables the fencedCode Lua option:

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

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

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

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

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

```
\renewcommand\markdownRendererInputFencedCodePrototype[3]{%
  \def\next##1 ##2\relax{%
    \ifthenelse{\equal{##1}{dot}}{%
      \markdownIfOption{frozenCache}{}{%
        \immediate\write18{%
          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:

```
      \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%
```

If the infostring does not start with `dot …`, we use the previous definition of the fenced code token renderer prototype:

```
    }{%
      \markdown@witiko@dot@oldRendererInputFencedCodePrototype
        {#1}{#2}{#3}%
    }%
  }%
  \next#2 \relax}%
```

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

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

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

```
\RequirePackage{catchfile,grffile}
```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```
\newcount\markdown@witiko@graphicx@http@counter
\markdown@witiko@graphicx@http@counter=0
\newcommand\markdown@witiko@graphicx@http@filename{%
  \markdownOptionCacheDir/witiko_graphicx_http%
  .\the\markdown@witiko@graphicx@http@counter}%
```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to downloads the online image to the pathname.

```
\newcommand\markdown@witiko@graphicx@http@download[2]{%
  wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}
```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
\begingroup
\catcode`\%=12
\catcode`\^^A=14
```

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

```
\global\def\markdownRendererImagePrototype#1#2#3#4{^^A
  \begingroup
    \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TEX option is disabled:

```
    \markdownIfOption{frozenCache}{}{^^A
      \immediate\write18{^^A
        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 "$OUTPUT" ];
          then
            \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
            printf '%s' "$OUTPUT" > "\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' > "\filename";
        fi}}^^A
```

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

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

The `witiko/markdown/defaults` LaTeX theme provides default definitions for token renderer prototypes. First, the LaTeX theme loads the plain TeX theme with the default definitions for plain TeX:

```
\markdownLoadPlainTeXTheme
```

Next, the LaTeX theme overrides some of the plain TeX definitions. See Section 3.3.4 for the actual definitions.

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

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

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the paralist package.

```
\@ifclassloaded{beamer}{}{%
  \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
  \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
}
```

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

```
\ExplSyntaxOn
\@ifpackageloaded{paralist}{
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_number_style_tl
  \tl_new:N
    \l_@@_latex_fancy_list_item_label_delimiter_style_tl
  \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_new:Nn
  \@@_latex_paralist_style:nn
  {
    \str_case:nn
      { #1 }
      {
        { Decimal } { 1 }
        { LowerRoman } { i }
        { UpperRoman } { I }
        { LowerAlpha } { a }
        { UpperAlpha } { A }
      }
    \@@_latex_fancy_list_item_label_delimiter:n
      { #2 }
  }
\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 }
      \@@_if_option:nTF
        { startNumber }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{enumerate} }
        }
        {
          \tl_set:Nn
            \l_tmpa_tl
            { \begin{enumerate}[ }
          \tl_put_right:Nx
            \l_tmpa_tl
            { \@@_latex_paralist_style:nn { #1 } { #2 } }
          \tl_put_right:Nn
            \l_tmpa_tl
            { ] }
        }
      \tl_use:N
        \l_tmpa_tl
    },
    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 }
    \@@_if_option:nTF
      { startNumber }
      {
        \tl_set:Nn
          \l_tmpa_tl
          { \begin{compactenum} }
      }
      {
        \tl_set:Nn
          \l_tmpa_tl
          { \begin{compactenum}[ }
        \tl_put_right:Nx
          \l_tmpa_tl
          { \@@_latex_paralist_style:nn { #1 } { #2 } }
        \tl_put_right:Nn
          \l_tmpa_tl
          { ] }
      }
    \tl_put_left:Nn
      \l_tmpa_tl
      {
        \plpartopsep=\partopsep
        \pltopsep=\topsep
      }
    \tl_use:N
      \l_tmpa_tl
  },
  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:
    }}}
  \cs_generate_variant:Nn
    \@@_latex_fancy_list_item_label:nnn
    { VVn }
}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\markdownRendererUlBegin},
    ulEndTight = {\markdownRendererUlEnd},
    fancyOlBegin = {\markdownRendererOlBegin},
    fancyOlEnd = {\markdownRendererOlEnd},
    olBeginTight = {\markdownRendererOlBegin},
    olEndTight = {\markdownRendererOlEnd},
    fancyOlBeginTight = {\markdownRendererOlBegin},
    fancyOlEndTight = {\markdownRendererOlEnd},
    dlBeginTight = {\markdownRendererDlBegin},
    dlEndTight = {\markdownRendererDlEnd}}}
}
\ExplSyntaxOff
\RequirePackage{amsmath}
```

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

```
\@ifpackageloaded{unicode-math}{
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\mdlgwhtsquare$},
  }}
}{
  \RequirePackage{amssymb}
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\square$},
  }}
}
\RequirePackage{csvsimple}
\RequirePackage{fancyvrb}
\RequirePackage{graphicx}
\markdownSetup{rendererPrototypes={
```

```
  hardLineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
```

We can capitalize on the fact that the expansion of renderers is performed by TEX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,[34] we can reliably detect math mode inside the renderer.

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
  }}}
\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}
            }
          { tex } { \markdownEscape{#3} }
        }
```

[34] This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```
        { \markdownInput{#3} }
    },
  },
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  image = {%
    \begin{figure}%
      \begin{center}%
        \includegraphics[alt={#1}]{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
    \end{figure}},
  ulBegin = {\begin{itemize}},
  ulEnd = {\end{itemize}},
  olBegin = {\begin{enumerate}},
  olItem = {\item{}},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  dlBegin = {\begin{description}},
  dlItem = {\item[#1]},
  dlEnd = {\end{description}},
  emphasis = {\emph{#1}},
  tickedBox = {$\boxtimes$},
  halfTickedBox = {$\boxdot$}}}
```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```
\ExplSyntaxOn
\seq_new:N
  \l_@@_header_identifiers_seq
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin = {
        \markdownSetup
          {
            rendererPrototypes = {
              attributeIdentifier = {
                \seq_put_right:Nn
                  \l_@@_header_identifiers_seq
                  { ##1 }
              },
            },
          }
      },
      headerAttributeContextEnd = {
```

```
        \seq_map_inline:Nn
          \l_@@_header_identifiers_seq
          { \label { ##1 } }
        \seq_clear:N
          \l_@@_header_identifiers_seq
      },
    },
  }
```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```
\bool_new:N
  \l_@@_header_unnumbered_bool
\markdownSetup
  {
    rendererPrototypes = {
      headerAttributeContextBegin += {
        \markdownSetup
          {
            rendererPrototypes = {
              attributeClassName = {
                \bool_if:nT
                  {
                    \str_if_eq_p:nn
                      { ##1 }
                      { unnumbered } &&
                    ! \l_@@_header_unnumbered_bool
                  }
                  {
                    \group_begin:
                    \bool_set_true:N
                      \l_@@_header_unnumbered_bool
                    \c@secnumdepth = 0
                    \markdownSetup
                      {
                        rendererPrototypes = {
                          sectionBegin = {
                            \group_begin:
                          },
                          sectionEnd = {
                            \group_end:
                          },
                        },
                      }
                  }
              },
            },
```

```
          }
      },
    },
  }
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  superscript = {\textsuperscript{#1}},
  subscript = {\textsubscript{#1}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  note = {\footnote{#1}}}}
```

3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\RequirePackage{ltxcmds}
\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.

```
      {
        \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`\#=6\relax
            \exp_args:NV
              \inputminted
              \l_tmpa_tl
              { #1 }
            \catcode`\#=12\relax
          }
```

```
13803           {
```

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

```
13804             \ltx@ifpackageloaded
13805               { listings }
13806               { \lstinputlisting[language=\l_tmpa_tl]{#1} }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
13807               { \markdownRendererInputFencedCode{#1}{}{} }
13808           }
13809       }
13810   }
13811 \ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
13812 \ExplSyntaxOn
13813 \def\markdownLATEXStrongEmphasis#1{%
13814   \str_if_in:NnTF
13815     \f@series
13816     { b }
13817     { \textnormal{#1} }
13818     { \textbf{#1} }
13819 }
13820 \ExplSyntaxOff
13821 \markdownSetup{rendererPrototypes={strongEmphasis={%
13822   \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
13823 \@ifundefined{chapter}{%
13824   \markdownSetup{rendererPrototypes = {
13825     headingOne = {\section{#1}},
13826     headingTwo = {\subsection{#1}},
13827     headingThree = {\subsubsection{#1}},
13828     headingFour = {\paragraph{#1}},
13829     headingFive = {\subparagraph{#1}}}}
13830 }{%
13831   \markdownSetup{rendererPrototypes = {
13832     headingOne = {\chapter{#1}},
13833     headingTwo = {\section{#1}},
13834     headingThree = {\subsection{#1}},
13835     headingFour = {\subsubsection{#1}},
13836     headingFive = {\paragraph{#1}},
13837     headingSix = {\subparagraph{#1}}}}
13838 }%
```

3.3.4.2 Tickboxes

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

```
\markdownSetup{
  rendererPrototypes = {
    ulItem = {%
      \futurelet\markdownLaTeXCheckbox\markdownLaTeXUlItem
    },
  },
}
\def\markdownLaTeXUlItem{%
  \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
    \item[\markdownLaTeXCheckbox]%
    \expandafter\@gobble
  \else
    \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
      \item[\markdownLaTeXCheckbox]%
      \expandafter\expandafter\expandafter\@gobble
    \else
      \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
        \item[\markdownLaTeXCheckbox]%
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\@gobble
      \else
        \item{}%
      \fi
    \fi
  \fi
}
```

3.3.4.3 HTML elements

If the `html` option is enabled and we are using TEX4ht[35], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
```

[35] See https://tug.org/tex4ht/.

```
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

3.3.4.4 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
\RequirePackage{gobble}
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
        \relax
        \cite{#1#2#6}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobblethree
      \fi
    \else%  Before a postnote (#5), dump the accumulator
      \ifx\relax#1\relax\else
        \cite{#1}%
      \fi
      \cite[#5]{#6}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
      \fi
      \expandafter\expandafter\expandafter
```

```
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobblethree
    \fi
  \else%  Before a prenote (#4), dump the accumulator
    \ifx\relax#1\relax\else
      \cite{#1}%
    \fi
    \ifnum\markdownLaTeXCitationsCounter>1\relax
      \space  % Insert a space before the prenote in later citations
    \fi
    #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
    \relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXBasicCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobblethree
  \fi\markdownLaTeXBasicCitations{#1#2#6},}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% Natbib implementation
\def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
        \citep{#1,#5}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  Before a postnote (#4), dump the accumulator
```

```
      \ifx\relax#1\relax\else
        \citep{#1}%
      \fi
      \citep[][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  Before a prenote (#3), dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citep{#1}%
    \fi
    \citep[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
    \relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibCitations{#1,#5}}
\def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
      \relax
        \citet{#1,#5}%  No prenotes/postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
```

```
      \fi
    \else%  After a prenote or a postnote, dump the accumulator
      \ifx\relax#1\relax\else
        \citet{#1}%
      \fi
      , \citet[#3][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
      \relax
        ,
      \else
        \ifnum
        \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
        \relax
          ,
        \fi
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibTextCitations
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  After a prenote or a postnote, dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citet{#1}%
    \fi
    , \citet[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
    \relax
      ,
    \else
      \ifnum
      \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
      \relax
        ,
      \fi
    \fi
    \expandafter\expandafter\expandafter
    \markdownLaTeXNatbibTextCitations
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
```

```
  \fi\markdownLaTeXNatbibTextCitations{#1,#5}}

% BibLaTeX implementation
\def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
  \relax
    \autocites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
\def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
  \relax
    \textcites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}

\markdownSetup{rendererPrototypes = {
  cite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    }},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
```

```
        \markdownLaTeXBasicTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    }}}}
```

3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```
\RequirePackage{url}
\RequirePackage{expl3}
\ExplSyntaxOn
\def\markdownRendererLinkPrototype#1#2#3#4{
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_set:Nn \l_tmpb_tl { #2 }
  \bool_set:Nn
    \l_tmpa_bool
    {
      \tl_if_eq_p:NN
        \l_tmpa_tl
        \l_tmpb_tl
    }
  \tl_set:Nn \l_tmpa_tl { #4 }
  \bool_set:Nn
    \l_tmpb_bool
    {
      \tl_if_empty_p:N
        \l_tmpa_tl
    }
```

If the label and the fully-escaped URI are equivalent and the title is empty, assume that the link is an autolink. Otherwise, assume that the link is either direct or indirect.

```
  \bool_if:nTF
    {
      \l_tmpa_bool && \l_tmpb_bool
    }
```

```
    {
      \markdownLaTeXRendererAutolink { #2 } { #3 }
    }{
      \markdownLaTeXRendererDirectOrIndirectLink
        { #1 } { #2 } { #3 } { #4 }
    }
}
\def\markdownLaTeXRendererAutolink#1#2{%
```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```
  \tl_set:Nn
    \l_tmpa_tl
    { #2 }
  \tl_trim_spaces:N
    \l_tmpa_tl
  \tl_set:Nx
    \l_tmpb_tl
    {
      \tl_range:Nnn
        \l_tmpa_tl
        { 1 }
        { 1 }
    }
  \str_if_eq:NNTF
    \l_tmpb_tl
    \c_hash_str
    {
      \tl_set:Nx
        \l_tmpb_tl
        {
          \tl_range:Nnn
            \l_tmpa_tl
            { 2 }
            { -1 }
        }
      \exp_args:NV
        \ref
        \l_tmpb_tl
    }{
      \url { #2 }
    }
}
\ExplSyntaxOff
\def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
  #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

3.3.4.6 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}
        \end{table}}%
    \fi
    \addto@hook\markdownLaTeXTable{\begin{tabular}}%
    \markdownLaTeXRowCounter=0%
    \markdownLaTeXRowTotal=#2%
    \markdownLaTeXColumnTotal=#3%
    \markdownLaTeXRenderTableRow
  }
}}
\def\markdownLaTeXRenderTableRow#1{%
  \markdownLaTeXColumnCounter=0%
  \ifnum\markdownLaTeXRowCounter=0\relax
    \markdownLaTeXReadAlignments#1%
```

```
    \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
        \the\markdownLaTeXTableAlignment}}%
    \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
  \else
    \markdownLaTeXRenderTableCell#1%
  \fi
  \ifnum\markdownLaTeXRowCounter=1\relax
    \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
  \fi
  \advance\markdownLaTeXRowCounter by 1\relax
  \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
    \the\markdownLaTeXTable
    \the\markdownLaTeXTableEnd
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableRow}
\def\markdownLaTeXReadAlignments#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \if#1d%
    \addto@hook\markdownLaTeXTableAlignment{l}%
  \else
    \addto@hook\markdownLaTeXTableAlignment{#1}%
  \fi
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
    \expandafter\@gobble
  \fi\markdownLaTeXReadAlignments}
\def\markdownLaTeXRenderTableCell#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
    \addto@hook\markdownLaTeXTable{#1&}%
  \else
    \addto@hook\markdownLaTeXTable{#1\\}%
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableCell}
```

3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

\markdownIfOption{lineBlocks}{%
  \RequirePackage{verse}
  \markdownSetup{rendererPrototypes={
    lineBlockBegin = {%
      \begingroup
        \def\markdownRendererHardLineBreak{\\}%
        \begin{verse}%
```

```
    },
    lineBlockEnd = {%
        \end{verse}%
      \endgroup
    },
  }}
}{}

```

3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  {
    author  .code:n = { \author{#1} },
    date    .code:n = { \date{#1}   },
    title   .code:n = { \title{#1}  },
  }
```

To complement the default setup of our key–values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```
\markdownSetup{
  rendererPrototypes = {
    jekyllDataEnd = {
      \AddToHook{begindocument/end}{\maketitle}
    },
  },
}
\ExplSyntaxOff
```

3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the soulutf8 package and use it to implement strike-throughs.

```
\markdownIfOption{strikeThrough}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
    rendererPrototypes = {
      strikeThrough = {%
        \st{#1}%
      },
```

```
    }
  }
}{}
```

3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the soulutf8 package and use it to implement marked text.

```
\markdownIfOption{mark}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
    rendererPrototypes = {
      mark = {%
        \hl{#1}%
      },
    }
  }
}{}
```

3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the graphicx package. Furthermore, in image attribute contexts, we will make attributes in the form ⟨*key*⟩=⟨*value*⟩ set the corresponding keys of the graphicx package to the corresponding values.

```
\ExplSyntaxOn
\@@_if_option:nT
  { linkAttributes }
  {
    \RequirePackage{graphicx}
    \markdownSetup{
      rendererPrototypes = {
        imageAttributeContextBegin = {
          \group_begin:
          \markdownSetup{
            rendererPrototypes = {
              attributeKeyValue = {
                \setkeys
                  { Gin }
                  { { ##1 } = { ##2 } }
              },
            },
          }
        },
        imageAttributeContextEnd = {
          \group_end:
        },
```

```
      },
    }
  }
\ExplSyntaxOff
```

3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block:nn
          { #1 }
          { #2 }
      }
```

```
  }
\ExplSyntaxOff
\fi % Closes `\markdownIfOption{plain}{\iffalse}{\iftrue}`
```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```
\newcommand\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat}%
```

3.4 ConTEXt Implementation

The ConTEXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTEXt formats *seem* to implement (the documentation is scarce) the majority of the plain TEX format required by the plain TEX implementation. As a consequence, we can directly reuse the existing plain TEX implementation after supplying the missing plain TEX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents LATEX package.

```
\def\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTEXt.

```
  \catcode`|=12}%
```

3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is 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}%
```

The `\startmarkdown` and `\stopmarkdown` 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. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because "these spaces are hard to see in an editor". At the moment, there is no option to suppress this behavior in (Lua)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}%
|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
\cs_gset:Nn
```

```
  \@@_load_theme:nn
  {
```

Determine whether a file named `t-markdowntheme`⟨*munged theme name*⟩`.tex` exists. If it does, load it. Otherwise, try loading a plain TEX theme instead.

```
    \file_if_exist:nTF
      { t - markdown theme #2.tex }
      {
        \msg_info:nnn
          { markdown }
          { loading-context-theme }
          { #1 }
        \usemodule
          [ t ]
          [ markdown theme #2 ]
      }
      {
        \@@_plain_tex_load_theme:nn
          { #1 }
          { #2 }
      }
  }
\msg_new:nnn
  { markdown }
  { loading-context-theme }
  { Loading~ConTeXt~Markdown~theme~#1 }
\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).

```
\markdownIfOption{noDefaults}{}{
  \setupmarkdown[theme=witiko/markdown/defaults]
}
\stopmodule
\protect
```

References

[1] LuaTEX development team. *LuaTEX reference manual*. Version 1.10 (stable). July 23, 2021. URL: https://www.pragma-ade.com/general/manuals/luatex.pdf (visited on 09/30/2022).

[2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[5] Bonita Sharif and Jonathan I. Maletic. "An Eye Tracking Study on camelCase and under_score Identifier Styles." In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: 10.1109/ICPC.2010.41.

[6] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[7] Frank Mittelbach. *The `doc` and `shortvrb` Packages*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/doc.pdf (visited on 02/19/2018).

[8] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf (visited on 02/11/2021).

[9] Vít 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).

[10] 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).

[11] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[12] Johannes Braams et al. *The LaTeX 2ε Sources*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/source2e.pdf (visited on 01/08/2018).

[13] Donald Ervin Knuth. *TEX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

[14] 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