A Markdown Interpreter for TEX

Vít Starý Novotný
witiko@mail.muni.cz

Version 3.5.0-0-gfd01a252
2024-04-29

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

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the 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",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2023 Vít Starý Novotný"},
    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");
```

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

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

```
(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.

```
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.

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.

```
⟨/tex⟩
⟨*context⟩
\unprotect
⟨/context⟩
⟨*context, tex⟩
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic
\fi
⟨/context, tex⟩
⟨*tex⟩
```

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

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

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

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

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

Unless you convert markdown documents to TEX manually using the Lua command-line interface (see Section 2.1.7), the plain TEX part of the package will require that either the LuaTEX `\directlua` primitive or the shell access file stream 18 is available in your TEX engine. If only the shell access file stream is available in your TEX engine (as is the case with pdfTEX and XeTEX), then unless your TEX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 LaTeX Requirements

The LATEX part of the package requires that the LATEX 2$_\varepsilon$ format is loaded,

```
34 \NeedsTeXFormat{LaTeX2e}%
```

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.

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

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

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).

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

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

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

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

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

graphicx A package that builds upon the graphics package, which is part of the LaTeX 2_ε kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

grffile A package that extends the name processing of the graphics package to support a larger range of file names in 2006 $\leqslant$ TeX Live $\leqslant$ 2019. Since TeX Live $\geqslant$ 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.

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.

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

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.

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

```
35 \RequirePackage{expl3}
```

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.

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

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

structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

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

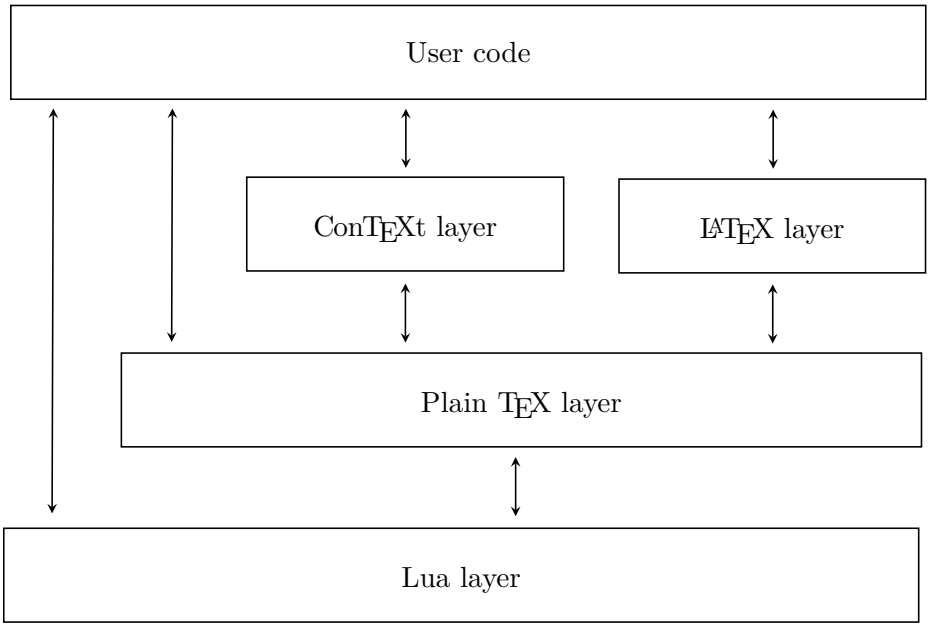

Figure 1: A block diagram of the Markdown package

2.1 Lua Interface

The Lua interface provides the conversion from 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.

```
36 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain TEX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain TEX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The

`options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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

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

2.1.2 User-Defined Syntax Extensions

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

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

```
37 local walkable_syntax = {
38   Block = {
39     "Blockquote",
40     "Verbatim",
41     "ThematicBreak",
42     "BulletList",
43     "OrderedList",
44     "DisplayHtml",
45     "Heading",
46   },
47   BlockOrParagraph = {
48     "Block",
49     "Paragraph",
50     "Plain",
51   },
52   Inline = {
53     "Str",
54     "Space",
55     "Endline",
56     "EndlineBreak",
57     "LinkAndEmph",
58     "Code",
59     "AutoLinkUrl",
60     "AutoLinkEmail",
61     "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: `false`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also

produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

`false` Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

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

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

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

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

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

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

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

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[7]:

```
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.

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

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

`blankBeforeCodeFence`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following fenced code block.

`false` Do not require a blank line between a paragraph and the following fenced code block.

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

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

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

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

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

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

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

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

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

`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[8]:

```
[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
```

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

`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
```

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

`true` Enable the Pandoc citation syntax extension[9]:

```
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
```

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

`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
```

`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
```

`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[10]:

```
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
```

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

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

`true` Enable the Pandoc fenced code attribute syntax extension[11]:

```
~~~~ {#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[12]:

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

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

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

# 123 Sesame Street
```

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

[13] 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 interpretted 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[14]:

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

[14]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[15]:

```
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[16] 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.
---
```

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

```
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[18]:

```
| 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
```

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

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

`true` Enable the Pandoc mark syntax extension[19]:

```
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[20]:

```
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
```

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

```
`$H_2 O$`{=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 \\
```

[21] 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[22] 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
```

[22] 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[23]:

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

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

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

[23] 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[24]:

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

`false` Disable the Pandoc subscript syntax extension.

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

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

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

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

```
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
```

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

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

true

: Enable the Pandoc table caption syntax extension[26] 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[27]:

```
- [ ] 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
```

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

```
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
```

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

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

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

```
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[30]:

```
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:

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

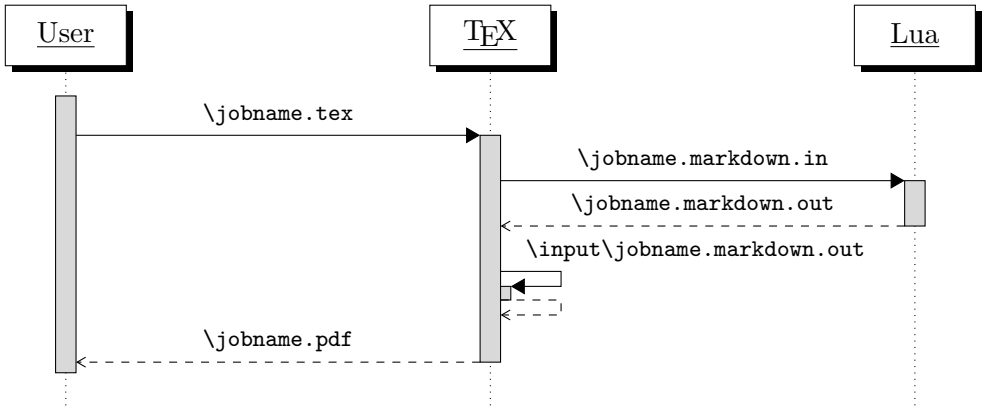

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
665
666 local HELP_STRING = [[
667 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
668 where OPTIONS are documented in the Lua interface section of the
669 technical Markdown package documentation.
670
671 When OUTPUT_FILE is unspecified, the result of the conversion will be
672 written to the standard output. When INPUT_FILE is also unspecified, the
673 result of the conversion will be read from the standard input.
674
675 Report bugs to: witiko@mail.muni.cz
676 Markdown package home page: <https://github.com/witiko/markdown>]]
677
```

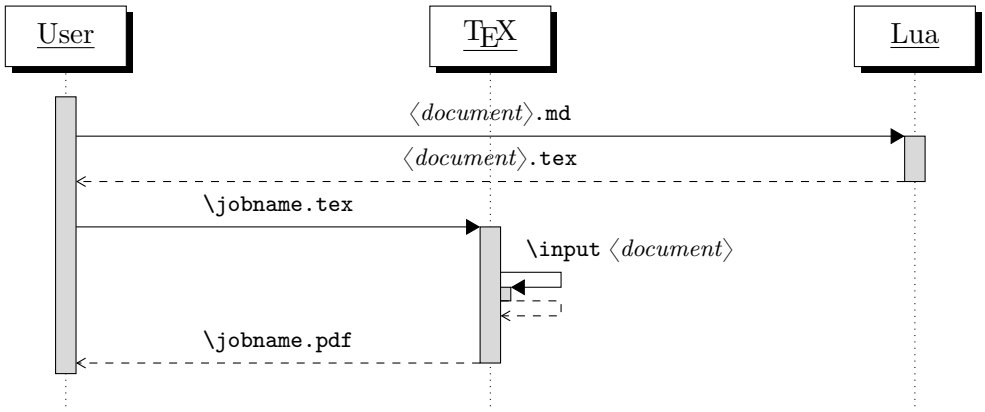

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
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)
    return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
```

```
  end)
  return cased_option_name
end

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

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

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

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

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

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

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

```
      local default_type = type(defaultOptions[key])
      if default_type == "boolean" then
        options[key] = (value == "true")
      elseif default_type == "number" then
        options[key] = tonumber(value)
      elseif default_type == "table" then
        options[key] = {}
        for item in value:gmatch("[^ ,]+") do
          table.insert(options[key], item)
```

```
        end
      else
        if default_type ~= "string" then
          if default_type == "nil" then
            warn('Option "' .. key .. '" not recognized.')
          else
            warn('Option "' .. key .. '" type not recognized, please file ' ..
                 'a report to the package maintainer.')
          end
          warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
               key .. '" as a string.')
        end
        options[key] = value
      end
      goto continue
```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

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

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

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

The first argument that matches none of the above patters 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 patters 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::
end
```

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

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

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

```
\input hello
```

2.2 Plain TEX Interface

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

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

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown

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.

```
\let\markdownBegin\relax
\let\markdownEnd\relax
```

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

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corrolary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of TEX [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.

```
775 \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.

```
\cs_generate_variant:Nn
  \@@_add_plain_tex_option:nnn
  { nnV }
```

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.

```
\ExplSyntaxOff
\input lt3luabridge.tex
\ExplSyntaxOn
\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_set:Nn
      \l_tmpa_tl
      { . }
  }
\@@_add_plain_tex_option:nnV
  { outputDir }
  { path }
  \l_tmpa_tl
```

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
  {
```

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
      }
  }
\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
      }
    \prop_gput:NVn
      \g_@@_snippets_prop
      \l_tmpa_tl
      { #2 }
  }
\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
          \@@_setup:V
            \l_tmpb_tl
        }
        {
          \msg_error:nnV
            { markdown }
            { undefined-snippet }
            \l_tmpa_tl
        }
    }
  }
\msg_new:nnn
  { markdown }
  { undefined-snippet }
  { Can't~invoke~undefined~snippet~#1 }
\cs_generate_variant:Nn
  \@@_setup:n
  { V }
```

```
1311 \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 contaning 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 contaning

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[31] 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
  { 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 }
```

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

```
  { 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 occurance 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` option is 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 `\markdownRendererFencedDiv` 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 `\markdownRendererHeaderAttri` 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 contaning 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 contaning 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 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 `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```
\def\markdownRendererJekyllDataString{%
  \markdownRendererJekyllDataStringPrototype}%
\ExplSyntaxOn
\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
```

2.2.5.44 Generating Plain TEX Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain TEX macros for token renderers. Futhermore, 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_function:NN
      \g_@@_renderers_seq
      \@@_define_renderer:n
  }
\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 += {...},
        },
      }
    },
  },
}
```

```
      \regex_match:NVTF
        \c_@@_appending_key_regex
        \l_keys_key_str
        {
          \bool_gset_true:N
            \g_@@_appending_renderer_bool
          \tl_set:NV
            \l_tmpa_tl
            \l_keys_key_str
          \regex_replace_once:NnN
            \c_@@_appending_key_regex
            { }
            \l_tmpa_tl
          \tl_set:Nx
            \l_tmpb_tl
            { { \l_tmpa_tl } = }
          \tl_put_right:Nn
            \l_tmpb_tl
            { { #1 } }
          \keys_set:nV
            { markdown/options/renderers }
            \l_tmpb_tl
          \bool_gset_false:N
            \g_@@_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. Futhermore, 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_function:NN
      \g_@@_renderers_seq
      \@@_define_renderer_prototype:n
  }
\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 }
          { }
      }
  }
\cs_generate_variant:Nn
  \@@_define_renderer_prototype:nNn
  { 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.
  }
}
```

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

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

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

```
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeClassName += {...},
        },
      }
    },
    % Define the processing of a single specific HTML identifier.
    headerAttributeContextBegin += {
      \markdownSetup{
        rendererPrototypes = {
          attributeIdentifier += {...},
        },
      }
    },
  },
}
```

```
3094        \regex_match:NVTF
3095          \c_@@_appending_key_regex
3096          \l_keys_key_str
3097          {
3098            \bool_gset_true:N
3099              \g_@@_appending_renderer_prototype_bool
3100            \tl_set:NV
3101              \l_tmpa_tl
3102              \l_keys_key_str
3103            \regex_replace_once:NnN
3104              \c_@@_appending_key_regex
3105              { }
3106              \l_tmpa_tl
3107            \tl_set:Nx
3108              \l_tmpb_tl
3109              { { \l_tmpa_tl } = }
3110            \tl_put_right:Nn
3111              \l_tmpb_tl
3112              { { #1 } }
3113            \keys_set:nV
3114              { markdown/options/renderer-prototypes }
3115              \l_tmpb_tl
3116            \bool_gset_false:N
3117              \g_@@_appending_renderer_prototype_bool
3118          }
```

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.
}
```

```
3119          {
3120            \@@_glob_seq:VnN
3121              \l_keys_key_str
3122              { g_@@_renderers_seq }
3123              \l_@@_renderer_glob_results_seq
3124            \seq_if_empty:NTF
3125              \l_@@_renderer_glob_results_seq
3126              {
3127                \msg_error:nnV
3128                  { markdown }
3129                  { undefined-renderer-prototype }
3130                  \l_keys_key_str
3131              }
```

```
            {
              \tl_set:Nn
                \l_@@_renderer_glob_definition_tl
                { \exp_not:n { #1 } }
              \seq_map_inline:Nn
                \l_@@_renderer_glob_results_seq
                {
                  \tl_set:Nn
                    \l_tmpa_tl
                    { { ##1 } = }
                  \tl_put_right:Nx
                    \l_tmpa_tl
                    { { \l_@@_renderer_glob_definition_tl } }
                  \keys_set:nV
                    { markdown/options/renderer-prototypes }
                    \l_tmpa_tl
                }
            }
        }
    },
  }
\msg_new:nnn
  { markdown }
  { undefined-renderer-prototype }
  {
    Renderer~prototype~#1~is~undefined.
  }
\@@_with_various_cases:nn
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/options }
      {
        #1 .code:n = {
          \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 2.2.5.44) 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 2$_\varepsilon$ parses package options.

2.3.1 Typesetting Markdown

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

The `markdown` and `markdown*` LaTeX environments are used to typeset markdown document fragments. Both LaTeX environments accept LaTeX interface options (see ection 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.

```
3208 \newenvironment{markdown}\relax\relax
3209 \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}
```

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}
```

```
\newif\ifmarkdownLaTeXLoaded
  \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 mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

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

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

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

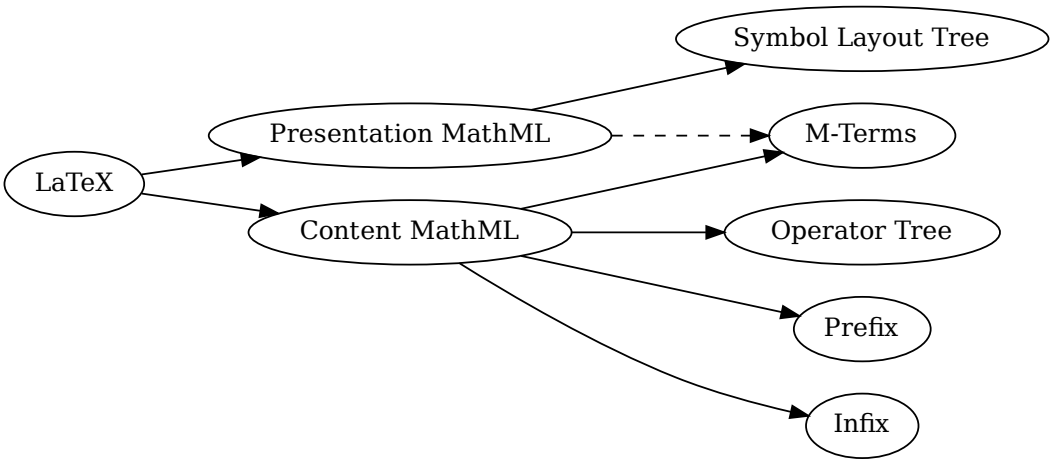

Figure 4: Various formats of mathematical formulae

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

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

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

```
\documentclass{article}
\usepackage[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

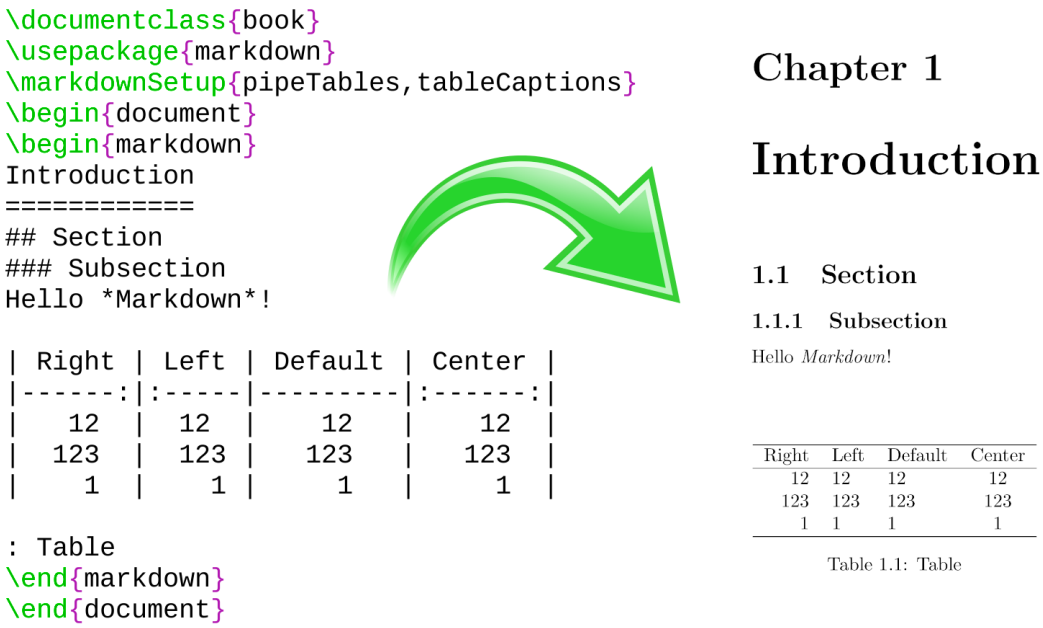

Figure 5: The banner of the Markdown package

theme requires the catchfile LaTeX package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `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
```

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 \inputting the file.

2.4.1 Typesetting Markdown

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

```
3251 \let\startmarkdown\relax
3252 \let\stopmarkdown\relax
3253 \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.

```
3284 \startmodule[markdownthemewitiko_markdown_defaults]
3285 \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 computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
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, then 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))
```

```
3308      assert(file:close())
3309    end
3310    return name
3311  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.

```
3312  function util.cache_verbatim(dir, string)
3313    local name = util.cache(dir, string, nil, nil, ".verbatim")
3314    return name
3315  end
```

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

```
3316  function util.table_copy(t)
3317    local u = { }
3318    for k, v in pairs(t) do u[k] = v end
3319    return setmetatable(u, getmetatable(t))
3320  end
```

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

```
3321  function util.encode_json_string(s)
3322    s = s:gsub([[\]], [[\\]])
3323    s = s:gsub([["]], [[\"]])
3324    return [["]] .. s .. [["]]
3325  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].

```
3326  function util.expand_tabs_in_line(s, tabstop)
3327    local tab = tabstop or 4
3328    local corr = 0
3329    return (s:gsub("()\t", function(p)
3330              local sp = tab - (p - 1 + corr) % tab
3331              corr = corr - 1 + sp
3332              return string.rep(" ", sp)
3333            end))
3334  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.

```
3335  function util.walk(t, f)
3336    local typ = type(t)
3337    if typ == "string" then
```

```
    f(t)
  elseif typ == "table" then
    local i = 1
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

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

```
function util.flatten(ary)
  local new = {}
  for _,v in ipairs(ary) do
    if type(v) == "table" then
      for _,w in ipairs(util.flatten(v)) do
        new[#new + 1] = w
      end
    else
      new[#new + 1] = v
    end
  end
  return new
end
```

The `util.rope_to_string` method converts a rope `rope` to a string and returns it. For the definition of a rope, see the definition of the `util.walk` method.

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

The `util.rope_last` method retrieves the last item in a rope. For the definition of a rope, see the definition of the `util.walk` method.

```
function util.rope_last(rope)
  if #rope == 0 then
```

```
3377      return nil
3378    else
3379      local l = rope[#rope]
3380      if type(l) == "table" then
3381        return util.rope_last(l)
3382      else
3383        return l
3384      end
3385    end
3386  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`.

```
3387  function util.intersperse(ary, x)
3388    local new = {}
3389    local l = #ary
3390    for i,v in ipairs(ary) do
3391      local n = #new
3392      new[n + 1] = v
3393      if i ~= l then
3394        new[n + 2] = x
3395      end
3396    end
3397    return new
3398  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`.

```
3399  function util.map(ary, f)
3400    local new = {}
3401    for i,v in ipairs(ary) do
3402      new[i] = f(v)
3403    end
3404    return new
3405  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 occurances of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

```
3406  function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3407    local char_escapes_list = ""
3408    for i,_ in pairs(char_escapes) do
3409      char_escapes_list = char_escapes_list .. i
```

```
3410    end
```

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

```
3411    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 occurance 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.

```
3412    if string_escapes then
3413      for k,v in pairs(string_escapes) do
3414        escapable = P(k) / v + escapable
3415      end
3416    end
```

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

```
3417    local escape_string = Cs((escapable + any)^0)
```

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

```
3418    return function(s)
3419      return lpeg.match(escape_string, s)
3420    end
3421  end
```

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

```
3422  function util.pathname(dir, file)
3423    if #dir == 0 then
3424      return file
3425    else
3426      return dir .. "/" .. file
3427    end
3428  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.

```
3429  local entities = {}
```

```
3430
3431 local character_entities = {
3432   ["Tab"] = 9,
3433   ["NewLine"] = 10,
3434   ["excl"] = 33,
3435   ["QUOT"] = 34,
3436   ["quot"] = 34,
3437   ["num"] = 35,
3438   ["dollar"] = 36,
3439   ["percnt"] = 37,
3440   ["AMP"] = 38,
3441   ["amp"] = 38,
3442   ["apos"] = 39,
3443   ["lpar"] = 40,
3444   ["rpar"] = 41,
3445   ["ast"] = 42,
3446   ["midast"] = 42,
3447   ["plus"] = 43,
3448   ["comma"] = 44,
3449   ["period"] = 46,
3450   ["sol"] = 47,
3451   ["colon"] = 58,
3452   ["semi"] = 59,
3453   ["LT"] = 60,
3454   ["lt"] = 60,
3455   ["nvlt"] = {60, 8402},
3456   ["bne"] = {61, 8421},
3457   ["equals"] = 61,
3458   ["GT"] = 62,
3459   ["gt"] = 62,
3460   ["nvgt"] = {62, 8402},
3461   ["quest"] = 63,
3462   ["commat"] = 64,
3463   ["lbrack"] = 91,
3464   ["lsqb"] = 91,
3465   ["bsol"] = 92,
3466   ["rbrack"] = 93,
3467   ["rsqb"] = 93,
3468   ["Hat"] = 94,
3469   ["UnderBar"] = 95,
3470   ["lowbar"] = 95,
3471   ["DiacriticalGrave"] = 96,
3472   ["grave"] = 96,
3473   ["fjlig"] = {102, 106},
3474   ["lbrace"] = 123,
3475   ["lcub"] = 123,
3476   ["VerticalLine"] = 124,
```

```
3477   ["verbar"] = 124,
3478   ["vert"] = 124,
3479   ["rbrace"] = 125,
3480   ["rcub"] = 125,
3481   ["NonBreakingSpace"] = 160,
3482   ["nbsp"] = 160,
3483   ["iexcl"] = 161,
3484   ["cent"] = 162,
3485   ["pound"] = 163,
3486   ["curren"] = 164,
3487   ["yen"] = 165,
3488   ["brvbar"] = 166,
3489   ["sect"] = 167,
3490   ["Dot"] = 168,
3491   ["DoubleDot"] = 168,
3492   ["die"] = 168,
3493   ["uml"] = 168,
3494   ["COPY"] = 169,
3495   ["copy"] = 169,
3496   ["ordf"] = 170,
3497   ["laquo"] = 171,
3498   ["not"] = 172,
3499   ["shy"] = 173,
3500   ["REG"] = 174,
3501   ["circledR"] = 174,
3502   ["reg"] = 174,
3503   ["macr"] = 175,
3504   ["strns"] = 175,
3505   ["deg"] = 176,
3506   ["PlusMinus"] = 177,
3507   ["plusmn"] = 177,
3508   ["pm"] = 177,
3509   ["sup2"] = 178,
3510   ["sup3"] = 179,
3511   ["DiacriticalAcute"] = 180,
3512   ["acute"] = 180,
3513   ["micro"] = 181,
3514   ["para"] = 182,
3515   ["CenterDot"] = 183,
3516   ["centerdot"] = 183,
3517   ["middot"] = 183,
3518   ["Cedilla"] = 184,
3519   ["cedil"] = 184,
3520   ["sup1"] = 185,
3521   ["ordm"] = 186,
3522   ["raquo"] = 187,
3523   ["frac14"] = 188,
```

```
3524    ["frac12"] = 189,
3525    ["half"] = 189,
3526    ["frac34"] = 190,
3527    ["iquest"] = 191,
3528    ["Agrave"] = 192,
3529    ["Aacute"] = 193,
3530    ["Acirc"] = 194,
3531    ["Atilde"] = 195,
3532    ["Auml"] = 196,
3533    ["Aring"] = 197,
3534    ["angst"] = 197,
3535    ["AElig"] = 198,
3536    ["Ccedil"] = 199,
3537    ["Egrave"] = 200,
3538    ["Eacute"] = 201,
3539    ["Ecirc"] = 202,
3540    ["Euml"] = 203,
3541    ["Igrave"] = 204,
3542    ["Iacute"] = 205,
3543    ["Icirc"] = 206,
3544    ["Iuml"] = 207,
3545    ["ETH"] = 208,
3546    ["Ntilde"] = 209,
3547    ["Ograve"] = 210,
3548    ["Oacute"] = 211,
3549    ["Ocirc"] = 212,
3550    ["Otilde"] = 213,
3551    ["Ouml"] = 214,
3552    ["times"] = 215,
3553    ["Oslash"] = 216,
3554    ["Ugrave"] = 217,
3555    ["Uacute"] = 218,
3556    ["Ucirc"] = 219,
3557    ["Uuml"] = 220,
3558    ["Yacute"] = 221,
3559    ["THORN"] = 222,
3560    ["szlig"] = 223,
3561    ["agrave"] = 224,
3562    ["aacute"] = 225,
3563    ["acirc"] = 226,
3564    ["atilde"] = 227,
3565    ["auml"] = 228,
3566    ["aring"] = 229,
3567    ["aelig"] = 230,
3568    ["ccedil"] = 231,
3569    ["egrave"] = 232,
3570    ["eacute"] = 233,
```

```
3571   ["ecirc"] = 234,
3572   ["euml"] = 235,
3573   ["igrave"] = 236,
3574   ["iacute"] = 237,
3575   ["icirc"] = 238,
3576   ["iuml"] = 239,
3577   ["eth"] = 240,
3578   ["ntilde"] = 241,
3579   ["ograve"] = 242,
3580   ["oacute"] = 243,
3581   ["ocirc"] = 244,
3582   ["otilde"] = 245,
3583   ["ouml"] = 246,
3584   ["div"] = 247,
3585   ["divide"] = 247,
3586   ["oslash"] = 248,
3587   ["ugrave"] = 249,
3588   ["uacute"] = 250,
3589   ["ucirc"] = 251,
3590   ["uuml"] = 252,
3591   ["yacute"] = 253,
3592   ["thorn"] = 254,
3593   ["yuml"] = 255,
3594   ["Amacr"] = 256,
3595   ["amacr"] = 257,
3596   ["Abreve"] = 258,
3597   ["abreve"] = 259,
3598   ["Aogon"] = 260,
3599   ["aogon"] = 261,
3600   ["Cacute"] = 262,
3601   ["cacute"] = 263,
3602   ["Ccirc"] = 264,
3603   ["ccirc"] = 265,
3604   ["Cdot"] = 266,
3605   ["cdot"] = 267,
3606   ["Ccaron"] = 268,
3607   ["ccaron"] = 269,
3608   ["Dcaron"] = 270,
3609   ["dcaron"] = 271,
3610   ["Dstrok"] = 272,
3611   ["dstrok"] = 273,
3612   ["Emacr"] = 274,
3613   ["emacr"] = 275,
3614   ["Edot"] = 278,
3615   ["edot"] = 279,
3616   ["Eogon"] = 280,
3617   ["eogon"] = 281,
```

```
3618   ["Ecaron"] = 282,
3619   ["ecaron"] = 283,
3620   ["Gcirc"] = 284,
3621   ["gcirc"] = 285,
3622   ["Gbreve"] = 286,
3623   ["gbreve"] = 287,
3624   ["Gdot"] = 288,
3625   ["gdot"] = 289,
3626   ["Gcedil"] = 290,
3627   ["Hcirc"] = 292,
3628   ["hcirc"] = 293,
3629   ["Hstrok"] = 294,
3630   ["hstrok"] = 295,
3631   ["Itilde"] = 296,
3632   ["itilde"] = 297,
3633   ["Imacr"] = 298,
3634   ["imacr"] = 299,
3635   ["Iogon"] = 302,
3636   ["iogon"] = 303,
3637   ["Idot"] = 304,
3638   ["imath"] = 305,
3639   ["inodot"] = 305,
3640   ["IJlig"] = 306,
3641   ["ijlig"] = 307,
3642   ["Jcirc"] = 308,
3643   ["jcirc"] = 309,
3644   ["Kcedil"] = 310,
3645   ["kcedil"] = 311,
3646   ["kgreen"] = 312,
3647   ["Lacute"] = 313,
3648   ["lacute"] = 314,
3649   ["Lcedil"] = 315,
3650   ["lcedil"] = 316,
3651   ["Lcaron"] = 317,
3652   ["lcaron"] = 318,
3653   ["Lmidot"] = 319,
3654   ["lmidot"] = 320,
3655   ["Lstrok"] = 321,
3656   ["lstrok"] = 322,
3657   ["Nacute"] = 323,
3658   ["nacute"] = 324,
3659   ["Ncedil"] = 325,
3660   ["ncedil"] = 326,
3661   ["Ncaron"] = 327,
3662   ["ncaron"] = 328,
3663   ["napos"] = 329,
3664   ["ENG"] = 330,
```

```
3665   ["eng"] = 331,
3666   ["Omacr"] = 332,
3667   ["omacr"] = 333,
3668   ["Odblac"] = 336,
3669   ["odblac"] = 337,
3670   ["OElig"] = 338,
3671   ["oelig"] = 339,
3672   ["Racute"] = 340,
3673   ["racute"] = 341,
3674   ["Rcedil"] = 342,
3675   ["rcedil"] = 343,
3676   ["Rcaron"] = 344,
3677   ["rcaron"] = 345,
3678   ["Sacute"] = 346,
3679   ["sacute"] = 347,
3680   ["Scirc"] = 348,
3681   ["scirc"] = 349,
3682   ["Scedil"] = 350,
3683   ["scedil"] = 351,
3684   ["Scaron"] = 352,
3685   ["scaron"] = 353,
3686   ["Tcedil"] = 354,
3687   ["tcedil"] = 355,
3688   ["Tcaron"] = 356,
3689   ["tcaron"] = 357,
3690   ["Tstrok"] = 358,
3691   ["tstrok"] = 359,
3692   ["Utilde"] = 360,
3693   ["utilde"] = 361,
3694   ["Umacr"] = 362,
3695   ["umacr"] = 363,
3696   ["Ubreve"] = 364,
3697   ["ubreve"] = 365,
3698   ["Uring"] = 366,
3699   ["uring"] = 367,
3700   ["Udblac"] = 368,
3701   ["udblac"] = 369,
3702   ["Uogon"] = 370,
3703   ["uogon"] = 371,
3704   ["Wcirc"] = 372,
3705   ["wcirc"] = 373,
3706   ["Ycirc"] = 374,
3707   ["ycirc"] = 375,
3708   ["Yuml"] = 376,
3709   ["Zacute"] = 377,
3710   ["zacute"] = 378,
3711   ["Zdot"] = 379,
```

```
3712   ["zdot"] = 380,
3713   ["Zcaron"] = 381,
3714   ["zcaron"] = 382,
3715   ["fnof"] = 402,
3716   ["imped"] = 437,
3717   ["gacute"] = 501,
3718   ["jmath"] = 567,
3719   ["circ"] = 710,
3720   ["Hacek"] = 711,
3721   ["caron"] = 711,
3722   ["Breve"] = 728,
3723   ["breve"] = 728,
3724   ["DiacriticalDot"] = 729,
3725   ["dot"] = 729,
3726   ["ring"] = 730,
3727   ["ogon"] = 731,
3728   ["DiacriticalTilde"] = 732,
3729   ["tilde"] = 732,
3730   ["DiacriticalDoubleAcute"] = 733,
3731   ["dblac"] = 733,
3732   ["DownBreve"] = 785,
3733   ["Alpha"] = 913,
3734   ["Beta"] = 914,
3735   ["Gamma"] = 915,
3736   ["Delta"] = 916,
3737   ["Epsilon"] = 917,
3738   ["Zeta"] = 918,
3739   ["Eta"] = 919,
3740   ["Theta"] = 920,
3741   ["Iota"] = 921,
3742   ["Kappa"] = 922,
3743   ["Lambda"] = 923,
3744   ["Mu"] = 924,
3745   ["Nu"] = 925,
3746   ["Xi"] = 926,
3747   ["Omicron"] = 927,
3748   ["Pi"] = 928,
3749   ["Rho"] = 929,
3750   ["Sigma"] = 931,
3751   ["Tau"] = 932,
3752   ["Upsilon"] = 933,
3753   ["Phi"] = 934,
3754   ["Chi"] = 935,
3755   ["Psi"] = 936,
3756   ["Omega"] = 937,
3757   ["ohm"] = 937,
3758   ["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,
```

```
3806   ["varepsilon"] = 1013,
3807   ["backepsilon"] = 1014,
3808   ["bepsi"] = 1014,
3809   ["IOcy"] = 1025,
3810   ["DJcy"] = 1026,
3811   ["GJcy"] = 1027,
3812   ["Jukcy"] = 1028,
3813   ["DScy"] = 1029,
3814   ["Iukcy"] = 1030,
3815   ["YIcy"] = 1031,
3816   ["Jsercy"] = 1032,
3817   ["LJcy"] = 1033,
3818   ["NJcy"] = 1034,
3819   ["TSHcy"] = 1035,
3820   ["KJcy"] = 1036,
3821   ["Ubrcy"] = 1038,
3822   ["DZcy"] = 1039,
3823   ["Acy"] = 1040,
3824   ["Bcy"] = 1041,
3825   ["Vcy"] = 1042,
3826   ["Gcy"] = 1043,
3827   ["Dcy"] = 1044,
3828   ["IEcy"] = 1045,
3829   ["ZHcy"] = 1046,
3830   ["Zcy"] = 1047,
3831   ["Icy"] = 1048,
3832   ["Jcy"] = 1049,
3833   ["Kcy"] = 1050,
3834   ["Lcy"] = 1051,
3835   ["Mcy"] = 1052,
3836   ["Ncy"] = 1053,
3837   ["Ocy"] = 1054,
3838   ["Pcy"] = 1055,
3839   ["Rcy"] = 1056,
3840   ["Scy"] = 1057,
3841   ["Tcy"] = 1058,
3842   ["Ucy"] = 1059,
3843   ["Fcy"] = 1060,
3844   ["KHcy"] = 1061,
3845   ["TScy"] = 1062,
3846   ["CHcy"] = 1063,
3847   ["SHcy"] = 1064,
3848   ["SHCHcy"] = 1065,
3849   ["HARDcy"] = 1066,
3850   ["Ycy"] = 1067,
3851   ["SOFTcy"] = 1068,
3852   ["Ecy"] = 1069,
```

```
3853   ["YUcy"] = 1070,
3854   ["YAcy"] = 1071,
3855   ["acy"] = 1072,
3856   ["bcy"] = 1073,
3857   ["vcy"] = 1074,
3858   ["gcy"] = 1075,
3859   ["dcy"] = 1076,
3860   ["iecy"] = 1077,
3861   ["zhcy"] = 1078,
3862   ["zcy"] = 1079,
3863   ["icy"] = 1080,
3864   ["jcy"] = 1081,
3865   ["kcy"] = 1082,
3866   ["lcy"] = 1083,
3867   ["mcy"] = 1084,
3868   ["ncy"] = 1085,
3869   ["ocy"] = 1086,
3870   ["pcy"] = 1087,
3871   ["rcy"] = 1088,
3872   ["scy"] = 1089,
3873   ["tcy"] = 1090,
3874   ["ucy"] = 1091,
3875   ["fcy"] = 1092,
3876   ["khcy"] = 1093,
3877   ["tscy"] = 1094,
3878   ["chcy"] = 1095,
3879   ["shcy"] = 1096,
3880   ["shchcy"] = 1097,
3881   ["hardcy"] = 1098,
3882   ["ycy"] = 1099,
3883   ["softcy"] = 1100,
3884   ["ecy"] = 1101,
3885   ["yucy"] = 1102,
3886   ["yacy"] = 1103,
3887   ["iocy"] = 1105,
3888   ["djcy"] = 1106,
3889   ["gjcy"] = 1107,
3890   ["jukcy"] = 1108,
3891   ["dscy"] = 1109,
3892   ["iukcy"] = 1110,
3893   ["yicy"] = 1111,
3894   ["jsercy"] = 1112,
3895   ["ljcy"] = 1113,
3896   ["njcy"] = 1114,
3897   ["tshcy"] = 1115,
3898   ["kjcy"] = 1116,
3899   ["ubrcy"] = 1118,
```

```
3900    ["dzcy"] = 1119,
3901    ["ensp"] = 8194,
3902    ["emsp"] = 8195,
3903    ["emsp13"] = 8196,
3904    ["emsp14"] = 8197,
3905    ["numsp"] = 8199,
3906    ["puncsp"] = 8200,
3907    ["ThinSpace"] = 8201,
3908    ["thinsp"] = 8201,
3909    ["VeryThinSpace"] = 8202,
3910    ["hairsp"] = 8202,
3911    ["NegativeMediumSpace"] = 8203,
3912    ["NegativeThickSpace"] = 8203,
3913    ["NegativeThinSpace"] = 8203,
3914    ["NegativeVeryThinSpace"] = 8203,
3915    ["ZeroWidthSpace"] = 8203,
3916    ["zwnj"] = 8204,
3917    ["zwj"] = 8205,
3918    ["lrm"] = 8206,
3919    ["rlm"] = 8207,
3920    ["dash"] = 8208,
3921    ["hyphen"] = 8208,
3922    ["ndash"] = 8211,
3923    ["mdash"] = 8212,
3924    ["horbar"] = 8213,
3925    ["Verbar"] = 8214,
3926    ["Vert"] = 8214,
3927    ["OpenCurlyQuote"] = 8216,
3928    ["lsquo"] = 8216,
3929    ["CloseCurlyQuote"] = 8217,
3930    ["rsquo"] = 8217,
3931    ["rsquor"] = 8217,
3932    ["lsquor"] = 8218,
3933    ["sbquo"] = 8218,
3934    ["OpenCurlyDoubleQuote"] = 8220,
3935    ["ldquo"] = 8220,
3936    ["CloseCurlyDoubleQuote"] = 8221,
3937    ["rdquo"] = 8221,
3938    ["rdquor"] = 8221,
3939    ["bdquo"] = 8222,
3940    ["ldquor"] = 8222,
3941    ["dagger"] = 8224,
3942    ["Dagger"] = 8225,
3943    ["ddagger"] = 8225,
3944    ["bull"] = 8226,
3945    ["bullet"] = 8226,
3946    ["nldr"] = 8229,
```

```
3947   ["hellip"] = 8230,
3948   ["mldr"] = 8230,
3949   ["permil"] = 8240,
3950   ["pertenk"] = 8241,
3951   ["prime"] = 8242,
3952   ["Prime"] = 8243,
3953   ["tprime"] = 8244,
3954   ["backprime"] = 8245,
3955   ["bprime"] = 8245,
3956   ["lsaquo"] = 8249,
3957   ["rsaquo"] = 8250,
3958   ["OverBar"] = 8254,
3959   ["oline"] = 8254,
3960   ["caret"] = 8257,
3961   ["hybull"] = 8259,
3962   ["frasl"] = 8260,
3963   ["bsemi"] = 8271,
3964   ["qprime"] = 8279,
3965   ["MediumSpace"] = 8287,
3966   ["ThickSpace"] = {8287, 8202},
3967   ["NoBreak"] = 8288,
3968   ["ApplyFunction"] = 8289,
3969   ["af"] = 8289,
3970   ["InvisibleTimes"] = 8290,
3971   ["it"] = 8290,
3972   ["InvisibleComma"] = 8291,
3973   ["ic"] = 8291,
3974   ["euro"] = 8364,
3975   ["TripleDot"] = 8411,
3976   ["tdot"] = 8411,
3977   ["DotDot"] = 8412,
3978   ["Copf"] = 8450,
3979   ["complexes"] = 8450,
3980   ["incare"] = 8453,
3981   ["gscr"] = 8458,
3982   ["HilbertSpace"] = 8459,
3983   ["Hscr"] = 8459,
3984   ["hamilt"] = 8459,
3985   ["Hfr"] = 8460,
3986   ["Poincareplane"] = 8460,
3987   ["Hopf"] = 8461,
3988   ["quaternions"] = 8461,
3989   ["planckh"] = 8462,
3990   ["hbar"] = 8463,
3991   ["hslash"] = 8463,
3992   ["planck"] = 8463,
3993   ["plankv"] = 8463,
```

```
3994    ["Iscr"] = 8464,
3995    ["imagline"] = 8464,
3996    ["Ifr"] = 8465,
3997    ["Im"] = 8465,
3998    ["image"] = 8465,
3999    ["imagpart"] = 8465,
4000    ["Laplacetrf"] = 8466,
4001    ["Lscr"] = 8466,
4002    ["lagran"] = 8466,
4003    ["ell"] = 8467,
4004    ["Nopf"] = 8469,
4005    ["naturals"] = 8469,
4006    ["numero"] = 8470,
4007    ["copysr"] = 8471,
4008    ["weierp"] = 8472,
4009    ["wp"] = 8472,
4010    ["Popf"] = 8473,
4011    ["primes"] = 8473,
4012    ["Qopf"] = 8474,
4013    ["rationals"] = 8474,
4014    ["Rscr"] = 8475,
4015    ["realine"] = 8475,
4016    ["Re"] = 8476,
4017    ["Rfr"] = 8476,
4018    ["real"] = 8476,
4019    ["realpart"] = 8476,
4020    ["Ropf"] = 8477,
4021    ["reals"] = 8477,
4022    ["rx"] = 8478,
4023    ["TRADE"] = 8482,
4024    ["trade"] = 8482,
4025    ["Zopf"] = 8484,
4026    ["integers"] = 8484,
4027    ["mho"] = 8487,
4028    ["Zfr"] = 8488,
4029    ["zeetrf"] = 8488,
4030    ["iiota"] = 8489,
4031    ["Bernoullis"] = 8492,
4032    ["Bscr"] = 8492,
4033    ["bernou"] = 8492,
4034    ["Cayleys"] = 8493,
4035    ["Cfr"] = 8493,
4036    ["escr"] = 8495,
4037    ["Escr"] = 8496,
4038    ["expectation"] = 8496,
4039    ["Fouriertrf"] = 8497,
4040    ["Fscr"] = 8497,
```

```
["Mellintrf"] = 8499,
["Mscr"] = 8499,
["phmmat"] = 8499,
["order"] = 8500,
["orderof"] = 8500,
["oscr"] = 8500,
["alefsym"] = 8501,
["aleph"] = 8501,
["beth"] = 8502,
["gimel"] = 8503,
["daleth"] = 8504,
["CapitalDifferentialD"] = 8517,
["DD"] = 8517,
["DifferentialD"] = 8518,
["dd"] = 8518,
["ExponentialE"] = 8519,
["ee"] = 8519,
["exponentiale"] = 8519,
["ImaginaryI"] = 8520,
["ii"] = 8520,
["frac13"] = 8531,
["frac23"] = 8532,
["frac15"] = 8533,
["frac25"] = 8534,
["frac35"] = 8535,
["frac45"] = 8536,
["frac16"] = 8537,
["frac56"] = 8538,
["frac18"] = 8539,
["frac38"] = 8540,
["frac58"] = 8541,
["frac78"] = 8542,
["LeftArrow"] = 8592,
["ShortLeftArrow"] = 8592,
["larr"] = 8592,
["leftarrow"] = 8592,
["slarr"] = 8592,
["ShortUpArrow"] = 8593,
["UpArrow"] = 8593,
["uarr"] = 8593,
["uparrow"] = 8593,
["RightArrow"] = 8594,
["ShortRightArrow"] = 8594,
["rarr"] = 8594,
["rightarrow"] = 8594,
["srarr"] = 8594,
["DownArrow"] = 8595,
```

```
  ["ShortDownArrow"] = 8595,
  ["darr"] = 8595,
  ["downarrow"] = 8595,
  ["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,
```

```
4135  ["hookleftarrow"] = 8617,
4136  ["larrhk"] = 8617,
4137  ["hookrightarrow"] = 8618,
4138  ["rarrhk"] = 8618,
4139  ["larrlp"] = 8619,
4140  ["looparrowleft"] = 8619,
4141  ["looparrowright"] = 8620,
4142  ["rarrlp"] = 8620,
4143  ["harrw"] = 8621,
4144  ["leftrightsquigarrow"] = 8621,
4145  ["nharr"] = 8622,
4146  ["nleftrightarrow"] = 8622,
4147  ["Lsh"] = 8624,
4148  ["lsh"] = 8624,
4149  ["Rsh"] = 8625,
4150  ["rsh"] = 8625,
4151  ["ldsh"] = 8626,
4152  ["rdsh"] = 8627,
4153  ["crarr"] = 8629,
4154  ["cularr"] = 8630,
4155  ["curvearrowleft"] = 8630,
4156  ["curarr"] = 8631,
4157  ["curvearrowright"] = 8631,
4158  ["circlearrowleft"] = 8634,
4159  ["olarr"] = 8634,
4160  ["circlearrowright"] = 8635,
4161  ["orarr"] = 8635,
4162  ["LeftVector"] = 8636,
4163  ["leftharpoonup"] = 8636,
4164  ["lharu"] = 8636,
4165  ["DownLeftVector"] = 8637,
4166  ["leftharpoondown"] = 8637,
4167  ["lhard"] = 8637,
4168  ["RightUpVector"] = 8638,
4169  ["uharr"] = 8638,
4170  ["upharpoonright"] = 8638,
4171  ["LeftUpVector"] = 8639,
4172  ["uharl"] = 8639,
4173  ["upharpoonleft"] = 8639,
4174  ["RightVector"] = 8640,
4175  ["rharu"] = 8640,
4176  ["rightharpoonup"] = 8640,
4177  ["DownRightVector"] = 8641,
4178  ["rhard"] = 8641,
4179  ["rightharpoondown"] = 8641,
4180  ["RightDownVector"] = 8642,
4181  ["dharr"] = 8642,
```

```
["downharpoonright"] = 8642,
["LeftDownVector"] = 8643,
["dharl"] = 8643,
["downharpoonleft"] = 8643,
["RightArrowLeftArrow"] = 8644,
["rightleftarrows"] = 8644,
["rlarr"] = 8644,
["UpArrowDownArrow"] = 8645,
["udarr"] = 8645,
["LeftArrowRightArrow"] = 8646,
["leftrightarrows"] = 8646,
["lrarr"] = 8646,
["leftleftarrows"] = 8647,
["llarr"] = 8647,
["upuparrows"] = 8648,
["uuarr"] = 8648,
["rightrightarrows"] = 8649,
["rrarr"] = 8649,
["ddarr"] = 8650,
["downdownarrows"] = 8650,
["ReverseEquilibrium"] = 8651,
["leftrightharpoons"] = 8651,
["lrhar"] = 8651,
["Equilibrium"] = 8652,
["rightleftharpoons"] = 8652,
["rlhar"] = 8652,
["nLeftarrow"] = 8653,
["nlArr"] = 8653,
["nLeftrightarrow"] = 8654,
["nhArr"] = 8654,
["nRightarrow"] = 8655,
["nrArr"] = 8655,
["DoubleLeftArrow"] = 8656,
["Leftarrow"] = 8656,
["lArr"] = 8656,
["DoubleUpArrow"] = 8657,
["Uparrow"] = 8657,
["uArr"] = 8657,
["DoubleRightArrow"] = 8658,
["Implies"] = 8658,
["Rightarrow"] = 8658,
["rArr"] = 8658,
["DoubleDownArrow"] = 8659,
["Downarrow"] = 8659,
["dArr"] = 8659,
["DoubleLeftRightArrow"] = 8660,
["Leftrightarrow"] = 8660,
```

```
  ["hArr"] = 8660,
  ["iff"] = 8660,
  ["DoubleUpDownArrow"] = 8661,
  ["Updownarrow"] = 8661,
  ["vArr"] = 8661,
  ["nwArr"] = 8662,
  ["neArr"] = 8663,
  ["seArr"] = 8664,
  ["swArr"] = 8665,
  ["Lleftarrow"] = 8666,
  ["lAarr"] = 8666,
  ["Rrightarrow"] = 8667,
  ["rAarr"] = 8667,
  ["zigrarr"] = 8669,
  ["LeftArrowBar"] = 8676,
  ["larrb"] = 8676,
  ["RightArrowBar"] = 8677,
  ["rarrb"] = 8677,
  ["DownArrowUpArrow"] = 8693,
  ["duarr"] = 8693,
  ["loarr"] = 8701,
  ["roarr"] = 8702,
  ["hoarr"] = 8703,
  ["ForAll"] = 8704,
  ["forall"] = 8704,
  ["comp"] = 8705,
  ["complement"] = 8705,
  ["PartialD"] = 8706,
  ["npart"] = {8706, 824},
  ["part"] = 8706,
  ["Exists"] = 8707,
  ["exist"] = 8707,
  ["NotExists"] = 8708,
  ["nexist"] = 8708,
  ["nexists"] = 8708,
  ["empty"] = 8709,
  ["emptyset"] = 8709,
  ["emptyv"] = 8709,
  ["varnothing"] = 8709,
  ["Del"] = 8711,
  ["nabla"] = 8711,
  ["Element"] = 8712,
  ["in"] = 8712,
  ["isin"] = 8712,
  ["isinv"] = 8712,
  ["NotElement"] = 8713,
  ["notin"] = 8713,
```

```
4276    ["notinva"] = 8713,
4277    ["ReverseElement"] = 8715,
4278    ["SuchThat"] = 8715,
4279    ["ni"] = 8715,
4280    ["niv"] = 8715,
4281    ["NotReverseElement"] = 8716,
4282    ["notni"] = 8716,
4283    ["notniva"] = 8716,
4284    ["Product"] = 8719,
4285    ["prod"] = 8719,
4286    ["Coproduct"] = 8720,
4287    ["coprod"] = 8720,
4288    ["Sum"] = 8721,
4289    ["sum"] = 8721,
4290    ["minus"] = 8722,
4291    ["MinusPlus"] = 8723,
4292    ["mnplus"] = 8723,
4293    ["mp"] = 8723,
4294    ["dotplus"] = 8724,
4295    ["plusdo"] = 8724,
4296    ["Backslash"] = 8726,
4297    ["setminus"] = 8726,
4298    ["setmn"] = 8726,
4299    ["smallsetminus"] = 8726,
4300    ["ssetmn"] = 8726,
4301    ["lowast"] = 8727,
4302    ["SmallCircle"] = 8728,
4303    ["compfn"] = 8728,
4304    ["Sqrt"] = 8730,
4305    ["radic"] = 8730,
4306    ["Proportional"] = 8733,
4307    ["prop"] = 8733,
4308    ["propto"] = 8733,
4309    ["varpropto"] = 8733,
4310    ["vprop"] = 8733,
4311    ["infin"] = 8734,
4312    ["angrt"] = 8735,
4313    ["ang"] = 8736,
4314    ["angle"] = 8736,
4315    ["nang"] = {8736, 8402},
4316    ["angmsd"] = 8737,
4317    ["measuredangle"] = 8737,
4318    ["angsph"] = 8738,
4319    ["VerticalBar"] = 8739,
4320    ["mid"] = 8739,
4321    ["shortmid"] = 8739,
4322    ["smid"] = 8739,
```

```
4323    ["NotVerticalBar"] = 8740,
4324    ["nmid"] = 8740,
4325    ["nshortmid"] = 8740,
4326    ["nsmid"] = 8740,
4327    ["DoubleVerticalBar"] = 8741,
4328    ["par"] = 8741,
4329    ["parallel"] = 8741,
4330    ["shortparallel"] = 8741,
4331    ["spar"] = 8741,
4332    ["NotDoubleVerticalBar"] = 8742,
4333    ["npar"] = 8742,
4334    ["nparallel"] = 8742,
4335    ["nshortparallel"] = 8742,
4336    ["nspar"] = 8742,
4337    ["and"] = 8743,
4338    ["wedge"] = 8743,
4339    ["or"] = 8744,
4340    ["vee"] = 8744,
4341    ["cap"] = 8745,
4342    ["caps"] = {8745, 65024},
4343    ["cup"] = 8746,
4344    ["cups"] = {8746, 65024},
4345    ["Integral"] = 8747,
4346    ["int"] = 8747,
4347    ["Int"] = 8748,
4348    ["iiint"] = 8749,
4349    ["tint"] = 8749,
4350    ["ContourIntegral"] = 8750,
4351    ["conint"] = 8750,
4352    ["oint"] = 8750,
4353    ["Conint"] = 8751,
4354    ["DoubleContourIntegral"] = 8751,
4355    ["Cconint"] = 8752,
4356    ["cwint"] = 8753,
4357    ["ClockwiseContourIntegral"] = 8754,
4358    ["cwconint"] = 8754,
4359    ["CounterClockwiseContourIntegral"] = 8755,
4360    ["awconint"] = 8755,
4361    ["Therefore"] = 8756,
4362    ["there4"] = 8756,
4363    ["therefore"] = 8756,
4364    ["Because"] = 8757,
4365    ["becaus"] = 8757,
4366    ["because"] = 8757,
4367    ["ratio"] = 8758,
4368    ["Colon"] = 8759,
4369    ["Proportion"] = 8759,
```

```
4370   ["dotminus"] = 8760,
4371   ["minusd"] = 8760,
4372   ["mDDot"] = 8762,
4373   ["homtht"] = 8763,
4374   ["Tilde"] = 8764,
4375   ["nvsim"] = {8764, 8402},
4376   ["sim"] = 8764,
4377   ["thicksim"] = 8764,
4378   ["thksim"] = 8764,
4379   ["backsim"] = 8765,
4380   ["bsim"] = 8765,
4381   ["race"] = {8765, 817},
4382   ["ac"] = 8766,
4383   ["acE"] = {8766, 819},
4384   ["mstpos"] = 8766,
4385   ["acd"] = 8767,
4386   ["VerticalTilde"] = 8768,
4387   ["wr"] = 8768,
4388   ["wreath"] = 8768,
4389   ["NotTilde"] = 8769,
4390   ["nsim"] = 8769,
4391   ["EqualTilde"] = 8770,
4392   ["NotEqualTilde"] = {8770, 824},
4393   ["eqsim"] = 8770,
4394   ["esim"] = 8770,
4395   ["nesim"] = {8770, 824},
4396   ["TildeEqual"] = 8771,
4397   ["sime"] = 8771,
4398   ["simeq"] = 8771,
4399   ["NotTildeEqual"] = 8772,
4400   ["nsime"] = 8772,
4401   ["nsimeq"] = 8772,
4402   ["TildeFullEqual"] = 8773,
4403   ["cong"] = 8773,
4404   ["simne"] = 8774,
4405   ["NotTildeFullEqual"] = 8775,
4406   ["ncong"] = 8775,
4407   ["TildeTilde"] = 8776,
4408   ["ap"] = 8776,
4409   ["approx"] = 8776,
4410   ["asymp"] = 8776,
4411   ["thickapprox"] = 8776,
4412   ["thkap"] = 8776,
4413   ["NotTildeTilde"] = 8777,
4414   ["nap"] = 8777,
4415   ["napprox"] = 8777,
4416   ["ape"] = 8778,
```

```
4417  ["approxeq"] = 8778,
4418  ["apid"] = 8779,
4419  ["napid"] = {8779, 824},
4420  ["backcong"] = 8780,
4421  ["bcong"] = 8780,
4422  ["CupCap"] = 8781,
4423  ["asympeq"] = 8781,
4424  ["nvap"] = {8781, 8402},
4425  ["Bumpeq"] = 8782,
4426  ["HumpDownHump"] = 8782,
4427  ["NotHumpDownHump"] = {8782, 824},
4428  ["bump"] = 8782,
4429  ["nbump"] = {8782, 824},
4430  ["HumpEqual"] = 8783,
4431  ["NotHumpEqual"] = {8783, 824},
4432  ["bumpe"] = 8783,
4433  ["bumpeq"] = 8783,
4434  ["nbumpe"] = {8783, 824},
4435  ["DotEqual"] = 8784,
4436  ["doteq"] = 8784,
4437  ["esdot"] = 8784,
4438  ["nedot"] = {8784, 824},
4439  ["doteqdot"] = 8785,
4440  ["eDot"] = 8785,
4441  ["efDot"] = 8786,
4442  ["fallingdotseq"] = 8786,
4443  ["erDot"] = 8787,
4444  ["risingdotseq"] = 8787,
4445  ["Assign"] = 8788,
4446  ["colone"] = 8788,
4447  ["coloneq"] = 8788,
4448  ["ecolon"] = 8789,
4449  ["eqcolon"] = 8789,
4450  ["ecir"] = 8790,
4451  ["eqcirc"] = 8790,
4452  ["circeq"] = 8791,
4453  ["cire"] = 8791,
4454  ["wedgeq"] = 8793,
4455  ["veeeq"] = 8794,
4456  ["triangleq"] = 8796,
4457  ["trie"] = 8796,
4458  ["equest"] = 8799,
4459  ["questeq"] = 8799,
4460  ["NotEqual"] = 8800,
4461  ["ne"] = 8800,
4462  ["Congruent"] = 8801,
4463  ["bnequiv"] = {8801, 8421},
```

```
4464    ["equiv"] = 8801,
4465    ["NotCongruent"] = 8802,
4466    ["nequiv"] = 8802,
4467    ["le"] = 8804,
4468    ["leq"] = 8804,
4469    ["nvle"] = {8804, 8402},
4470    ["GreaterEqual"] = 8805,
4471    ["ge"] = 8805,
4472    ["geq"] = 8805,
4473    ["nvge"] = {8805, 8402},
4474    ["LessFullEqual"] = 8806,
4475    ["lE"] = 8806,
4476    ["leqq"] = 8806,
4477    ["nlE"] = {8806, 824},
4478    ["nleqq"] = {8806, 824},
4479    ["GreaterFullEqual"] = 8807,
4480    ["NotGreaterFullEqual"] = {8807, 824},
4481    ["gE"] = 8807,
4482    ["geqq"] = 8807,
4483    ["ngE"] = {8807, 824},
4484    ["ngeqq"] = {8807, 824},
4485    ["lnE"] = 8808,
4486    ["lneqq"] = 8808,
4487    ["lvertneqq"] = {8808, 65024},
4488    ["lvnE"] = {8808, 65024},
4489    ["gnE"] = 8809,
4490    ["gneqq"] = 8809,
4491    ["gvertneqq"] = {8809, 65024},
4492    ["gvnE"] = {8809, 65024},
4493    ["Lt"] = 8810,
4494    ["NestedLessLess"] = 8810,
4495    ["NotLessLess"] = {8810, 824},
4496    ["ll"] = 8810,
4497    ["nLt"] = {8810, 8402},
4498    ["nLtv"] = {8810, 824},
4499    ["Gt"] = 8811,
4500    ["NestedGreaterGreater"] = 8811,
4501    ["NotGreaterGreater"] = {8811, 824},
4502    ["gg"] = 8811,
4503    ["nGt"] = {8811, 8402},
4504    ["nGtv"] = {8811, 824},
4505    ["between"] = 8812,
4506    ["twixt"] = 8812,
4507    ["NotCupCap"] = 8813,
4508    ["NotLess"] = 8814,
4509    ["nless"] = 8814,
4510    ["nlt"] = 8814,
```

```
["NotGreater"] = 8815,
["ngt"] = 8815,
["ngtr"] = 8815,
["NotLessEqual"] = 8816,
["nle"] = 8816,
["nleq"] = 8816,
["NotGreaterEqual"] = 8817,
["nge"] = 8817,
["ngeq"] = 8817,
["LessTilde"] = 8818,
["lesssim"] = 8818,
["lsim"] = 8818,
["GreaterTilde"] = 8819,
["gsim"] = 8819,
["gtrsim"] = 8819,
["NotLessTilde"] = 8820,
["nlsim"] = 8820,
["NotGreaterTilde"] = 8821,
["ngsim"] = 8821,
["LessGreater"] = 8822,
["lessgtr"] = 8822,
["lg"] = 8822,
["GreaterLess"] = 8823,
["gl"] = 8823,
["gtrless"] = 8823,
["NotLessGreater"] = 8824,
["ntlg"] = 8824,
["NotGreaterLess"] = 8825,
["ntgl"] = 8825,
["Precedes"] = 8826,
["pr"] = 8826,
["prec"] = 8826,
["Succeeds"] = 8827,
["sc"] = 8827,
["succ"] = 8827,
["PrecedesSlantEqual"] = 8828,
["prcue"] = 8828,
["preccurlyeq"] = 8828,
["SucceedsSlantEqual"] = 8829,
["sccue"] = 8829,
["succcurlyeq"] = 8829,
["PrecedesTilde"] = 8830,
["precsim"] = 8830,
["prsim"] = 8830,
["NotSucceedsTilde"] = {8831, 824},
["SucceedsTilde"] = 8831,
["scsim"] = 8831,
```

```
4558  ["succsim"] = 8831,
4559  ["NotPrecedes"] = 8832,
4560  ["npr"] = 8832,
4561  ["nprec"] = 8832,
4562  ["NotSucceeds"] = 8833,
4563  ["nsc"] = 8833,
4564  ["nsucc"] = 8833,
4565  ["NotSubset"] = {8834, 8402},
4566  ["nsubset"] = {8834, 8402},
4567  ["sub"] = 8834,
4568  ["subset"] = 8834,
4569  ["vnsub"] = {8834, 8402},
4570  ["NotSuperset"] = {8835, 8402},
4571  ["Superset"] = 8835,
4572  ["nsupset"] = {8835, 8402},
4573  ["sup"] = 8835,
4574  ["supset"] = 8835,
4575  ["vnsup"] = {8835, 8402},
4576  ["nsub"] = 8836,
4577  ["nsup"] = 8837,
4578  ["SubsetEqual"] = 8838,
4579  ["sube"] = 8838,
4580  ["subseteq"] = 8838,
4581  ["SupersetEqual"] = 8839,
4582  ["supe"] = 8839,
4583  ["supseteq"] = 8839,
4584  ["NotSubsetEqual"] = 8840,
4585  ["nsube"] = 8840,
4586  ["nsubseteq"] = 8840,
4587  ["NotSupersetEqual"] = 8841,
4588  ["nsupe"] = 8841,
4589  ["nsupseteq"] = 8841,
4590  ["subne"] = 8842,
4591  ["subsetneq"] = 8842,
4592  ["varsubsetneq"] = {8842, 65024},
4593  ["vsubne"] = {8842, 65024},
4594  ["supne"] = 8843,
4595  ["supsetneq"] = 8843,
4596  ["varsupsetneq"] = {8843, 65024},
4597  ["vsupne"] = {8843, 65024},
4598  ["cupdot"] = 8845,
4599  ["UnionPlus"] = 8846,
4600  ["uplus"] = 8846,
4601  ["NotSquareSubset"] = {8847, 824},
4602  ["SquareSubset"] = 8847,
4603  ["sqsub"] = 8847,
4604  ["sqsubset"] = 8847,
```

```
["NotSquareSuperset"] = {8848, 824},
["SquareSuperset"] = 8848,
["sqsup"] = 8848,
["sqsupset"] = 8848,
["SquareSubsetEqual"] = 8849,
["sqsube"] = 8849,
["sqsubseteq"] = 8849,
["SquareSupersetEqual"] = 8850,
["sqsupe"] = 8850,
["sqsupseteq"] = 8850,
["SquareIntersection"] = 8851,
["sqcap"] = 8851,
["sqcaps"] = {8851, 65024},
["SquareUnion"] = 8852,
["sqcup"] = 8852,
["sqcups"] = {8852, 65024},
["CirclePlus"] = 8853,
["oplus"] = 8853,
["CircleMinus"] = 8854,
["ominus"] = 8854,
["CircleTimes"] = 8855,
["otimes"] = 8855,
["osol"] = 8856,
["CircleDot"] = 8857,
["odot"] = 8857,
["circledcirc"] = 8858,
["ocir"] = 8858,
["circledast"] = 8859,
["oast"] = 8859,
["circleddash"] = 8861,
["odash"] = 8861,
["boxplus"] = 8862,
["plusb"] = 8862,
["boxminus"] = 8863,
["minusb"] = 8863,
["boxtimes"] = 8864,
["timesb"] = 8864,
["dotsquare"] = 8865,
["sdotb"] = 8865,
["RightTee"] = 8866,
["vdash"] = 8866,
["LeftTee"] = 8867,
["dashv"] = 8867,
["DownTee"] = 8868,
["top"] = 8868,
["UpTee"] = 8869,
["bot"] = 8869,
```

```
4652  ["bottom"] = 8869,
4653  ["perp"] = 8869,
4654  ["models"] = 8871,
4655  ["DoubleRightTee"] = 8872,
4656  ["vDash"] = 8872,
4657  ["Vdash"] = 8873,
4658  ["Vvdash"] = 8874,
4659  ["VDash"] = 8875,
4660  ["nvdash"] = 8876,
4661  ["nvDash"] = 8877,
4662  ["nVdash"] = 8878,
4663  ["nVDash"] = 8879,
4664  ["prurel"] = 8880,
4665  ["LeftTriangle"] = 8882,
4666  ["vartriangleleft"] = 8882,
4667  ["vltri"] = 8882,
4668  ["RightTriangle"] = 8883,
4669  ["vartriangleright"] = 8883,
4670  ["vrtri"] = 8883,
4671  ["LeftTriangleEqual"] = 8884,
4672  ["ltrie"] = 8884,
4673  ["nvltrie"] = {8884, 8402},
4674  ["trianglelefteq"] = 8884,
4675  ["RightTriangleEqual"] = 8885,
4676  ["nvrtrie"] = {8885, 8402},
4677  ["rtrie"] = 8885,
4678  ["trianglerighteq"] = 8885,
4679  ["origof"] = 8886,
4680  ["imof"] = 8887,
4681  ["multimap"] = 8888,
4682  ["mumap"] = 8888,
4683  ["hercon"] = 8889,
4684  ["intcal"] = 8890,
4685  ["intercal"] = 8890,
4686  ["veebar"] = 8891,
4687  ["barvee"] = 8893,
4688  ["angrtvb"] = 8894,
4689  ["lrtri"] = 8895,
4690  ["Wedge"] = 8896,
4691  ["bigwedge"] = 8896,
4692  ["xwedge"] = 8896,
4693  ["Vee"] = 8897,
4694  ["bigvee"] = 8897,
4695  ["xvee"] = 8897,
4696  ["Intersection"] = 8898,
4697  ["bigcap"] = 8898,
4698  ["xcap"] = 8898,
```

```
["Union"] = 8899,
["bigcup"] = 8899,
["xcup"] = 8899,
["Diamond"] = 8900,
["diam"] = 8900,
["diamond"] = 8900,
["sdot"] = 8901,
["Star"] = 8902,
["sstarf"] = 8902,
["divideontimes"] = 8903,
["divonx"] = 8903,
["bowtie"] = 8904,
["ltimes"] = 8905,
["rtimes"] = 8906,
["leftthreetimes"] = 8907,
["lthree"] = 8907,
["rightthreetimes"] = 8908,
["rthree"] = 8908,
["backsimeq"] = 8909,
["bsime"] = 8909,
["curlyvee"] = 8910,
["cuvee"] = 8910,
["curlywedge"] = 8911,
["cuwed"] = 8911,
["Sub"] = 8912,
["Subset"] = 8912,
["Sup"] = 8913,
["Supset"] = 8913,
["Cap"] = 8914,
["Cup"] = 8915,
["fork"] = 8916,
["pitchfork"] = 8916,
["epar"] = 8917,
["lessdot"] = 8918,
["ltdot"] = 8918,
["gtdot"] = 8919,
["gtrdot"] = 8919,
["Ll"] = 8920,
["nLl"] = {8920, 824},
["Gg"] = 8921,
["ggg"] = 8921,
["nGg"] = {8921, 824},
["LessEqualGreater"] = 8922,
["leg"] = 8922,
["lesg"] = {8922, 65024},
["lesseqgtr"] = 8922,
["GreaterEqualLess"] = 8923,
```

```
["gel"] = 8923,
["gesl"] = {8923, 65024},
["gtreqless"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["NotPrecedesSlantEqual"] = 8928,
["nprcue"] = 8928,
["NotSucceedsSlantEqual"] = 8929,
["nsccue"] = 8929,
["NotSquareSubsetEqual"] = 8930,
["nsqsube"] = 8930,
["NotSquareSupersetEqual"] = 8931,
["nsqsupe"] = 8931,
["lnsim"] = 8934,
["gnsim"] = 8935,
["precnsim"] = 8936,
["prnsim"] = 8936,
["scnsim"] = 8937,
["succnsim"] = 8937,
["NotLeftTriangle"] = 8938,
["nltri"] = 8938,
["ntriangleleft"] = 8938,
["NotRightTriangle"] = 8939,
["nrtri"] = 8939,
["ntriangleright"] = 8939,
["NotLeftTriangleEqual"] = 8940,
["nltrie"] = 8940,
["ntrianglelefteq"] = 8940,
["NotRightTriangleEqual"] = 8941,
["nrtrie"] = 8941,
["ntrianglerighteq"] = 8941,
["vellip"] = 8942,
["ctdot"] = 8943,
["utdot"] = 8944,
["dtdot"] = 8945,
["disin"] = 8946,
["isinsv"] = 8947,
["isins"] = 8948,
["isindot"] = 8949,
["notindot"] = {8949, 824},
["notinvc"] = 8950,
["notinvb"] = 8951,
["isinE"] = 8953,
["notinE"] = {8953, 824},
["nisd"] = 8954,
```

```
4793  ["xnis"] = 8955,
4794  ["nis"] = 8956,
4795  ["notnivc"] = 8957,
4796  ["notnivb"] = 8958,
4797  ["barwed"] = 8965,
4798  ["barwedge"] = 8965,
4799  ["Barwed"] = 8966,
4800  ["doublebarwedge"] = 8966,
4801  ["LeftCeiling"] = 8968,
4802  ["lceil"] = 8968,
4803  ["RightCeiling"] = 8969,
4804  ["rceil"] = 8969,
4805  ["LeftFloor"] = 8970,
4806  ["lfloor"] = 8970,
4807  ["RightFloor"] = 8971,
4808  ["rfloor"] = 8971,
4809  ["drcrop"] = 8972,
4810  ["dlcrop"] = 8973,
4811  ["urcrop"] = 8974,
4812  ["ulcrop"] = 8975,
4813  ["bnot"] = 8976,
4814  ["profline"] = 8978,
4815  ["profsurf"] = 8979,
4816  ["telrec"] = 8981,
4817  ["target"] = 8982,
4818  ["ulcorn"] = 8988,
4819  ["ulcorner"] = 8988,
4820  ["urcorn"] = 8989,
4821  ["urcorner"] = 8989,
4822  ["dlcorn"] = 8990,
4823  ["llcorner"] = 8990,
4824  ["drcorn"] = 8991,
4825  ["lrcorner"] = 8991,
4826  ["frown"] = 8994,
4827  ["sfrown"] = 8994,
4828  ["smile"] = 8995,
4829  ["ssmile"] = 8995,
4830  ["cylcty"] = 9005,
4831  ["profalar"] = 9006,
4832  ["topbot"] = 9014,
4833  ["ovbar"] = 9021,
4834  ["solbar"] = 9023,
4835  ["angzarr"] = 9084,
4836  ["lmoust"] = 9136,
4837  ["lmoustache"] = 9136,
4838  ["rmoust"] = 9137,
4839  ["rmoustache"] = 9137,
```

```
4840   ["OverBracket"] = 9140,
4841   ["tbrk"] = 9140,
4842   ["UnderBracket"] = 9141,
4843   ["bbrk"] = 9141,
4844   ["bbrktbrk"] = 9142,
4845   ["OverParenthesis"] = 9180,
4846   ["UnderParenthesis"] = 9181,
4847   ["OverBrace"] = 9182,
4848   ["UnderBrace"] = 9183,
4849   ["trpezium"] = 9186,
4850   ["elinters"] = 9191,
4851   ["blank"] = 9251,
4852   ["circledS"] = 9416,
4853   ["oS"] = 9416,
4854   ["HorizontalLine"] = 9472,
4855   ["boxh"] = 9472,
4856   ["boxv"] = 9474,
4857   ["boxdr"] = 9484,
4858   ["boxdl"] = 9488,
4859   ["boxur"] = 9492,
4860   ["boxul"] = 9496,
4861   ["boxvr"] = 9500,
4862   ["boxvl"] = 9508,
4863   ["boxhd"] = 9516,
4864   ["boxhu"] = 9524,
4865   ["boxvh"] = 9532,
4866   ["boxH"] = 9552,
4867   ["boxV"] = 9553,
4868   ["boxdR"] = 9554,
4869   ["boxDr"] = 9555,
4870   ["boxDR"] = 9556,
4871   ["boxdL"] = 9557,
4872   ["boxDl"] = 9558,
4873   ["boxDL"] = 9559,
4874   ["boxuR"] = 9560,
4875   ["boxUr"] = 9561,
4876   ["boxUR"] = 9562,
4877   ["boxuL"] = 9563,
4878   ["boxUl"] = 9564,
4879   ["boxUL"] = 9565,
4880   ["boxvR"] = 9566,
4881   ["boxVr"] = 9567,
4882   ["boxVR"] = 9568,
4883   ["boxvL"] = 9569,
4884   ["boxVl"] = 9570,
4885   ["boxVL"] = 9571,
4886   ["boxHd"] = 9572,
```

```
4887   ["boxhD"] = 9573,
4888   ["boxHD"] = 9574,
4889   ["boxHu"] = 9575,
4890   ["boxhU"] = 9576,
4891   ["boxHU"] = 9577,
4892   ["boxvH"] = 9578,
4893   ["boxVh"] = 9579,
4894   ["boxVH"] = 9580,
4895   ["uhblk"] = 9600,
4896   ["lhblk"] = 9604,
4897   ["block"] = 9608,
4898   ["blk14"] = 9617,
4899   ["blk12"] = 9618,
4900   ["blk34"] = 9619,
4901   ["Square"] = 9633,
4902   ["squ"] = 9633,
4903   ["square"] = 9633,
4904   ["FilledVerySmallSquare"] = 9642,
4905   ["blacksquare"] = 9642,
4906   ["squarf"] = 9642,
4907   ["squf"] = 9642,
4908   ["EmptyVerySmallSquare"] = 9643,
4909   ["rect"] = 9645,
4910   ["marker"] = 9646,
4911   ["fltns"] = 9649,
4912   ["bigtriangleup"] = 9651,
4913   ["xutri"] = 9651,
4914   ["blacktriangle"] = 9652,
4915   ["utrif"] = 9652,
4916   ["triangle"] = 9653,
4917   ["utri"] = 9653,
4918   ["blacktriangleright"] = 9656,
4919   ["rtrif"] = 9656,
4920   ["rtri"] = 9657,
4921   ["triangleright"] = 9657,
4922   ["bigtriangledown"] = 9661,
4923   ["xdtri"] = 9661,
4924   ["blacktriangledown"] = 9662,
4925   ["dtrif"] = 9662,
4926   ["dtri"] = 9663,
4927   ["triangledown"] = 9663,
4928   ["blacktriangleleft"] = 9666,
4929   ["ltrif"] = 9666,
4930   ["ltri"] = 9667,
4931   ["triangleleft"] = 9667,
4932   ["loz"] = 9674,
4933   ["lozenge"] = 9674,
```

```
  ["cir"] = 9675,
  ["tridot"] = 9708,
  ["bigcirc"] = 9711,
  ["xcirc"] = 9711,
  ["ultri"] = 9720,
  ["urtri"] = 9721,
  ["lltri"] = 9722,
  ["EmptySmallSquare"] = 9723,
  ["FilledSmallSquare"] = 9724,
  ["bigstar"] = 9733,
  ["starf"] = 9733,
  ["star"] = 9734,
  ["phone"] = 9742,
  ["female"] = 9792,
  ["male"] = 9794,
  ["spades"] = 9824,
  ["spadesuit"] = 9824,
  ["clubs"] = 9827,
  ["clubsuit"] = 9827,
  ["hearts"] = 9829,
  ["heartsuit"] = 9829,
  ["diamondsuit"] = 9830,
  ["diams"] = 9830,
  ["sung"] = 9834,
  ["flat"] = 9837,
  ["natur"] = 9838,
  ["natural"] = 9838,
  ["sharp"] = 9839,
  ["check"] = 10003,
  ["checkmark"] = 10003,
  ["cross"] = 10007,
  ["malt"] = 10016,
  ["maltese"] = 10016,
  ["sext"] = 10038,
  ["VerticalSeparator"] = 10072,
  ["lbbrk"] = 10098,
  ["rbbrk"] = 10099,
  ["bsolhsub"] = 10184,
  ["suphsol"] = 10185,
  ["LeftDoubleBracket"] = 10214,
  ["lobrk"] = 10214,
  ["RightDoubleBracket"] = 10215,
  ["robrk"] = 10215,
  ["LeftAngleBracket"] = 10216,
  ["lang"] = 10216,
  ["langle"] = 10216,
  ["RightAngleBracket"] = 10217,
```

```
4981  ["rang"] = 10217,
4982  ["rangle"] = 10217,
4983  ["Lang"] = 10218,
4984  ["Rang"] = 10219,
4985  ["loang"] = 10220,
4986  ["roang"] = 10221,
4987  ["LongLeftArrow"] = 10229,
4988  ["longleftarrow"] = 10229,
4989  ["xlarr"] = 10229,
4990  ["LongRightArrow"] = 10230,
4991  ["longrightarrow"] = 10230,
4992  ["xrarr"] = 10230,
4993  ["LongLeftRightArrow"] = 10231,
4994  ["longleftrightarrow"] = 10231,
4995  ["xharr"] = 10231,
4996  ["DoubleLongLeftArrow"] = 10232,
4997  ["Longleftarrow"] = 10232,
4998  ["xlArr"] = 10232,
4999  ["DoubleLongRightArrow"] = 10233,
5000  ["Longrightarrow"] = 10233,
5001  ["xrArr"] = 10233,
5002  ["DoubleLongLeftRightArrow"] = 10234,
5003  ["Longleftrightarrow"] = 10234,
5004  ["xhArr"] = 10234,
5005  ["longmapsto"] = 10236,
5006  ["xmap"] = 10236,
5007  ["dzigrarr"] = 10239,
5008  ["nvlArr"] = 10498,
5009  ["nvrArr"] = 10499,
5010  ["nvHarr"] = 10500,
5011  ["Map"] = 10501,
5012  ["lbarr"] = 10508,
5013  ["bkarow"] = 10509,
5014  ["rbarr"] = 10509,
5015  ["lBarr"] = 10510,
5016  ["dbkarow"] = 10511,
5017  ["rBarr"] = 10511,
5018  ["RBarr"] = 10512,
5019  ["drbkarow"] = 10512,
5020  ["DDotrahd"] = 10513,
5021  ["UpArrowBar"] = 10514,
5022  ["DownArrowBar"] = 10515,
5023  ["Rarrtl"] = 10518,
5024  ["latail"] = 10521,
5025  ["ratail"] = 10522,
5026  ["lAtail"] = 10523,
5027  ["rAtail"] = 10524,
```

```
  ["larrfs"] = 10525,
  ["rarrfs"] = 10526,
  ["larrbfs"] = 10527,
  ["rarrbfs"] = 10528,
  ["nwarhk"] = 10531,
  ["nearhk"] = 10532,
  ["hksearow"] = 10533,
  ["searhk"] = 10533,
  ["hkswarow"] = 10534,
  ["swarhk"] = 10534,
  ["nwnear"] = 10535,
  ["nesear"] = 10536,
  ["toea"] = 10536,
  ["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,
```

```
5075    ["DownRightTeeVector"] = 10591,
5076    ["LeftUpTeeVector"] = 10592,
5077    ["LeftDownTeeVector"] = 10593,
5078    ["lHar"] = 10594,
5079    ["uHar"] = 10595,
5080    ["rHar"] = 10596,
5081    ["dHar"] = 10597,
5082    ["luruhar"] = 10598,
5083    ["ldrdhar"] = 10599,
5084    ["ruluhar"] = 10600,
5085    ["rdldhar"] = 10601,
5086    ["lharul"] = 10602,
5087    ["llhard"] = 10603,
5088    ["rharul"] = 10604,
5089    ["lrhard"] = 10605,
5090    ["UpEquilibrium"] = 10606,
5091    ["udhar"] = 10606,
5092    ["ReverseUpEquilibrium"] = 10607,
5093    ["duhar"] = 10607,
5094    ["RoundImplies"] = 10608,
5095    ["erarr"] = 10609,
5096    ["simrarr"] = 10610,
5097    ["larrsim"] = 10611,
5098    ["rarrsim"] = 10612,
5099    ["rarrap"] = 10613,
5100    ["ltlarr"] = 10614,
5101    ["gtrarr"] = 10616,
5102    ["subrarr"] = 10617,
5103    ["suplarr"] = 10619,
5104    ["lfisht"] = 10620,
5105    ["rfisht"] = 10621,
5106    ["ufisht"] = 10622,
5107    ["dfisht"] = 10623,
5108    ["lopar"] = 10629,
5109    ["ropar"] = 10630,
5110    ["lbrke"] = 10635,
5111    ["rbrke"] = 10636,
5112    ["lbrkslu"] = 10637,
5113    ["rbrksld"] = 10638,
5114    ["lbrksld"] = 10639,
5115    ["rbrkslu"] = 10640,
5116    ["langd"] = 10641,
5117    ["rangd"] = 10642,
5118    ["lparlt"] = 10643,
5119    ["rpargt"] = 10644,
5120    ["gtlPar"] = 10645,
5121    ["ltrPar"] = 10646,
```

```
5122  ["vzigzag"] = 10650,
5123  ["vangrt"] = 10652,
5124  ["angrtvbd"] = 10653,
5125  ["ange"] = 10660,
5126  ["range"] = 10661,
5127  ["dwangle"] = 10662,
5128  ["uwangle"] = 10663,
5129  ["angmsdaa"] = 10664,
5130  ["angmsdab"] = 10665,
5131  ["angmsdac"] = 10666,
5132  ["angmsdad"] = 10667,
5133  ["angmsdae"] = 10668,
5134  ["angmsdaf"] = 10669,
5135  ["angmsdag"] = 10670,
5136  ["angmsdah"] = 10671,
5137  ["bemptyv"] = 10672,
5138  ["demptyv"] = 10673,
5139  ["cemptyv"] = 10674,
5140  ["raemptyv"] = 10675,
5141  ["laemptyv"] = 10676,
5142  ["ohbar"] = 10677,
5143  ["omid"] = 10678,
5144  ["opar"] = 10679,
5145  ["operp"] = 10681,
5146  ["olcross"] = 10683,
5147  ["odsold"] = 10684,
5148  ["olcir"] = 10686,
5149  ["ofcir"] = 10687,
5150  ["olt"] = 10688,
5151  ["ogt"] = 10689,
5152  ["cirscir"] = 10690,
5153  ["cirE"] = 10691,
5154  ["solb"] = 10692,
5155  ["bsolb"] = 10693,
5156  ["boxbox"] = 10697,
5157  ["trisb"] = 10701,
5158  ["rtriltri"] = 10702,
5159  ["LeftTriangleBar"] = 10703,
5160  ["NotLeftTriangleBar"] = {10703, 824},
5161  ["NotRightTriangleBar"] = {10704, 824},
5162  ["RightTriangleBar"] = 10704,
5163  ["iinfin"] = 10716,
5164  ["infintie"] = 10717,
5165  ["nvinfin"] = 10718,
5166  ["eparsl"] = 10723,
5167  ["smeparsl"] = 10724,
5168  ["eqvparsl"] = 10725,
```

```
5169   ["blacklozenge"] = 10731,
5170   ["lozf"] = 10731,
5171   ["RuleDelayed"] = 10740,
5172   ["dsol"] = 10742,
5173   ["bigodot"] = 10752,
5174   ["xodot"] = 10752,
5175   ["bigoplus"] = 10753,
5176   ["xoplus"] = 10753,
5177   ["bigotimes"] = 10754,
5178   ["xotime"] = 10754,
5179   ["biguplus"] = 10756,
5180   ["xuplus"] = 10756,
5181   ["bigsqcup"] = 10758,
5182   ["xsqcup"] = 10758,
5183   ["iiiint"] = 10764,
5184   ["qint"] = 10764,
5185   ["fpartint"] = 10765,
5186   ["cirfnint"] = 10768,
5187   ["awint"] = 10769,
5188   ["rppolint"] = 10770,
5189   ["scpolint"] = 10771,
5190   ["npolint"] = 10772,
5191   ["pointint"] = 10773,
5192   ["quatint"] = 10774,
5193   ["intlarhk"] = 10775,
5194   ["pluscir"] = 10786,
5195   ["plusacir"] = 10787,
5196   ["simplus"] = 10788,
5197   ["plusdu"] = 10789,
5198   ["plussim"] = 10790,
5199   ["plustwo"] = 10791,
5200   ["mcomma"] = 10793,
5201   ["minusdu"] = 10794,
5202   ["loplus"] = 10797,
5203   ["roplus"] = 10798,
5204   ["Cross"] = 10799,
5205   ["timesd"] = 10800,
5206   ["timesbar"] = 10801,
5207   ["smashp"] = 10803,
5208   ["lotimes"] = 10804,
5209   ["rotimes"] = 10805,
5210   ["otimesas"] = 10806,
5211   ["Otimes"] = 10807,
5212   ["odiv"] = 10808,
5213   ["triplus"] = 10809,
5214   ["triminus"] = 10810,
5215   ["tritime"] = 10811,
```

```
5216   ["intprod"] = 10812,
5217   ["iprod"] = 10812,
5218   ["amalg"] = 10815,
5219   ["capdot"] = 10816,
5220   ["ncup"] = 10818,
5221   ["ncap"] = 10819,
5222   ["capand"] = 10820,
5223   ["cupor"] = 10821,
5224   ["cupcap"] = 10822,
5225   ["capcup"] = 10823,
5226   ["cupbrcap"] = 10824,
5227   ["capbrcup"] = 10825,
5228   ["cupcup"] = 10826,
5229   ["capcap"] = 10827,
5230   ["ccups"] = 10828,
5231   ["ccaps"] = 10829,
5232   ["ccupssm"] = 10832,
5233   ["And"] = 10835,
5234   ["Or"] = 10836,
5235   ["andand"] = 10837,
5236   ["oror"] = 10838,
5237   ["orslope"] = 10839,
5238   ["andslope"] = 10840,
5239   ["andv"] = 10842,
5240   ["orv"] = 10843,
5241   ["andd"] = 10844,
5242   ["ord"] = 10845,
5243   ["wedbar"] = 10847,
5244   ["sdote"] = 10854,
5245   ["simdot"] = 10858,
5246   ["congdot"] = 10861,
5247   ["ncongdot"] = {10861, 824},
5248   ["easter"] = 10862,
5249   ["apacir"] = 10863,
5250   ["apE"] = 10864,
5251   ["napE"] = {10864, 824},
5252   ["eplus"] = 10865,
5253   ["pluse"] = 10866,
5254   ["Esim"] = 10867,
5255   ["Colone"] = 10868,
5256   ["Equal"] = 10869,
5257   ["ddotseq"] = 10871,
5258   ["eDDot"] = 10871,
5259   ["equivDD"] = 10872,
5260   ["ltcir"] = 10873,
5261   ["gtcir"] = 10874,
5262   ["ltquest"] = 10875,
```

```
5263   ["gtquest"] = 10876,
5264   ["LessSlantEqual"] = 10877,
5265   ["NotLessSlantEqual"] = {10877, 824},
5266   ["leqslant"] = 10877,
5267   ["les"] = 10877,
5268   ["nleqslant"] = {10877, 824},
5269   ["nles"] = {10877, 824},
5270   ["GreaterSlantEqual"] = 10878,
5271   ["NotGreaterSlantEqual"] = {10878, 824},
5272   ["geqslant"] = 10878,
5273   ["ges"] = 10878,
5274   ["ngeqslant"] = {10878, 824},
5275   ["nges"] = {10878, 824},
5276   ["lesdot"] = 10879,
5277   ["gesdot"] = 10880,
5278   ["lesdoto"] = 10881,
5279   ["gesdoto"] = 10882,
5280   ["lesdotor"] = 10883,
5281   ["gesdotol"] = 10884,
5282   ["lap"] = 10885,
5283   ["lessapprox"] = 10885,
5284   ["gap"] = 10886,
5285   ["gtrapprox"] = 10886,
5286   ["lne"] = 10887,
5287   ["lneq"] = 10887,
5288   ["gne"] = 10888,
5289   ["gneq"] = 10888,
5290   ["lnap"] = 10889,
5291   ["lnapprox"] = 10889,
5292   ["gnap"] = 10890,
5293   ["gnapprox"] = 10890,
5294   ["lEg"] = 10891,
5295   ["lesseqqgtr"] = 10891,
5296   ["gEl"] = 10892,
5297   ["gtreqqless"] = 10892,
5298   ["lsime"] = 10893,
5299   ["gsime"] = 10894,
5300   ["lsimg"] = 10895,
5301   ["gsiml"] = 10896,
5302   ["lgE"] = 10897,
5303   ["glE"] = 10898,
5304   ["lesges"] = 10899,
5305   ["gesles"] = 10900,
5306   ["els"] = 10901,
5307   ["eqslantless"] = 10901,
5308   ["egs"] = 10902,
5309   ["eqslantgtr"] = 10902,
```

```
5310    ["elsdot"] = 10903,
5311    ["egsdot"] = 10904,
5312    ["el"] = 10905,
5313    ["eg"] = 10906,
5314    ["siml"] = 10909,
5315    ["simg"] = 10910,
5316    ["simlE"] = 10911,
5317    ["simgE"] = 10912,
5318    ["LessLess"] = 10913,
5319    ["NotNestedLessLess"] = {10913, 824},
5320    ["GreaterGreater"] = 10914,
5321    ["NotNestedGreaterGreater"] = {10914, 824},
5322    ["glj"] = 10916,
5323    ["gla"] = 10917,
5324    ["ltcc"] = 10918,
5325    ["gtcc"] = 10919,
5326    ["lescc"] = 10920,
5327    ["gescc"] = 10921,
5328    ["smt"] = 10922,
5329    ["lat"] = 10923,
5330    ["smte"] = 10924,
5331    ["smtes"] = {10924, 65024},
5332    ["late"] = 10925,
5333    ["lates"] = {10925, 65024},
5334    ["bumpE"] = 10926,
5335    ["NotPrecedesEqual"] = {10927, 824},
5336    ["PrecedesEqual"] = 10927,
5337    ["npre"] = {10927, 824},
5338    ["npreceq"] = {10927, 824},
5339    ["pre"] = 10927,
5340    ["preceq"] = 10927,
5341    ["NotSucceedsEqual"] = {10928, 824},
5342    ["SucceedsEqual"] = 10928,
5343    ["nsce"] = {10928, 824},
5344    ["nsucceq"] = {10928, 824},
5345    ["sce"] = 10928,
5346    ["succeq"] = 10928,
5347    ["prE"] = 10931,
5348    ["scE"] = 10932,
5349    ["precneqq"] = 10933,
5350    ["prnE"] = 10933,
5351    ["scnE"] = 10934,
5352    ["succneqq"] = 10934,
5353    ["prap"] = 10935,
5354    ["precapprox"] = 10935,
5355    ["scap"] = 10936,
5356    ["succapprox"] = 10936,
```

```
5357   ["precnapprox"] = 10937,
5358   ["prnap"] = 10937,
5359   ["scnap"] = 10938,
5360   ["succnapprox"] = 10938,
5361   ["Pr"] = 10939,
5362   ["Sc"] = 10940,
5363   ["subdot"] = 10941,
5364   ["supdot"] = 10942,
5365   ["subplus"] = 10943,
5366   ["supplus"] = 10944,
5367   ["submult"] = 10945,
5368   ["supmult"] = 10946,
5369   ["subedot"] = 10947,
5370   ["supedot"] = 10948,
5371   ["nsubE"] = {10949, 824},
5372   ["nsubseteqq"] = {10949, 824},
5373   ["subE"] = 10949,
5374   ["subseteqq"] = 10949,
5375   ["nsupE"] = {10950, 824},
5376   ["nsupseteqq"] = {10950, 824},
5377   ["supE"] = 10950,
5378   ["supseteqq"] = 10950,
5379   ["subsim"] = 10951,
5380   ["supsim"] = 10952,
5381   ["subnE"] = 10955,
5382   ["subsetneqq"] = 10955,
5383   ["varsubsetneqq"] = {10955, 65024},
5384   ["vsubnE"] = {10955, 65024},
5385   ["supnE"] = 10956,
5386   ["supsetneqq"] = 10956,
5387   ["varsupsetneqq"] = {10956, 65024},
5388   ["vsupnE"] = {10956, 65024},
5389   ["csub"] = 10959,
5390   ["csup"] = 10960,
5391   ["csube"] = 10961,
5392   ["csupe"] = 10962,
5393   ["subsup"] = 10963,
5394   ["supsub"] = 10964,
5395   ["subsub"] = 10965,
5396   ["supsup"] = 10966,
5397   ["suphsub"] = 10967,
5398   ["supdsub"] = 10968,
5399   ["forkv"] = 10969,
5400   ["topfork"] = 10970,
5401   ["mlcp"] = 10971,
5402   ["Dashv"] = 10980,
5403   ["DoubleLeftTee"] = 10980,
```

```
  ["Vdashl"] = 10982,
  ["Barv"] = 10983,
  ["vBar"] = 10984,
  ["vBarv"] = 10985,
  ["Vbar"] = 10987,
  ["Not"] = 10988,
  ["bNot"] = 10989,
  ["rnmid"] = 10990,
  ["cirmid"] = 10991,
  ["midcir"] = 10992,
  ["topcir"] = 10993,
  ["nhpar"] = 10994,
  ["parsim"] = 10995,
  ["nparsl"] = {11005, 8421},
  ["parsl"] = 11005,
  ["fflig"] = 64256,
  ["filig"] = 64257,
  ["fllig"] = 64258,
  ["ffilig"] = 64259,
  ["ffllig"] = 64260,
  ["Ascr"] = 119964,
  ["Cscr"] = 119966,
  ["Dscr"] = 119967,
  ["Gscr"] = 119970,
  ["Jscr"] = 119973,
  ["Kscr"] = 119974,
  ["Nscr"] = 119977,
  ["Oscr"] = 119978,
  ["Pscr"] = 119979,
  ["Qscr"] = 119980,
  ["Sscr"] = 119982,
  ["Tscr"] = 119983,
  ["Uscr"] = 119984,
  ["Vscr"] = 119985,
  ["Wscr"] = 119986,
  ["Xscr"] = 119987,
  ["Yscr"] = 119988,
  ["Zscr"] = 119989,
  ["ascr"] = 119990,
  ["bscr"] = 119991,
  ["cscr"] = 119992,
  ["dscr"] = 119993,
  ["fscr"] = 119995,
  ["hscr"] = 119997,
  ["iscr"] = 119998,
  ["jscr"] = 119999,
  ["kscr"] = 120000,
```

```
  ["lscr"] = 120001,
  ["mscr"] = 120002,
  ["nscr"] = 120003,
  ["pscr"] = 120005,
  ["qscr"] = 120006,
  ["rscr"] = 120007,
  ["sscr"] = 120008,
  ["tscr"] = 120009,
  ["uscr"] = 120010,
  ["vscr"] = 120011,
  ["wscr"] = 120012,
  ["xscr"] = 120013,
  ["yscr"] = 120014,
  ["zscr"] = 120015,
  ["Afr"] = 120068,
  ["Bfr"] = 120069,
  ["Dfr"] = 120071,
  ["Efr"] = 120072,
  ["Ffr"] = 120073,
  ["Gfr"] = 120074,
  ["Jfr"] = 120077,
  ["Kfr"] = 120078,
  ["Lfr"] = 120079,
  ["Mfr"] = 120080,
  ["Nfr"] = 120081,
  ["Ofr"] = 120082,
  ["Pfr"] = 120083,
  ["Qfr"] = 120084,
  ["Sfr"] = 120086,
  ["Tfr"] = 120087,
  ["Ufr"] = 120088,
  ["Vfr"] = 120089,
  ["Wfr"] = 120090,
  ["Xfr"] = 120091,
  ["Yfr"] = 120092,
  ["afr"] = 120094,
  ["bfr"] = 120095,
  ["cfr"] = 120096,
  ["dfr"] = 120097,
  ["efr"] = 120098,
  ["ffr"] = 120099,
  ["gfr"] = 120100,
  ["hfr"] = 120101,
  ["ifr"] = 120102,
  ["jfr"] = 120103,
  ["kfr"] = 120104,
  ["lfr"] = 120105,
```

```
  ["mfr"] = 120106,
  ["nfr"] = 120107,
  ["ofr"] = 120108,
  ["pfr"] = 120109,
  ["qfr"] = 120110,
  ["rfr"] = 120111,
  ["sfr"] = 120112,
  ["tfr"] = 120113,
  ["ufr"] = 120114,
  ["vfr"] = 120115,
  ["wfr"] = 120116,
  ["xfr"] = 120117,
  ["yfr"] = 120118,
  ["zfr"] = 120119,
  ["Aopf"] = 120120,
  ["Bopf"] = 120121,
  ["Dopf"] = 120123,
  ["Eopf"] = 120124,
  ["Fopf"] = 120125,
  ["Gopf"] = 120126,
  ["Iopf"] = 120128,
  ["Jopf"] = 120129,
  ["Kopf"] = 120130,
  ["Lopf"] = 120131,
  ["Mopf"] = 120132,
  ["Oopf"] = 120134,
  ["Sopf"] = 120138,
  ["Topf"] = 120139,
  ["Uopf"] = 120140,
  ["Vopf"] = 120141,
  ["Wopf"] = 120142,
  ["Xopf"] = 120143,
  ["Yopf"] = 120144,
  ["aopf"] = 120146,
  ["bopf"] = 120147,
  ["copf"] = 120148,
  ["dopf"] = 120149,
  ["eopf"] = 120150,
  ["fopf"] = 120151,
  ["gopf"] = 120152,
  ["hopf"] = 120153,
  ["iopf"] = 120154,
  ["jopf"] = 120155,
  ["kopf"] = 120156,
  ["lopf"] = 120157,
  ["mopf"] = 120158,
  ["nopf"] = 120159,
```

```
  ["oopf"] = 120160,
  ["popf"] = 120161,
  ["qopf"] = 120162,
  ["ropf"] = 120163,
  ["sopf"] = 120164,
  ["topf"] = 120165,
  ["uopf"] = 120166,
  ["vopf"] = 120167,
  ["wopf"] = 120168,
  ["xopf"] = 120169,
  ["yopf"] = 120170,
  ["zopf"] = 120171,
}
```

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

```
function entities.dec_entity(s)
  local n = tonumber(s)
  if n == nil then
    return "&#" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.hex_entity(s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#x" .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```
function entities.hex_entity_with_x_char(x, s)
  local n = tonumber("0x"..s)
  if n == nil then
    return "&#" .. x .. s .. ";"  -- fallback for unknown entities
  end
  return unicode.utf8.char(n)
end
```

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

```
function entities.char_entity(s)
```

```
5580   local code_points = character_entities[s]
5581   if code_points == nil then
5582     return "&" .. s .. ";"
5583   end
5584   if type(code_points) ~= 'table' then
5585     code_points = {code_points}
5586   end
5587   local char_table = {}
5588     for _, code_point in ipairs(code_points) do
5589       table.insert(char_table, unicode.utf8.char(code_point))
5590     end
5591   return table.concat(char_table)
5592 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.

```
5593 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.

```
5594 function M.writer.new(options)
5595   local self = {}
```

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

```
5596   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.

```
5597   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.

```
5598  local slice_specifiers = {}
5599  for specifier in options.slice:gmatch("[^%s]+") do
5600    table.insert(slice_specifiers, specifier)
5601  end
5602
5603  if #slice_specifiers == 2 then
5604    self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5605    local slice_begin_type = self.slice_begin:sub(1, 1)
5606    if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5607      self.slice_begin = "^" .. self.slice_begin
5608    end
5609    local slice_end_type = self.slice_end:sub(1, 1)
5610    if slice_end_type ~= "^" and slice_end_type ~= "$" then
5611      self.slice_end = "$" .. self.slice_end
5612    end
5613  elseif #slice_specifiers == 1 then
5614    self.slice_begin = "^" .. slice_specifiers[1]
5615    self.slice_end = "$" .. slice_specifiers[1]
5616  end
5617
5618  self.slice_begin_type = self.slice_begin:sub(1, 1)
5619  self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5620  self.slice_end_type = self.slice_end:sub(1, 1)
5621  self.slice_end_identifier = self.slice_end:sub(2) or ""
5622
5623  if self.slice_begin == "^" and self.slice_end ~= "^" then
5624    self.is_writing = true
5625  else
5626    self.is_writing = false
5627  end
```

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

```
5628  self.suffix = ".tex"
```

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

```
5629  self.space = " "
```

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

```
5630  self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5631  function self.plain(s)
5632    return s
5633  end
```

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

```
function self.paragraph(s)
  if not self.is_writing then return "" end
  return s
end
```

Define `writer->pack` as a function that will take the filename `name` of the output file prepared by the reader and transform it to the output format.

```
function self.pack(name)
  return [[\input{]] .. name .. [[}\relax]]
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{}",
}
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 writer->escape_typographic_text, writer->escape_programmatic_text, and writer->escape_minimal 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->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
```

```
5725    local buf = {}
5726    if attributes ~= nil then
5727      table.insert(buf,
5728                   "\\markdownRendererCodeSpanAttributeContextBegin\n")
5729      table.insert(buf, self.attributes(attributes))
5730    end
5731    table.insert(buf,
5732                 {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
5733    if attributes ~= nil then
5734      table.insert(buf,
5735                   "\\markdownRendererCodeSpanAttributeContextEnd{}")
5736    end
5737    return buf
5738  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.

```
5739  function self.link(lab, src, tit, attributes)
5740    if self.flatten_inlines then return lab end
5741    local buf = {}
5742    if attributes ~= nil then
5743      table.insert(buf,
5744                   "\\markdownRendererLinkAttributeContextBegin\n")
5745      table.insert(buf, self.attributes(attributes))
5746    end
5747    table.insert(buf, {"\\markdownRendererLink{",lab,"}",
5748                       "{",self.escape(src),"}",
5749                       "{",self.uri(src),"}",
5750                       "{",self.string(tit or ""),"}"})
5751    if attributes ~= nil then
5752      table.insert(buf,
5753                   "\\markdownRendererLinkAttributeContextEnd{}")
5754    end
5755    return buf
5756  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.

```
5757  function self.image(lab, src, tit, attributes)
5758    if self.flatten_inlines then return lab end
5759    local buf = {}
5760    if attributes ~= nil then
5761      table.insert(buf,
5762                   "\\markdownRendererImageAttributeContextBegin\n")
5763      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
    local buffer = {}
    local num = startnum
```

```
5800      for _,item in ipairs(items) do
5801        if item ~= "" then
5802          buffer[#buffer + 1] = self.ordereditem(item,num)
5803        end
5804        if num ~= nil and item ~= "" then
5805          num = num + 1
5806        end
5807      end
5808      local contents = util.intersperse(buffer,"\n")
5809      if tight and options.tightLists then
5810        return {"\\markdownRendererOlBeginTight\n",contents,
5811                "\n\\markdownRendererOlEndTight "}
5812      else
5813        return {"\\markdownRendererOlBegin\n",contents,
5814                "\n\\markdownRendererOlEnd "}
5815      end
5816    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.

```
5817    function self.ordereditem(s,num)
5818      if num ~= nil then
5819        return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
5820                "\\markdownRendererOlItemEnd "}
5821      else
5822        return {"\\markdownRendererOlItem ",s,
5823                "\\markdownRendererOlItemEnd "}
5824      end
5825    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.

```
5826    function self.inline_html_comment(contents)
5827      if self.flatten_inlines then return contents end
5828      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
5829    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.

```
5830    function self.inline_html_tag(contents)
5831      if self.flatten_inlines then return contents end
5832      return {"\\markdownRendererInlineHtmlTag{",self.string(contents),"}"}
5833    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.

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

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

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

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

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

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

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

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

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

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

```
function self.verbatim(s)
  if not self.is_writing then return "" end
  s = s:gsub("\n$", "")
```

```
5864      local name = util.cache_verbatim(options.cacheDir, s)
5865      return {"\\markdownRendererInputVerbatim{",name,"}"}
5866    end
```

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

```
5867    function self.document(d)
5868      local buf = {"\\markdownRendererDocumentBegin\n", d}
5869
5870      -- pop all attributes
5871      table.insert(buf, self.pop_attributes())
5872
5873      table.insert(buf, "\\markdownRendererDocumentEnd")
5874
5875      return buf
5876    end
```

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

```
5877    local seen_identifiers = {}
5878    local key_value_regex = "([^= ]+)%s*=%s*(.*)"
5879    local function normalize_attributes(attributes, auto_identifiers)
5880      -- normalize attributes
5881      local normalized_attributes = {}
5882      local has_explicit_identifiers = false
5883      local key, value
5884      for _, attribute in ipairs(attributes or {}) do
5885        if attribute:sub(1, 1) == "#" then
5886          table.insert(normalized_attributes, attribute)
5887          has_explicit_identifiers = true
5888          seen_identifiers[attribute:sub(2)] = true
5889        elseif attribute:sub(1, 1) == "." then
5890          table.insert(normalized_attributes, attribute)
5891        else
5892          key, value = attribute:match(key_value_regex)
5893          if key:lower() == "id" then
5894            table.insert(normalized_attributes, "#" .. value)
5895          elseif key:lower() == "class" then
5896            local classes = {}
5897            for class in value:gmatch("%S+") do
5898              table.insert(classes, class)
5899            end
5900            table.sort(classes)
5901            for _, class in ipairs(classes) do
5902              table.insert(normalized_attributes, "." .. class)
5903            end
5904          else
5905            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
    return buf
```

```
5991  end
5992
5993  local function tear_down_attributes()
5994    local buf = {}
5995    for i = #self.active_attributes, 1, -1 do
5996      local end_output = self.active_attributes[i][4]
5997      if end_output ~= nil then
5998        table.insert(buf, end_output)
5999      end
6000    end
6001    return buf
6002  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.

```
6003  function self.push_attributes(attribute_type, attributes,
6004                                start_output, end_output)
6005    local attribute_type_level = self.attribute_type_levels[attribute_type]
6006    self.attribute_type_levels[attribute_type] = attribute_type_level + 1
6007
6008    -- index attributes in a hash table for easy lookup
6009    attributes = attributes or {}
6010    for i = 1, #attributes do
6011      attributes[attributes[i]] = true
6012    end
6013
6014    local buf = {}
6015    -- handle slicing
6016    if attributes["#" .. self.slice_end_identifier] ~= nil and
6017       self.slice_end_type == "^" then
6018      if self.is_writing then
6019        table.insert(buf, self.undosep())
6020        table.insert(buf, tear_down_attributes())
6021      end
6022      self.is_writing = false
6023    end
6024    if attributes["#" .. self.slice_begin_identifier] ~= nil and
6025       self.slice_begin_type == "^" then
6026      table.insert(buf, apply_attributes())
6027      self.is_writing = true
6028    end
6029    if self.is_writing and start_output ~= nil then
6030      table.insert(buf, start_output)
6031    end
```

```
6032    table.insert(self.active_attributes,
6033                 {attribute_type, attributes,
6034                  start_output, end_output})
6035    return buf
6036  end
6037
```

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).

```
6038  function self.pop_attributes(attribute_type)
6039    local buf = {}
6040    -- pop attributes until we find attributes of correct type
6041    -- or until no attributes remain
6042    local current_attribute_type = false
6043    while current_attribute_type ~= attribute_type and
6044          #self.active_attributes > 0 do
6045      local attributes, _, end_output
6046      current_attribute_type, attributes, _, end_output = table.unpack(
6047        self.active_attributes[#self.active_attributes])
6048      local attribute_type_level = self.attribute_type_levels[current_attribute_type]
6049      self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
6050      if self.is_writing and end_output ~= nil then
6051        table.insert(buf, end_output)
6052      end
6053      table.remove(self.active_attributes, #self.active_attributes)
6054      -- handle slicing
6055      if attributes["#" .. self.slice_end_identifier] ~= nil
6056         and self.slice_end_type == "$" then
6057        if self.is_writing then
6058          table.insert(buf, self.undosep())
6059          table.insert(buf, tear_down_attributes())
6060        end
6061        self.is_writing = false
6062      end
6063      if attributes["#" .. self.slice_begin_identifier] ~= nil and
6064         self.slice_begin_type == "$" then
6065        self.is_writing = true
6066        table.insert(buf, apply_attributes())
6067      end
6068    end
6069    return buf
6070  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

  for _, code in utf8.codes(uni_algos.normalize.NFC(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 per
    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

    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
```

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

```
  local function create_gfm_auto_identifier(s)
    local buffer = {}
    local prev_space = false
    local letter_found = false

    for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
      local char = utf8.char(code)

      -- Remove everything up to the first non-space.
      if not letter_found then
        local is_letter = unicode.utf8.match(char, "%S")
        if is_letter then
          letter_found = true
        else
          goto continue
        end
      end

      -- Remove all non-alphanumeric characters, except underscores and hyphens.
      if not unicode.utf8.match(char, "[%w_%-%s]") then
        prev_space = false
        goto continue
      end

      -- Replace all spaces and newlines with hyphens.
      if unicode.utf8.match(char, "[%s\n]") then
        char = "-"
        if prev_space then
          goto continue
        else
          prev_space = true
        end
      else
        -- Convert all alphabetic characters to lowercase.
        char = unicode.utf8.lower(char)
        prev_space = false
      end

      table.insert(buffer, char)

      ::continue::
```

```
    end

    if prev_space then
      table.remove(buffer)
    end

    local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
    return identifier
  end
```

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

```
  self.secbegin_text = "\\markdownRendererSectionBegin\n"
  self.secend_text = "\n\\markdownRendererSectionEnd "
  function self.heading(s, level, attributes)
    local buf = {}
    local flat_text, inlines = table.unpack(s)

    -- 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
      table.insert(buf, self.pop_attributes("heading"))
    end

    -- construct attributes for the new section
    local auto_identifiers = {}
    if self.options.autoIdentifiers then
      table.insert(auto_identifiers, create_auto_identifier(flat_text))
    end
    if self.options.gfmAutoIdentifiers then
      table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
    end
    local normalized_attributes = normalize_attributes(attributes, auto_identifiers)

    -- push attributes for the new section
    local start_output = {}
    local end_output = {}
    table.insert(start_output, self.secbegin_text)
    table.insert(end_output, self.secend_text)
```

```
6205
6206      table.insert(buf, self.push_attributes("heading",
6207                                             normalized_attributes,
6208                                             start_output,
6209                                             end_output))
6210      assert(self.attribute_type_levels["heading"] == level)
6211
6212      -- render the heading and its attributes
6213      if self.is_writing and #normalized_attributes > 0 then
6214        table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
6215        table.insert(buf, self.attributes(normalized_attributes, false))
6216      end
6217
6218      local cmd
6219      level = level + options.shiftHeadings
6220      if level <= 1 then
6221        cmd = "\\markdownRendererHeadingOne"
6222      elseif level == 2 then
6223        cmd = "\\markdownRendererHeadingTwo"
6224      elseif level == 3 then
6225        cmd = "\\markdownRendererHeadingThree"
6226      elseif level == 4 then
6227        cmd = "\\markdownRendererHeadingFour"
6228      elseif level == 5 then
6229        cmd = "\\markdownRendererHeadingFive"
6230      elseif level >= 6 then
6231        cmd = "\\markdownRendererHeadingSix"
6232      else
6233        cmd = ""
6234      end
6235      if self.is_writing then
6236        table.insert(buf, {cmd, "{", inlines, "}"})
6237      end
6238
6239      if self.is_writing and #normalized_attributes > 0 then
6240        table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{}")
6241      end
6242
6243      return buf
6244    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.

```
6245    function self.get_state()
6246      return {
6247        is_writing=self.is_writing,
6248        flatten_inlines=self.flatten_inlines,
```

```
6249        active_attributes={table.unpack(self.active_attributes)},
6250      }
6251    end
```

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

```
6252    function self.set_state(s)
6253      local previous_state = self.get_state()
6254      for key, value in pairs(s) do
6255        self[key] = value
6256      end
6257      return previous_state
6258    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`.

```
6259    function self.defer_call(f)
6260      local previous_state = self.get_state()
6261      return function(...)
6262        local state = self.set_state(previous_state)
6263        local return_value = f(...)
6264        self.set_state(state)
6265        return return_value
6266      end
6267    end
6268
6269    return self
6270  end
```

3.1.4 Parsers

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

```
6271  local parsers                 = {}
```

3.1.4.1 Basic Parsers

```
6272  parsers.percent               = P("%")
6273  parsers.at                    = P("@")
6274  parsers.comma                 = P(",")
6275  parsers.asterisk              = P("*")
6276  parsers.dash                  = P("-")
6277  parsers.plus                  = P("+")
6278  parsers.underscore            = P("_")
6279  parsers.period                = P(".")
6280  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[32] 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))
```

```
parsers.punctuation            = {}
(function()
  local pathname = kpse.lookup("UnicodeData.txt")
  local file = assert(io.open(pathname, "r"),
    [[Could not open file "UnicodeData.txt"]])
  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 parsers.punctuation[#code] == nil then
        parsers.punctuation[#code] = parsers.fail
      end
      local code_parser = parsers.succeed
      for i = 1, #code do
        local byte = code:sub(i, i)
        local byte_parser = S(byte)
        code_parser = code_parser
                    * byte_parser
      end
      parsers.punctuation[#code] = parsers.punctuation[#code]
                                 + code_parser
    end
  end
  assert(file:close())
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)
```

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

```
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 function remove_indent_level(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)
```

```
6435  left_strip_length = left_strip_length or 0
6436
6437  local count = 0
6438  local tab_value = 4 - (indent) % 4
6439
6440  local code_started, minimum_found = false, false
6441  local code_start, minimum_remainder = "", ""
6442
6443  local left_total_stripped = 0
6444  local full_remainder = ""
6445
6446  if spacing ~= nil then
6447    for i = 1, #spacing do
6448      local character = spacing:sub(i, i)
6449
6450      if character == "\t" then
6451        count = count + tab_value
6452        tab_value = 4
6453      elseif character == " " then
6454        count = count + 1
6455        tab_value = 4 - (1 - tab_value) % 4
6456      end
6457
6458      if (left_strip_length ~= 0) then
6459        local possible_to_strip = math.min(count, left_strip_length)
6460        count = count - possible_to_strip
6461        left_strip_length = left_strip_length - possible_to_strip
6462        left_total_stripped = left_total_stripped + possible_to_strip
6463      else
6464        full_remainder =  full_remainder .. character
6465      end
6466
6467      if (minimum_found) then
6468        minimum_remainder = minimum_remainder .. character
6469      elseif (count >= minimum) then
6470        minimum_found = true
6471        minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6472      end
6473
6474      if (code_started) then
6475        code_start = code_start .. character
6476      elseif (count >= minimum + 4) then
6477        code_started = true
6478        code_start = code_start .. string.rep(" ", count - (minimum + 4))
6479      end
6480    end
6481  end
```

```
6482
6483   local remainder
6484   if (code_started) then
6485     remainder = code_start
6486   else
6487     remainder = string.rep(" ", count - minimum)
6488   end
6489
6490   local is_minimum = count >= minimum
6491   return {
6492     is_code = code_started,
6493     remainder = remainder,
6494     left_total_stripped = left_total_stripped,
6495     is_minimum = is_minimum,
6496     minimum_remainder = minimum_remainder,
6497     total_length = count,
6498     full_remainder = full_remainder
6499   }
6500 end
6501
```

Count the total width of all indents in the indent table `indent_table`.

```
6502 local function count_indent_tab_level(indent_table)
6503   local count = 0
6504   if not has_indents(indent_table) then
6505     return count
6506   end
6507
6508   for i=1, #indent_table.indents do
6509     count = count + indent_table.indents[i].length
6510   end
6511   return count
6512 end
6513
```

Count the total width of a delimiter `delimiter`.

```
6514 local function total_delimiter_length(delimiter)
6515   local count = 0
6516   if type(delimiter) == "string" then return #delimiter end
6517   for _, value in pairs(delimiter) do
6518     count = count + total_delimiter_length(value)
6519   end
6520   return count
6521 end
6522
```

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`.

```
6523 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6524   local last_trail = starter[1]
6525   local delimiter = starter[2]
6526   local raw_new_trail = starter[3]
6527
6528   if indent_type == "bq" and not breakable then
6529     indent_table.ignore_blockquote_blank = true
6530   end
6531
6532   if has_trail(indent_table) then
6533     local trail = indent_table.trail
6534     if trail.is_code then
6535       return false
6536     end
6537     last_trail = trail.remainder
6538   else
6539     local sp = process_starter_spacing(0, last_trail, 0, 0)
6540
6541     if sp.is_code then
6542       return false
6543     end
6544     last_trail = sp.remainder
6545   end
6546
6547   local preceding_indentation = count_indent_tab_level(indent_table) % 4
6548   local last_trail_length = #last_trail
6549   local delimiter_length = total_delimiter_length(delimiter)
6550
6551   local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6552
6553   local sp = {}
6554   if not is_blank then
6555     sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6556   end
6557
6558   local del_trail_length = sp.left_total_stripped
6559   if is_blank then
6560     del_trail_length = 1
6561   elseif not sp.is_code then
6562     del_trail_length = del_trail_length + #sp.remainder
6563   end
6564
6565   local indent_length = last_trail_length + delimiter_length + del_trail_length
6566   local new_indent_info = {name=indent_type, length=indent_length}
6567
6568   indent_table = update_indent_table(indent_table, new_indent_info, true)
6569   indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
```

```
                                                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_lin
  local new_index = i

  local preceding_indentation = 0
```

```
6607  local current_trail = {}
6608
6609  local blank_starter = left_blank_starter(indent_table)
6610
6611  if current_line_indents == nil then
6612    current_line_indents = {}
6613  end
6614
6615  for index = 1,#indent_table.indents do
6616    local value = indent_table.indents[index]
6617    local pattern = decode_pattern(value.name)
6618
6619    -- match decoded pattern
6620    local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6621    if new_indent_info == nil then
6622      local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, ne
6623      if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end
6624        return is_optional, new_index, current_trail, current_line_indents
6625      end
6626
6627      return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optiona
6628    end
6629
6630    local raw_last_trail = new_indent_info[1]
6631    local delimiter = new_indent_info[2]
6632    local raw_new_trail = new_indent_info[3]
6633    local next_index = new_indent_info[4]
6634
6635    local space_only = delimiter == ""
6636
6637    -- check previous trail
6638    if not space_only and next(current_trail) == nil then
6639      local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6640      current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.to
6641                       full_remainder=sp.full_remainder}
6642    end
6643
6644    if next(current_trail) ~= nil then
6645      if not space_only and current_trail.is_code then
6646        return is_optional, new_index, current_trail, current_line_indents
6647      end
6648      if current_trail.internal_remainder ~= nil then
6649        raw_last_trail = current_trail.internal_remainder
6650      end
6651    end
6652
6653    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 + delimi

    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, minimu

    if space_only and not sp.is_minimum then
      return is_optional or (is_blank and blank_starter <= index), new_index, current
    end

    local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_st

    -- 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=s
                     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 function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_rema
  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 function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
  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_bla
  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 function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
  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 function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
  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 function check_continuation_indentation_and_trail(s, i, indent_table, is_option
                                                        reset_rem, omit_remainder)
```

```
6818   if not has_indents(indent_table) then
6819     local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6820     local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, tra
6821     if remainder == nil then
6822       if result then
6823         return new_index
6824       end
6825       return false
6826     end
6827     if result then
6828       return new_index, remainder
6829     end
6830     return false
6831   end
6832
6833   local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
6834
6835   if passes then
6836     local spacing
6837     if current_trail == nil then
6838       local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6839       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6840       new_index = newer_index
6841       spacing = newer_spacing
6842     else
6843       spacing = current_trail.remainder
6844     end
6845     local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6846     if remainder == nil or omit_remainder then
6847       if result then
6848         return new_index
6849       end
6850       return false
6851     end
6852
6853     if is_blank and reset_rem then
6854       remainder = remove_remainder_if_blank(indent_table, remainder)
6855     end
6856     if result then
6857       return new_index, remainder
6858     end
6859     return false
6860   end
6861   return false
6862 end
6863
```

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), che

parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals

parsers.check_code_trail  = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)

parsers.check_trail_length_range  = function(min, max)
  return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
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_

parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde

```

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_

parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_

parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch

```

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")
                                                 check_continuation_indentation_and_t

parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
                                                            * Cc(false) * Cc(true) *
    code") * Cc(true),
                                                            check_continuation_indent

```

```
6900
6901 parsers.check_minimal_indent_and_any_trail = Cmt( Cb("indent_info")
6902                                                 * Cc(false) * Cc(false) * Cc("full-
     any") * Cc(true) * Cc(false),
6903                                                 check_continuation_indentation_and_tr
6904
6905 parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6906                                                       * Cc(false) * Cc(true) * Cc("fu
     any") * Cc(true) * Cc(false),
6907                                                        check_continuation_indentation
6908
6909 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
6910                                                          * Cc(false) * Cc(true) * Cc("
     any") * Cc(true) * Cc(true),
6911                                                          check_continuation_indentatio
6912
6913 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6914                                                 * Cc(true) * Cc(false) * Cc("full-
     any") * Cc(true) * Cc(false),
6915                                                 check_continuation_indentation_and_tr
6916
6917 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6918                                                       * Cc(true) * Cc(true) * Cc("ful
     any") * Cc(true) * Cc(false),
6919                                                        check_continuation_indentation
6920
```

The following patterns specify behaviour around newlines.

```
6921
6922 parsers.spnlc_noexc = parsers.optionalspace
6923                     * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
     1
6924
6925 parsers.spnlc = parsers.optionalspace
6926               * (V("EndlineNoSub"))^-1
6927
6928 parsers.spnlc_sep  = parsers.optionalspace * V("EndlineNoSub")
6929                    + parsers.spacechar^1
6930
6931 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6932
6933 %   \end{macrocode}
6934 % \begin{figure}
6935 % \hspace*{-0.1\textwidth}
6936 % \begin{minipage}{1.2\textwidth}
6937 % \centering
6938 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6939 % \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 tabs and spac
                                  + #(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.rbra
                                  - parsers.space)^0
parsers.attribute_identifier    = parsers.attribute_key_char^1
```

```
parsers.attribute_classname    = parsers.letter
                               * parsers.attribute_key_char^0
parsers.attribute_raw          = parsers.attribute_raw_char^1

parsers.attribute = (parsers.dash * Cc(".unnumbered"))
                  + C( parsers.hash
                     * parsers.attribute_identifier)
                  + C( parsers.period
                     * parsers.attribute_classname)
                  + Cs( parsers.attribute_key
                      * parsers.optionalspace * parsers.equal * parsers.optionalspace
                      * parsers.attribute_value)
parsers.attributes = parsers.lbrace
                   * parsers.optionalspace
                   * parsers.attribute
                   * (parsers.spacechar^1
                     * parsers.attribute)^0
                   * parsers.optionalspace
                   * parsers.rbrace


parsers.raw_attribute = parsers.lbrace
                      * parsers.optionalspace
                      * parsers.equal
                      * C(parsers.attribute_raw)
                      * parsers.optionalspace
                      * parsers.rbrace

-- block followed by 0 or more optionally
-- indented blocks with first line indented.
parsers.indented_blocks = function(bl)
  return Cs( bl
         * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
         * (parsers.blankline^1 + parsers.eof) )
end
```

3.1.5.2 Parsers Used for HTML Entities

```
local function repeat_between(pattern, min, max)
  return -pattern^(max + 1) * pattern^min
end

parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
                  * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
parsers.decentity = parsers.ampersand * parsers.hash
                  * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
```

```
                  * parsers.semicolon

parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
                      + parsers.decentity / entities.dec_entity
                      + parsers.tagentity / entities.char_entity
```

3.1.5.3 Parsers Used for Markdown Lists

```
parsers.bullet = function(bullet_char, interrupting)
  local allowed_end
  if interrupting then
    allowed_end = C(parsers.spacechar^1) * #parsers.linechar
  else
    allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
  end
  return parsers.check_trail
        * Ct(C(bullet_char) * Cc(""))
        * allowed_end
end

local function tickbox(interior)
  return parsers.optionalspace * parsers.lbracket
       * interior * parsers.rbracket * parsers.spacechar^1
end

parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)

```

3.1.5.4 Parsers Used for Markdown Code Spans

```
parsers.openticks   = Cg(parsers.backtick^1, "ticks")

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

parsers.closeticks  = Cmt(C(parsers.backtick^1)
                          * Cb("ticks"), captures_equal_length)

parsers.intickschar = (parsers.any - S("\n\r`"))
                    + V("NoSoftLineBreakEndline")
                    + (parsers.backtick^1 - parsers.closeticks)

local function process_inticks(s)
  s = s:gsub("\n", " ")
  s = s:gsub("^ (.*) $", "%1")
```

```
  return s
end

parsers.inticks = parsers.openticks
                * C(parsers.space^0)
                * parsers.closeticks
                + parsers.openticks
                * Cs(Cs(parsers.intickschar^0) / process_inticks)
                * parsers.closeticks

```

3.1.5.5 Parsers Used for HTML

```
-- case-insensitive match (we assume s is lowercase). must be single byte encoding
parsers.keyword_exact = function(s)
  local parser = P(0)
  for i=1,#s do
    local c = s:sub(i,i)
    local m = c .. upper(c)
    parser = parser * S(m)
  end
  return parser
end

parsers.special_block_keyword =
    parsers.keyword_exact("pre") +
    parsers.keyword_exact("script") +
    parsers.keyword_exact("style") +
    parsers.keyword_exact("textarea")

parsers.block_keyword =
    parsers.keyword_exact("address") +
    parsers.keyword_exact("article") +
    parsers.keyword_exact("aside") +
    parsers.keyword_exact("base") +
    parsers.keyword_exact("basefont") +
    parsers.keyword_exact("blockquote") +
    parsers.keyword_exact("body") +
    parsers.keyword_exact("caption") +
    parsers.keyword_exact("center") +
    parsers.keyword_exact("col") +
    parsers.keyword_exact("colgroup") +
    parsers.keyword_exact("dd") +
    parsers.keyword_exact("details") +
    parsers.keyword_exact("dialog") +
    parsers.keyword_exact("dir") +
    parsers.keyword_exact("div") +
```

```
    parsers.keyword_exact("dl") +
    parsers.keyword_exact("dt") +
    parsers.keyword_exact("fieldset") +
    parsers.keyword_exact("figcaption") +
    parsers.keyword_exact("figure") +
    parsers.keyword_exact("footer") +
    parsers.keyword_exact("form") +
    parsers.keyword_exact("frame") +
    parsers.keyword_exact("frameset") +
    parsers.keyword_exact("h1") +
    parsers.keyword_exact("h2") +
    parsers.keyword_exact("h3") +
    parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") +
    parsers.keyword_exact("h6") +
    parsers.keyword_exact("head") +
    parsers.keyword_exact("header") +
    parsers.keyword_exact("hr") +
    parsers.keyword_exact("html") +
    parsers.keyword_exact("iframe") +
    parsers.keyword_exact("legend") +
    parsers.keyword_exact("li") +
    parsers.keyword_exact("link") +
    parsers.keyword_exact("main") +
    parsers.keyword_exact("menu") +
    parsers.keyword_exact("menuitem") +
    parsers.keyword_exact("nav") +
    parsers.keyword_exact("noframes") +
    parsers.keyword_exact("ol") +
    parsers.keyword_exact("optgroup") +
    parsers.keyword_exact("option") +
    parsers.keyword_exact("p") +
    parsers.keyword_exact("param") +
    parsers.keyword_exact("section") +
    parsers.keyword_exact("source") +
    parsers.keyword_exact("summary") +
    parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") +
    parsers.keyword_exact("td") +
    parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("th") +
    parsers.keyword_exact("thead") +
    parsers.keyword_exact("title") +
    parsers.keyword_exact("tr") +
    parsers.keyword_exact("track") +
    parsers.keyword_exact("ul")

```

```
-- end conditions
parsers.html_blankline_end_condition  = parsers.linechar^0
                                      * ( parsers.newline
                                        * (parsers.check_minimal_blank_indent_and_any
                                          * #parsers.blankline
                                          + parsers.check_minimal_indent_and_any_trai
                                        * 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.undersc
                            + 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.newl
                                      - parsers.dquote - parsers.squote - parsers.bac
                                      - parsers.equal - parsers.less - parsers.more)^

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_val

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_commen
                            * (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_
                                  * 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_de
                                * parsers.html_declaration_end

parsers.html_instruction_start = P("<?")

parsers.html_instruction_end = P("?>")

parsers.html_instruction  = Cs( parsers.html_instruction_start
                              * parsers.html_until_end(parsers.html_instruction_end))

parsers.html_inline_instruction = parsers.html_instruction_start
                                * Cs(V("NoSoftLineBreakEndline") + parsers.any
                                    - parsers.nested_breaking_blank - parsers.html_in
                                * parsers.html_instruction_end

parsers.html_blankline  = parsers.newline
                        * parsers.optionalspace
                        * parsers.newline

parsers.html_tag_start = parsers.less
```

```
7348
7349 parsers.html_tag_closing_start  = parsers.less
7350                                * parsers.slash
7351
7352 parsers.html_tag_end  = parsers.html_spnl
7353                       * parsers.more
7354
7355 parsers.html_empty_tag_end  = parsers.html_spnl
7356                             * parsers.slash
7357                             * parsers.more
7358
7359 -- opening tags
7360 parsers.html_any_open_inline_tag  = parsers.html_tag_start
7361                                   * parsers.keyword
7362                                   * parsers.html_attribute^0
7363                                   * parsers.html_tag_end
7364
7365 parsers.html_any_open_tag = parsers.html_tag_start
7366                           * parsers.keyword
7367                           * parsers.html_non_newline_attribute^0
7368                           * parsers.html_tag_end
7369
7370 parsers.html_open_tag = parsers.html_tag_start
7371                       * parsers.block_keyword
7372                       * parsers.html_attribute^0
7373                       * parsers.html_tag_end
7374
7375 parsers.html_open_special_tag = parsers.html_tag_start
7376                               * parsers.special_block_keyword
7377                               * parsers.html_attribute^0
7378                               * parsers.html_tag_end
7379
7380 -- incomplete tags
7381 parsers.incomplete_tag_following  = parsers.spacechar
7382                                   + parsers.more
7383                                   + parsers.slash * parsers.more
7384                                   + #(parsers.newline + parsers.eof)
7385
7386 parsers.incomplete_special_tag_following  = parsers.spacechar
7387                                           + parsers.more
7388                                           + #(parsers.newline + parsers.eof)
7389
7390 parsers.html_incomplete_open_tag  = parsers.html_tag_start
7391                                   * parsers.block_keyword
7392                                   * parsers.incomplete_tag_following
7393
7394 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_ta
                                                   * #(parsers.optionalspace
                                                      * (parsers.newline + parsers.e

parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_b
                                            * 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_speci

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 + parser

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_
                                  * parsers.html_comment_end

parsers.html_inline_tags  = parsers.html_inline_comment_full
                          + parsers.html_any_empty_inline_tag
```

```
                            + parsers.html_inline_instruction
                            + parsers.html_inline_cdatasection
                            + parsers.html_inline_declaration
                            + parsers.html_any_open_inline_tag
                            + parsers.html_any_close_tag

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```
parsers.urlchar = parsers.anyescaped
                - parsers.newline
                - parsers.more

parsers.auto_link_scheme_part = parsers.alphanumeric
                              + parsers.plus
                              + parsers.period
                              + parsers.dash

parsers.auto_link_scheme  = parsers.letter
                          * parsers.auto_link_scheme_part
                          * parsers.auto_link_scheme_part^-30

parsers.absolute_uri  = parsers.auto_link_scheme * parsers.colon
                      * (parsers.any - parsers.spacing - parsers.less - parsers.more)

parsers.printable_characters = S(".!#$%&'*+/=?^_`{|}~-")

parsers.email_address_local_part_char = parsers.alphanumeric
                                      + parsers.printable_characters

parsers.email_address_local_part = parsers.email_address_local_part_char^1

parsers.email_address_dns_label = parsers.alphanumeric
                                * (parsers.alphanumeric + parsers.dash)^-
62
                                * B(parsers.alphanumeric)

parsers.email_address_domain  = parsers.email_address_dns_label
                              * (parsers.period * parsers.email_address_dns_label)^0

parsers.email_address = parsers.email_address_local_part
                      * parsers.at
                      * parsers.email_address_domain

parsers.auto_link_url = parsers.less
                      * C(parsers.absolute_uri)
                      * parsers.more
```

```
7531
7532 parsers.auto_link_email = parsers.less
7533                         * C(parsers.email_address)
7534                         * parsers.more
7535
7536 parsers.auto_link_relative_reference = parsers.less
7537                                      * C(parsers.urlchar^1)
7538                                      * parsers.more
7539
7540 parsers.autolink  = parsers.auto_link_url
7541                   + parsers.auto_link_email
7542
7543 -- content in balanced brackets, parentheses, or quotes:
7544 parsers.bracketed   = P{ parsers.lbracket
7545                        * (( parsers.backslash / "" * parsers.rbracket
7546                           + parsers.any - (parsers.lbracket
7547                                          + parsers.rbracket
7548                                          + parsers.blankline^2)
7549                           ) + V(1))^0
7550                        * parsers.rbracket }
7551
7552 parsers.inparens    = P{ parsers.lparent
7553                        * ((parsers.anyescaped - (parsers.lparent
7554                                                 + parsers.rparent
7555                                                 + parsers.blankline^2)
7556                           ) + V(1))^0
7557                        * parsers.rparent }
7558
7559 parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
7560                        * ((parsers.anyescaped - (parsers.squote
7561                                                 + parsers.blankline^2)
7562                           ) + V(1))^0
7563                        * parsers.squote }
7564
7565 parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
7566                        * ((parsers.anyescaped - (parsers.dquote
7567                                                 + parsers.blankline^2)
7568                           ) + V(1))^0
7569                        * parsers.dquote }
7570
7571 parsers.link_text  = parsers.lbracket
7572                    * Cs((parsers.alphanumeric^1
7573                        + parsers.bracketed
7574                        + parsers.inticks
7575                        + parsers.autolink
7576                        + V("InlineHtml")
7577                        + ( parsers.backslash * parsers.backslash)
```

```
                        + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
                          + V("NoSoftLineBreakSpace")
                          + V("NoSoftLineBreakEndline")
                          + (parsers.any
                            - (parsers.newline + parsers.lbracket + parsers.rbracket
                    * parsers.rbracket

parsers.link_label  = parsers.lbracket
                    * -#(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.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 - p
```

```
                    * parsers.squote

parsers.title_d     = parsers.dquote
                    * Cs((parsers.html_entities
                        + V("NoSoftLineBreakSpace")
                        + V("NoSoftLineBreakEndline")
                        + (parsers.anyescaped - parsers.newline - parsers.dquote - p
                    * parsers.dquote

parsers.title_p     = parsers.lparent
                    * Cs((parsers.html_entities
                        + V("NoSoftLineBreakSpace")
                        + V("NoSoftLineBreakEndline")
                        + (parsers.anyescaped - parsers.newline - parsers.lparent -
                          - 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 * p
                                * parsers.spnlc * parsers.url
                                * ( parsers.spnlc_sep * parsers.title * parsers.only_
                                  + 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))
```

```
7664 end
7665
7666 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7667                              + parsers.lineof(parsers.dash)
7668                              + parsers.lineof(parsers.underscore)
```

3.1.5.10 Headings

```
7669 -- parse Atx heading start and return level
7670 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
7671                       * -parsers.hash / length
7672
7673 -- parse setext header ending and return level
7674 parsers.heading_level = parsers.nonindentspace * parsers.equal^1 * parsers.optionalsp
7675                       + parsers.nonindentspace * parsers.dash^1 * parsers.optionalspa
7676
7677 local function strip_atx_end(s)
7678   return s:gsub("%s+#*%s*\n$","")
7679 end
7680
7681 parsers.atx_heading = parsers.check_trail_no_rem
7682                     * Cg(parsers.heading_start, "level")
7683                     * (C( parsers.optionalspace
7684                         * parsers.hash^0
7685                         * parsers.optionalspace
7686                         * parsers.newline)
7687                       + parsers.spacechar^1
7688                       * 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*⟩.

```
7689 M.reader = {}
7690 function M.reader.new(writer, options)
7691   local self = {}
```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```
7692  self.writer = writer
7693  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.

```
7694  self.parsers = {}
7695  (function(parsers)
7696    setmetatable(self.parsers, {
7697      __index = function (_, key)
7698        return parsers[key]
7699      end
7700    })
7701  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.

```
7702  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.

```
7703  function self.normalize_tag(tag)
7704    tag = util.rope_to_string(tag)
7705    tag = tag:gsub("[ \n\r\t]+", " ")
7706    tag = tag:gsub("^ ", ""):gsub(" $", "")
7707    tag = uni_algos.case.casefold(tag, true, false)
7708    return tag
7709  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.

```
7710  local function iterlines(s, f)
7711    local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7712    return util.rope_to_string(rope)
7713  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.

```
7714  if options.preserveTabs then
7715    self.expandtabs = function(s) return s end
7716  else
7717    self.expandtabs = function(s)
7718                        if s:find("\t") then
```

```
7719                          return iterlines(s, util.expand_tabs_in_line)
7720                        else
7721                          return s
7722                        end
7723                      end
7724  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.

```
7725  self.parser_functions = {}
7726  self.create_parser = function(name, grammar, toplevel)
7727    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.

```
7728      if toplevel and options.stripIndent then
7729          local min_prefix_length, min_prefix = nil, ''
7730          str = iterlines(str, function(line)
7731              if lpeg.match(parsers.nonemptyline, line) == nil then
7732                  return line
7733              end
7734              line = util.expand_tabs_in_line(line)
7735              local prefix = lpeg.match(C(parsers.optionalspace), line)
7736              local prefix_length = #prefix
7737              local is_shorter = min_prefix_length == nil
7738              is_shorter = is_shorter or prefix_length < min_prefix_length
7739              if is_shorter then
7740                  min_prefix_length, min_prefix = prefix_length, prefix
7741              end
7742              return line
7743          end)
7744          str = str:gsub('^' .. min_prefix, '')
7745      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.

```
7746      if toplevel and (options.texComments or options.hybrid) then
7747        str = lpeg.match(Ct(parsers.commented_line^1), str)
7748        str = util.rope_to_string(str)
```

```
7749        end
7750        local res = lpeg.match(grammar(), str)
7751        if res == nil then
7752          error(format("%s failed on:\n%s", name, str:sub(1,20)))
7753        else
7754          return res
7755        end
7756      end
7757    end
7758
7759    self.create_parser("parse_blocks",
7760                       function()
7761                         return parsers.blocks
7762                       end, true)
7763
7764    self.create_parser("parse_blocks_nested",
7765                       function()
7766                         return parsers.blocks_nested
7767                       end, false)
7768
7769    self.create_parser("parse_inlines",
7770                       function()
7771                         return parsers.inlines
7772                       end, false)
7773
7774    self.create_parser("parse_inlines_no_inline_note",
7775                       function()
7776                         return parsers.inlines_no_inline_note
7777                       end, false)
7778
7779    self.create_parser("parse_inlines_no_html",
7780                       function()
7781                         return parsers.inlines_no_html
7782                       end, false)
7783
7784    self.create_parser("parse_inlines_nbsp",
7785                       function()
7786                         return parsers.inlines_nbsp
7787                       end, false)
7788    self.create_parser("parse_inlines_no_link_or_emphasis",
7789                      function()
7790                        return parsers.inlines_no_link_or_emphasis
7791                      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.blan

parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")

parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("B

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_
                                                  - parsers.minimally_indented_blan

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 * (pa

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_
                                                - parsers.minimally_indented_blankl

```

The following pattern parses the properly indented content that follows the initial container start.

```

parsers.separator_loop = function(separated_block, paragraph, block_separator, para
  return  separated_block
        + block_separator
          * paragraph
          * separated_block
        + paragraph_separator
        * paragraph
end

parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
  return {
    block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
    par = parsers.separator_loop(separated_block, paragraph, block_separator, parag
```

```
7836    }
7837  end
7838
7839  parsers.block_sep_group = function(blank)
7840    return  blank^0 * parsers.eof
7841          + ( blank^2 / writer.paragraphsep
7842            + blank^0 / writer.interblocksep
7843            )
7844  end
7845
7846  parsers.par_sep_group = function(blank)
7847    return  blank^0 * parsers.eof
7848          + blank^0 / writer.paragraphsep
7849  end
7850
7851  parsers.sep_group_no_output = function(blank)
7852    return  blank^0 * parsers.eof
7853          + blank^0
7854  end
7855
7856  parsers.content_blank = parsers.minimally_indented_blankline
7857
7858  parsers.ref_or_block_separated  = parsers.sep_group_no_output(parsers.content_blank
7859                                  * ( parsers.minimally_indented_ref
7860                                    - parsers.content_blank)
7861                                  + parsers.block_sep_group(parsers.content_blank)
7862                                  * ( parsers.minimally_indented_block
7863                                    - parsers.content_blank)
7864
7865  parsers.loop_body_pair  =
7866    parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7867                                  parsers.minimally_indented_par_or_plain_no_blank,
7868                                  parsers.block_sep_group(parsers.content_blank),
7869                                  parsers.par_sep_group(parsers.content_blank))
7870
7871  parsers.content_loop  = ( V("Block")
7872                          * parsers.loop_body_pair.block^0
7873                          + (V("Paragraph") + V("Plain"))
7874                          * parsers.ref_or_block_separated
7875                          * parsers.loop_body_pair.block^0
7876                          + (V("Paragraph") + V("Plain"))
7877                          * parsers.loop_body_pair.par^0)
7878                        * parsers.content_blank^0
7879
7880  parsers.indented_content = function()
7881    return  Ct( (V("Reference") + (parsers.blankline / ""))
7882              * 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)
  return  Cg(Cmt( Cb("indent_info")
                * Ct(pattern)
                * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
                * Cc(name)
                * Cc(breakable),
              process_starter_indent), "indent_info")
end

```

3.1.6.4 Parsers Used for Markdown Lists (local)

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

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

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

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

```

3.1.6.5 Parsers Used for Blockquotes (local)

```
parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.spaced

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

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

3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

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

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
local function collect_emphasis_content(t, opening_index, closing_index)
  local content = {}

  local content_part = {}
  for i = opening_index, closing_index do
    local value = t[i]

    if value.rendered ~= nil then
      content[#content + 1] = parse_content_part(content_part)
      content_part = {}
      content[#content + 1] = value.rendered
      value.rendered = nil
    else
      if value.type == "delimiter" and value.element == "emphasis" then
        if value.is_active then
          content_part[#content_part + 1] = string.rep(value.character, value.curre
        end
      else
        content_part[#content_part + 1] = value.content
      end
```

```
7965          value.content = ''
7966          value.is_active = false
7967        end
7968      end
7969
7970      if next(content_part) ~= nil then
7971        content[#content + 1] = parse_content_part(content_part)
7972      end
7973
7974      return content
7975    end
7976
```

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

```
7977    local function fill_emph(t, opening_index, closing_index)
7978      local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7979      t[opening_index + 1].is_active = true
7980      t[opening_index + 1].rendered = writer.emphasis(content)
7981    end
7982
```

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

```
7983    local function fill_strong(t, opening_index, closing_index)
7984      local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7985      t[opening_index + 1].is_active = true
7986      t[opening_index + 1].rendered = writer.strong(content)
7987    end
7988
```

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

```
7989    local function breaks_three_rule(opening_delimiter, closing_delimiter)
7990      return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
7991        ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0
7992        (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count
7993    end
7994
```

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`.

```
7995    local function find_emphasis_opener(t, bottom_index, latest_index, character, closi
7996      for i = latest_index, bottom_index, -1 do
7997        local value = t[i]
7998        if value.is_active and
7999           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_le

      local opener_position = find_emphasis_opener(t, current_openers_bottom, current

      if (opener_position == nil) then
        openers_bottom[character][is_opening][closing_length_modulo_three + 1] = curr
        current_position = current_position + 1
        goto continue
      end

      local opening_delimiter = t[opener_position]

      local current_opening_count = opening_delimiter.current_count
      local current_closing_count = t[current_position].current_count

      if (current_opening_count >= 2) and (current_closing_count >= 2) then
        opening_delimiter.current_count = current_opening_count - 2
        t[current_position].current_count = current_closing_count - 2
        fill_strong(t, opener_position, current_position)
      else
        opening_delimiter.current_count = current_opening_count - 1
        t[current_position].current_count = current_closing_count - 1
        fill_emph(t, opener_position, current_position)
      end

      ::continue::
    end
  end

  local cont = lpeg.R("\128\191") -- continuation byte

```

Match a UTF-8 character of byte length `n`.

```
  local function utf8_by_byte_count(n)
    if (n == 1) then
      return lpeg.R("\0\127")
    end
    if (n == 2) then
      return lpeg.R("\194\223") * cont
    end
```

```
    if (n == 3) then
      return lpeg.R("\224\239") * cont * cont
    end
    if (n == 4) then
      return lpeg.R("\240\244") * cont * cont * cont
    end
  end
```

Check if a there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```
  local function check_unicode_type(s, i, start_pos, end_pos, chartype)
    local c
    local char_length
    for pos = start_pos, end_pos, 1 do
      if (start_pos < 0) then
        char_length = -pos
      else
        char_length = pos + 1
      end

      if (chartype == "punctuation") then
        if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
          return i
        end
      else
        c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
        if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
          return i
        end
      end
    end
  end

  local function check_preceding_unicode_punctuation(s, i)
    return check_unicode_type(s, i, -4, -1, "punctuation")
  end

  local function check_preceding_unicode_whitespace(s, i)
    return check_unicode_type(s, i, -4, -1, "%s")
  end

  local function check_following_unicode_punctuation(s, i)
    return check_unicode_type(s, i, 0, 3, "punctuation")
  end

  local function check_following_unicode_whitespace(s, i)
    return check_unicode_type(s, i, 0, 3, "%s")
```

```
end

parsers.unicode_preceding_punctuation = B(parsers.escapable)
                                      + Cmt(parsers.succeed, check_preceding_unicod

parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode

parsers.unicode_following_punctuation = #parsers.escapable
                                      + Cmt(parsers.succeed, check_following_unicod

parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicode

parsers.delimiter_run = function(character)
  return  (B(parsers.backslash * character) + -B(character))
        * character^1
        * -#character
end

parsers.left_flanking_delimiter_run = function(character)
  return  (B( parsers.any)
            * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh
           + -B(parsers.any))
          * parsers.delimiter_run(character)
          * parsers.unicode_following_punctuation
          + parsers.delimiter_run(character)
          * -#(parsers.unicode_following_punctuation + parsers.unicode_following_wh
              + parsers.eof)
end

parsers.right_flanking_delimiter_run = function(character)
  return  parsers.unicode_preceding_punctuation
        * parsers.delimiter_run(character)
        * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
          + parsers.eof)
        + (B(parsers.any)
          * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whi
        * parsers.delimiter_run(character)
end

if options.underscores then
  parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
                     + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
                         + (parsers.unicode_preceding_punctuation
                           * #parsers.right_flanking_delimiter_run(parsers.underscor
                     * parsers.left_flanking_delimiter_run(parsers.underscore)

  parsers.emph_end  = parsers.right_flanking_delimiter_run(parsers.asterisk)
```

```
                      + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
                        + #(parsers.left_flanking_delimiter_run(parsers.underscore)
                          * parsers.unicode_following_punctuation))
                      * parsers.right_flanking_delimiter_run(parsers.underscore)
else
  parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)

  parsers.emph_end  = parsers.right_flanking_delimiter_run(parsers.asterisk)
end

parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
                                      * Ct( Cg(Cc("delimiter"), "type")
                                          * Cg(Cc("emphasis"), "element")
                                          * Cg(C(parsers.emph_start), "content")
                                          * Cg(Cc(true), "is_opening")
                                          * Cg(Cc(true), "is_closing"))

parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
                                * Cg(Cc("emphasis"), "element")
                                * Cg(C(parsers.emph_start), "content")
                                * Cg(Cc(true), "is_opening")
                                * Cg(Cc(false), "is_closing"))

parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
                                 * Cg(Cc("emphasis"), "element")
                                 * Cg(C(parsers.emph_end), "content")
                                 * Cg(Cc(false), "is_opening")
                                 * Cg(Cc(true), "is_closing"))

parsers.emph_open_or_close  = parsers.emph_capturing_open_and_close
                            + parsers.emph_capturing_open
                            + parsers.emph_capturing_close

parsers.emph_open = parsers.emph_capturing_open_and_close
                  + parsers.emph_capturing_open

parsers.emph_close  = parsers.emph_capturing_open_and_close
                    + parsers.emph_capturing_close

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```
-- List of references defined in the document
local references

```

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

parsers.title_s_direct_ref  = parsers.squote
                            * Cs((parsers.html_entities
                                + (parsers.anyescaped - parsers.squote - parsers.bl
                            * parsers.squote

parsers.title_d_direct_ref  = parsers.dquote
                            * Cs((parsers.html_entities
                                + (parsers.anyescaped - parsers.dquote - parsers.bl
                            * parsers.dquote

parsers.title_p_direct_ref  = parsers.lparent
                            * Cs((parsers.html_entities
                                + (parsers.anyescaped - parsers.lparent - parsers.r
                            * 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.link_image_opening  = Ct( Cg(Cc("delimiter"), "type")
                                * Cg(Cc(true), "is_opening")
                                * Cg(Cc(false), "is_closing")
```

```
                                  * ( Cg(Cc("image"), "element")
                                    * parsers.image_opening
                                    * Cg(parsers.exclamation * parsers.lbracket, "con
                                    + Cg(Cc("link"), "element")
                                    * parsers.link_opening
                                    * Cg(parsers.lbracket, "content")))

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_re
                                    + 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), "con

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_ope
                                    * 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")
       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 function delete_parsed_content_in_range(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 function render_link_or_image(t, opening_index, closing_index, content_end_in
    process_emphasis(t, opening_index, content_end_index)
    local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index -

    local rendered = {}
    if (t[opening_index].element == "link") then
      rendered = writer.link(mapped, reference.url, reference.title, reference.attrib
    end

    if (t[opening_index].element == "image") then
      rendered = writer.image(mapped, reference.url, reference.title, reference.attri
    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 function resolve_inline_following_content(t, closing_index, match_reference,
```

```
8473     local content = ""
8474     for i = closing_index + 1, #t do
8475       content = content .. t[i].content
8476     end
8477
8478     local matching_content = parsers.succeed
8479
8480     if match_reference then
8481       matching_content = matching_content * parsers.inline_direct_ref_inside
8482     end
8483
8484     if match_link_attributes then
8485       matching_content = matching_content * Cg(Ct(parsers.attributes^-
    1), "attributes")
8486     end
8487
8488     local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), conte
8489
8490     local matched_count = matched.end_position - 1
8491     for i = closing_index + 1, #t do
8492       local value = t[i]
8493
8494       local chars_left = matched_count
8495       matched_count = matched_count - #value.content
8496
8497       if matched_count <= 0 then
8498         value.content = value.content:sub(chars_left + 1)
8499         break
8500       end
8501
8502       value.content = ''
8503       value.is_active = false
8504     end
8505
8506     local attributes = matched.attributes
8507     if attributes == nil or next(attributes) == nil then
8508       attributes = nil
8509     end
8510
8511     return {
8512       url = matched.url or "",
8513       title = matched.title or "",
8514       attributes = attributes
8515     }
8516   end
8517
```

Resolve an inline link a[33] 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
  render_link_or_image(t, opening_index, closing_index, closing_index, inline_conte
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
    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 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_cl
  local r = self.lookup_reference(next_link_content)

  if r then
    local inline_content = resolve_inline_following_content(t, next_link_closing_in
                                                            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,
  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)
```

[33] See b.

```
8551
8552      if r then
8553        local inline_content = resolve_inline_following_content(t, closing_index, false
8554        r.attributes = join_attributes(r.attributes, inline_content.attributes)
8555        render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8556      end
8557    end
8558
```

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.

```
8559    local function process_links_and_emphasis(t)
8560      for _,value in ipairs(t) do
8561        value.is_active = true
8562      end
8563
8564      for i,value in ipairs(t) do
8565        if not value.is_closing or
8566          value.type ~= "delimiter" or
8567          not (value.element == "link" or value.element == "image") then
8568          goto continue
8569        end
8570
8571        local opener_position = find_link_opener(t, 1, i - 1)
8572        if (opener_position == nil) then
8573          goto continue
8574        end
8575
8576        local opening_delimiter = t[opener_position]
8577        opening_delimiter.removed = true
8578
8579        local link_type = opening_delimiter.link_type
8580
8581        if (link_type == "inline") then
8582          resolve_inline_link(t, opener_position, i)
8583        end
8584        if (link_type == "shortcut") then
8585          resolve_shortcut_link(t, opener_position, i)
8586        end
8587        if (link_type == "full") then
8588          resolve_full_link(t, opener_position, i)
8589        end
8590        if (link_type == "collapsed") then
8591          resolve_collapsed_link(t, opener_position, i)
8592        end
```

```
8593
8594        ::continue::
8595      end
8596
8597      t[#t].content = t[#t].content:gsub("%s*$","")
8598
8599      process_emphasis(t, 1, #t)
8600      local final_result = collect_emphasis_content(t, 1, #t)
8601      return final_result
8602    end
8603
8604    function self.defer_link_and_emphasis_processing(delimiter_table)
8605      return writer.defer_call(function()
8606        return process_links_and_emphasis(delimiter_table)
8607      end)
8608    end
8609
```

3.1.6.8 Inline Elements (local)

```
8610    parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8611                       / writer.string
8612
8613    parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
8614                       / writer.string
8615
8616    parsers.Ellipsis = P("...") / writer.ellipsis
8617
8618    parsers.Smart    = parsers.Ellipsis
8619
8620    parsers.Code     = parsers.inticks / writer.code
8621
8622    if options.blankBeforeBlockquote then
8623      parsers.bqstart = parsers.fail
8624    else
8625      parsers.bqstart = parsers.blockquote_start
8626    end
8627
8628    if options.blankBeforeHeading then
8629      parsers.headerstart = parsers.fail
8630    else
8631      parsers.headerstart = parsers.atx_heading
8632    end
8633
8634    if options.blankBeforeList then
8635      parsers.interrupting_bullets = parsers.fail
8636      parsers.interrupting_enumerators = parsers.fail
```

```
8637  else
8638    parsers.interrupting_bullets  = parsers.bullet(parsers.dash, true)
8639                                  + parsers.bullet(parsers.asterisk, true)
8640                                  + parsers.bullet(parsers.plus, true)
8641
8642    parsers.interrupting_enumerators  = parsers.enumerator(parsers.period, true)
8643                                      + parsers.enumerator(parsers.rparent, true)
8644  end
8645
8646  if options.html then
8647    parsers.html_interrupting = parsers.check_trail
8648                              * ( parsers.html_incomplete_open_tag
8649                                + parsers.html_incomplete_close_tag
8650                                + parsers.html_incomplete_open_special_tag
8651                                + parsers.html_comment_start
8652                                + parsers.html_cdatasection_start
8653                                + parsers.html_declaration_start
8654                                + parsers.html_instruction_start
8655                                - parsers.html_close_special_tag
8656                                - parsers.html_empty_special_tag)
8657  else
8658    parsers.html_interrupting = parsers.fail
8659  end
8660
8661  parsers.EndlineExceptions
8662                    = parsers.blankline -- paragraph break
8663                    + parsers.eof       -- end of document
8664                    + parsers.bqstart
8665                    + parsers.thematic_break_lines
8666                    + parsers.interrupting_bullets
8667                    + parsers.interrupting_enumerators
8668                    + parsers.headerstart
8669                    + parsers.html_interrupting
8670
8671  parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8672
8673  parsers.endline = parsers.newline
8674                  * (parsers.check_minimal_indent
8675                    * -V("EndlineExceptions")
8676                    + parsers.check_optional_indent
8677                    * -V("EndlineExceptions")
8678                    * -parsers.starter)
8679                  * parsers.spacechar^0
8680
8681  parsers.Endline = parsers.endline
8682                  / writer.soft_line_break
8683
```

```
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.
                    + 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.
                    + parsers.spacechar^1 * parsers.Endline
                                          / writer.soft_line_break
                    + parsers.spacechar^1 * -parsers.newline / self.expandtabs

parsers.NonbreakingEndline
                   = parsers.endline
                   / writer.soft_line_break

parsers.NonbreakingSpace
                 = parsers.spacechar^2 * parsers.Endline
                                       / writer.hard_line_break
                 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                 + parsers.spacechar^1 * parsers.Endline
                                       * parsers.optionalspace
                                       / writer.soft_line_break
                 + 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_commen
                        + 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

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

parsers.Blockquote   = parsers.blockquote_body
                     / writer.blockquote

parsers.ThematicBreak = parsers.thematic_break_lines
                      / writer.thematic_break

parsers.Reference    = parsers.define_reference_parser
                     / self.register_link

parsers.Paragraph   = parsers.freeze_trail
                    * (Ct((parsers.Inline)^1)
                    * (parsers.newline + parsers.eof)
                    * parsers.unfreeze_trail
                    / writer.paragraph)

parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
                     / writer.plain
```

3.1.6.10 Lists (local)

```

if options.taskLists then
  parsers.tickbox = ( parsers.ticked_box
                    + parsers.halfticked_box
                    + parsers.unticked_box
                    ) / writer.tickbox
else
   parsers.tickbox = parsers.fail
```

```
8812  end
8813
8814  parsers.list_blank = parsers.conditionally_indented_blankline
8815
8816  parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_blan
8817                                      * parsers.minimally_indented_ref
8818                                      + parsers.block_sep_group(parsers.list_blank)
8819                                      * parsers.minimally_indented_block
8820
8821  parsers.ref_or_block_non_separated  = parsers.minimally_indented_ref
8822                                      + (parsers.succeed / writer.interblocksep)
8823                                      * parsers.minimally_indented_block
8824                                      - parsers.minimally_indented_blankline
8825
8826  parsers.tight_list_loop_body_pair  =
8827    parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8828                                  parsers.minimally_indented_par_or_plain_no_blank,
8829                                  (parsers.succeed / writer.interblocksep),
8830                                  (parsers.succeed / writer.paragraphsep))
8831
8832  parsers.loose_list_loop_body_pair  =
8833    parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,
8834                                  parsers.minimally_indented_par_or_plain,
8835                                  parsers.block_sep_group(parsers.list_blank),
8836                                  parsers.par_sep_group(parsers.list_blank))
8837
8838  parsers.tight_list_content_loop = V("Block")
8839                                  * parsers.tight_list_loop_body_pair.block^0
8840                                  + (V("Paragraph") + V("Plain"))
8841                                  * parsers.ref_or_block_non_separated
8842                                  * parsers.tight_list_loop_body_pair.block^0
8843                                  +  (V("Paragraph") + V("Plain"))
8844                                  * parsers.tight_list_loop_body_pair.par^0
8845
8846  parsers.loose_list_content_loop = V("Block")
8847                                  * parsers.loose_list_loop_body_pair.block^0
8848                                  + (V("Paragraph") + V("Plain"))
8849                                  * parsers.ref_or_block_list_separated
8850                                  * parsers.loose_list_loop_body_pair.block^0
8851                                  + (V("Paragraph") + V("Plain"))
8852                                  * parsers.loose_list_loop_body_pair.par^0
8853
8854  parsers.list_item_tightness_condition = -#( parsers.list_blank^0
8855                                              * parsers.minimally_indented_ref_or_block
8856                                        * remove_indent("li")
8857                                        + remove_indent("li")
8858                                        * parsers.fail
```

```
8859
8860  parsers.indented_content_tight  = Ct( (parsers.blankline / "")
8861                                      * #parsers.list_blank
8862                                      * remove_indent("li")
8863                                      + ( (V("Reference") + (parsers.blankline / ""))
8864                                        * parsers.check_minimal_indent
8865                                        * parsers.tight_list_content_loop
8866                                        + (V("Reference") + (parsers.blankline / ""))
8867                                        + (parsers.tickbox^-1 / writer.escape)
8868                                        * parsers.tight_list_content_loop
8869                                        )
8870                                      * parsers.list_item_tightness_condition
8871                                  )
8872
8873  parsers.indented_content_loose  = Ct( (parsers.blankline / "")
8874                                      * #parsers.list_blank
8875                                      + ( (V("Reference") + (parsers.blankline / ""))
8876                                        * parsers.check_minimal_indent
8877                                        * parsers.loose_list_content_loop
8878                                        + (V("Reference") + (parsers.blankline / ""))
8879                                        + (parsers.tickbox^-1 / writer.escape)
8880                                        * parsers.loose_list_content_loop
8881                                        )
8882                                  )
8883
8884  parsers.TightListItem = function(starter)
8885    return  -parsers.ThematicBreak
8886            * parsers.add_indent(starter, "li")
8887            * parsers.indented_content_tight
8888  end
8889
8890  parsers.LooseListItem = function(starter)
8891    return  -parsers.ThematicBreak
8892            * parsers.add_indent(starter, "li")
8893            * parsers.indented_content_loose
8894            * remove_indent("li")
8895  end
8896
8897  parsers.BulletListOfType = function(bullet_type)
8898    local bullet = parsers.bullet(bullet_type)
8899    return  ( Ct( parsers.TightListItem(bullet)
8900                * ( (parsers.check_minimal_indent / "")
8901                  * parsers.TightListItem(bullet)
8902                  )^0
8903              )
8904              * Cc(true)
8905              * -#( (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(enumera
        * 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(enumera
        * 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.hea
                     * parsers.newline^-1

parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
                      * #(parsers.heading_text
                         * parsers.check_minimal_indent * parsers.check_trail * par
                      * Cs(parsers.heading_text)
                      / parsers.parse_heading_text
                      * parsers.check_minimal_indent_and_trail * parsers.heading_le
                      * 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 / writer.interblocksep)
                                )^-1
                              * 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.fail,

        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,
```

```
9115          Smart                = parsers.Smart,
9116          Symbol               = parsers.Symbol,
9117          SpecialChar          = parsers.fail,
9118          InitializeState      = parsers.succeed,
9119        }
```

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`]`.

```
9120      self.update_rule = function(rule_name, get_pattern)
9121        assert(current_extension_name ~= nil)
9122        assert(syntax[rule_name] ~= nil,
9123          [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
9124        local previous_pattern
9125        local extension_name
9126        if walkable_syntax[rule_name] then
9127          local previous_accountable_pattern = walkable_syntax[rule_name][1]
9128          previous_pattern = previous_accountable_pattern[1]
9129          extension_name = previous_accountable_pattern[2] .. ", " .. current_extension
9130        else
9131          previous_pattern = nil
9132          extension_name = current_extension_name
9133        end
9134        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
```

```
9135        if type(get_pattern) == "function" then
9136          pattern = get_pattern(previous_pattern)
9137        else
9138          assert(previous_pattern == nil,
9139                 [[Rule ]] .. rule_name ..
9140                 [[ has already been updated by ]] .. extension_name)
9141          pattern = get_pattern
9142        end
9143        local accountable_pattern = { pattern, extension_name, rule_name }
```

```
9144        walkable_syntax[rule_name] = { accountable_pattern }
9145      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.

```
9146      local special_characters = {}
9147      self.add_special_character = function(c)
9148        table.insert(special_characters, c)
9149        syntax.SpecialChar = S(table.concat(special_characters, ""))
9150      end
9151
9152      self.add_special_character("*")
9153      self.add_special_character("[")
9154      self.add_special_character("]")
9155      self.add_special_character("<")
9156      self.add_special_character("!")
9157      self.add_special_character("\\")
```

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

```
9158      self.initialize_named_group = function(name, value)
9159        local pattern = Ct("")
9160        if value ~= nil then
9161          pattern = pattern / value
9162        end
9163        syntax.InitializeState = syntax.InitializeState
9164                               * Cg(pattern, name)
9165      end
```

Add a named group for indentation.

```
9166      self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
9167      for _, extension in ipairs(extensions) do
9168        current_extension_name = extension.name
9169        extension.extend_writer(writer)
9170        extension.extend_reader(self)
9171      end
9172      current_extension_name = nil
```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```
9173      if options.debugExtensions then
9174        local sorted_lhs = {}
9175        for lhs, _ in pairs(walkable_syntax) do
9176          table.insert(sorted_lhs, lhs)
9177        end
```

```
9178      table.sort(sorted_lhs)
9179
9180      local output_lines = {"{"}
9181      for lhs_index, lhs in ipairs(sorted_lhs) do
9182        local encoded_lhs = util.encode_json_string(lhs)
9183        table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
9184        local rule = walkable_syntax[lhs]
9185        for rhs_index, rhs in ipairs(rule) do
9186          local human_readable_rhs
9187          if type(rhs) == "string" then
9188            human_readable_rhs = rhs
9189          else
9190            local pattern_name
9191            if rhs[3] then
9192              pattern_name = rhs[3]
9193            else
9194              pattern_name = "Anonymous Pattern"
9195            end
9196            local extension_name = rhs[2]
9197            human_readable_rhs = pattern_name .. [[ (]] .. extension_name .. [[)]]
9198          end
9199          local encoded_rhs = util.encode_json_string(human_readable_rhs)
9200          local output_line = [[        ]] .. encoded_rhs
9201          if rhs_index < #rule then
9202            output_line = output_line .. ","
9203          end
9204          table.insert(output_lines, output_line)
9205        end
9206        local output_line = "    ]"
9207        if lhs_index < #sorted_lhs then
9208          output_line = output_line .. ","
9209        end
9210        table.insert(output_lines, output_line)
9211      end
9212      table.insert(output_lines, "}")
9213
9214      local output = table.concat(output_lines, "\n")
9215      local output_filename = options.debugExtensionsFileName
9216      local output_file = assert(io.open(output_filename, "w"),
9217        [[Could not open file "]] .. output_filename .. [[" for writing]])
9218      assert(output_file:write(output))
9219      assert(output_file:close())
9220    end
```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```
    for lhs, rule in pairs(walkable_syntax) do
      syntax[lhs] = parsers.fail
      for _, rhs in ipairs(rule) do
        local pattern
```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```
        if type(rhs) == "string" then
          pattern = V(rhs)
        else
          pattern = rhs[1]
          if type(pattern) == "string" then
            pattern = V(pattern)
          end
        end
        syntax[lhs] = syntax[lhs] + pattern
      end
    end
```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

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

    if not options.codeSpans then
      syntax.Code = parsers.fail
    else
      self.add_special_character("`")
    end

    if not options.html then
      syntax.DisplayHtml = parsers.fail
      syntax.InlineHtml = parsers.fail
      syntax.HtmlEntity  = parsers.fail
    else
      self.add_special_character("&")
    end

    if options.preserveTabs then
      options.stripIndent = false
    end

    if not options.smartEllipses then
```

```
      syntax.Smart = parsers.fail
    else
      self.add_special_character(".")
    end

    if not options.relativeReferences then
      syntax.AutoLinkRelativeReference = parsers.fail
    end

    if options.contentLevel == "inline" then
      syntax[1] = "Inlines"
      syntax.Inlines = V("InitializeState")
                       * parsers.Inline^0
                       * ( parsers.spacing^0
                         * parsers.eof / "")
      syntax.Space = parsers.Space + parsers.blankline / writer.space
    end

    local blocks_nested_t = util.table_copy(syntax)
    blocks_nested_t.ExpectedJekyllData = parsers.fail
    parsers.blocks_nested = Ct(blocks_nested_t)

    parsers.blocks = Ct(syntax)

    local inlines_t = util.table_copy(syntax)
    inlines_t[1] = "Inlines"
    inlines_t.Inlines = V("InitializeState")
                       * parsers.Inline^0
                       * ( parsers.spacing^0
                         * parsers.eof / "")
    parsers.inlines = Ct(inlines_t)

    local inlines_no_inline_note_t = util.table_copy(inlines_t)
    inlines_no_inline_note_t.InlineNote = parsers.fail
    parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)

    local inlines_no_html_t = util.table_copy(inlines_t)
    inlines_no_html_t.DisplayHtml = parsers.fail
    inlines_no_html_t.InlineHtml = parsers.fail
    inlines_no_html_t.HtmlEntity = parsers.fail
    parsers.inlines_no_html = Ct(inlines_no_html_t)

    local inlines_nbsp_t = util.table_copy(inlines_t)
    inlines_nbsp_t.Endline = parsers.NonbreakingEndline
    inlines_nbsp_t.Space = parsers.NonbreakingSpace
    parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

```
9306      local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9307      inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9308      inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9309      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..

```
9310      return function(input)
```

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.

```
9311        input = input:gsub("\r\n?", "\n")
9312        if input:sub(-1) ~= "\n" then
9313          input = input .. "\n"
9314        end
```

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). The `cacheDir` option is disregarded.

```
9315        references = {}
9316        local opt_string = {}
9317        for k, _ in pairs(defaultOptions) do
9318          local v = options[k]
9319          if type(v) == "table" then
9320            for _, i in ipairs(v) do
9321              opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9322            end
9323          elseif k ~= "cacheDir" then
9324            opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9325          end
9326        end
9327        table.sort(opt_string)
9328        local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9329        local output
9330        local function convert(input)
9331          local document = self.parser_functions.parse_blocks(input)
9332          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`.

```
9333          local undosep_start, undosep_end
9334          local potential_secend_start, secend_start
9335          local potential_sep_start, sep_start
9336          while true do
9337            -- find a `writer->undosep`
9338            undosep_start, undosep_end = output:find(writer.undosep_text, 1, true)
9339            if undosep_start == nil then break end
```

```
9340          -- skip any preceding section ends
9341          secend_start = undosep_start
9342          while true do
9343            potential_secend_start = secend_start - #writer.secend_text
9344            if potential_secend_start < 1
9345               or output:sub(potential_secend_start, secend_start - 1) ~= writer.sece
9346              break
9347            end
9348            secend_start = potential_secend_start
9349          end
9350          -- find an immediately preceding block element / paragraph separator
9351          sep_start = secend_start
9352          potential_sep_start = sep_start - #writer.interblocksep_text
9353          if potential_sep_start >= 1
9354             and output:sub(potential_sep_start, sep_start - 1) == writer.interblocks
9355            sep_start = potential_sep_start
9356          else
9357            potential_sep_start = sep_start - #writer.paragraphsep_text
9358            if potential_sep_start >= 1
9359               and output:sub(potential_sep_start, sep_start - 1) == writer.paragraph
9360              sep_start = potential_sep_start
9361            end
9362          end
9363          -- remove `writer->undosep` and immediately preceding block element / parag
9364          output = output:sub(1, sep_start - 1)
9365                .. output:sub(secend_start, undosep_start - 1)
9366                .. output:sub(undosep_end + 1)
9367        end
9368        return output
9369      end
```

If we cache markdown documents, produce the cache file and transform its filename to plain TEX output via the `writer->pack` method.

```
9370      if options.eagerCache or options.finalizeCache then
9371        local name = util.cache(options.cacheDir, input, salt, convert,
9372                                ".md" .. writer.suffix)
9373        output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
9374      else
9375        output = convert(input)
9376      end
```

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

```
9377      if options.finalizeCache then
9378        local file, mode
```

```
        if options.frozenCacheCounter > 0 then
          mode = "a"
        else
          mode = "w"
        end
        file = assert(io.open(options.frozenCacheFileName, mode),
          [[Could not open file "]] .. options.frozenCacheFileName
          .. [[" for writing]])
        assert(file:write([[\expandafter\global\expandafter\def\csname ]]
          .. [[markdownFrozenCache]] .. options.frozenCacheCounter
          .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
        assert(file:close())
      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.rbr
                                + V("Space") + V("Endline")
                                + (parsers.any
                                  - (parsers.newline + parsers.lbracket + parsers.rbr
                                    + 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.

```
9440      function self.citations(text_cites, cites)
9441        local buffer = {}
9442        if self.flatten_inlines then
9443          for _,cite in ipairs(cites) do
9444            if cite.prenote then
9445              table.insert(buffer, {cite.prenote, " "})
9446            end
9447            table.insert(buffer, cite.name)
9448            if cite.postnote then
9449              table.insert(buffer, {" ", cite.postnote})
9450            end
9451          end
9452        else
9453          table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "C
9454            "{", #cites, "}"})
9455          for _,cite in ipairs(cites) do
9456            table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9457              cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"})
9458          end
9459        end
9460        return buffer
9461      end
9462    end, extend_reader = function(self)
9463      local parsers = self.parsers
9464      local writer = self.writer
9465
9466      local citation_chars
9467                    = parsers.alphanumeric
9468                    + S("#$%&-+<>~/_")
9469
9470      local citation_name
9471                    = Cs(parsers.dash^-1) * parsers.at
9472                    * Cs(citation_chars
9473                        * (((citation_chars + parsers.internal_punctuation
9474                            - parsers.comma - parsers.semicolon)
9475                           * -#((parsers.internal_punctuation - parsers.comma
9476                                - parsers.semicolon)^0
9477                               * -(citation_chars + parsers.internal_punctuation
9478                                  - parsers.comma - parsers.semicolon)))^0
9479                          * citation_chars)^-1)
9480
9481      local citation_body_prenote
9482                    = Cs((parsers.alphanumeric^1
```

```
                        + parsers.bracketed
                        + parsers.inticks
                        + parsers.autolink
                        + V("InlineHtml")
                        + V("Space") + V("Endline")
                        + (parsers.anyescaped
                          - (parsers.newline + parsers.rbracket + parsers.blankline^
                        - (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("Endline")
                        + (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
                      * citation_body_postnote
                      + Cc("")
                      * parsers.spnlc
                    )

      local citation_body
                    = citation_body_chunk
                    * ( parsers.semicolon
                      * parsers.spnlc
                      * citation_body_chunk
                    )^0

      local citation_headless_body_postnote
                    = Cs((parsers.alphanumeric^1
                        + parsers.bracketed
                        + parsers.inticks
```

```
                        + parsers.autolink
                        + V("InlineHtml")
                        + V("Space") + V("Endline")
                        + (parsers.anyescaped
                          - (parsers.newline + parsers.rbracket + parsers.at
                            + parsers.semicolon + parsers.blankline^2))
                        - (parsers.spnl * parsers.rbracket))^0

    local citation_headless_body
                  = citation_headless_body_postnote
                  * ( parsers.semicolon
                    * parsers.spnlc
                    * citation_body_chunk
                  )^0

    local citations
                  = function(text_cites, raw_cites)
        local function normalize(str)
            if str == "" then
                str = nil
            else
                str = (citation_nbsps and
                  self.parser_functions.parse_inlines_nbsp or
                  self.parser_functions.parse_inlines)(str)
            end
            return str
        end

        local cites = {}
        for i = 1,#raw_cites,4 do
            cites[#cites+1] = {
                prenote = normalize(raw_cites[i]),
                suppress_author = raw_cites[i+1] == "-",
                name = writer.identifier(raw_cites[i+2]),
                postnote = normalize(raw_cites[i+3]),
            }
        end
        return writer.citations(text_cites, cites)
    end

    local TextCitations
                  = Ct((parsers.spnlc
                  * Cc("")
                  * citation_name
                  * ((parsers.spnlc
                      * parsers.lbracket
                      * citation_headless_body
```

```
                        * parsers.rbracket) + Cc("")))^1)
                    / function(raw_cites)
                        return citations(true, raw_cites)
                      end

      local ParenthesizedCitations
                    = Ct((parsers.spnlc
                    * parsers.lbracket
                    * citation_body
                    * parsers.rbracket)^1)
                    / function(raw_cites)
                        return citations(false, raw_cites)
                      end

      local Citations = TextCitations + ParenthesizedCitations

      self.insert_pattern("Inline before LinkAndEmph",
                          Citations, "Citations")

      self.add_special_character("@")
      self.add_special_character("-")
    end
  }
end
```

3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```
M.extensions.content_blocks = function(language_map)
```

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

```
  local languages_json = (function()
    local base, prev, curr
    for _, pathname in ipairs{kpse.lookup(language_map, { all=true })} do
      local file = io.open(pathname, "r")
      if not file then goto continue end
      local input = assert(file:read("*a"))
      assert(file:close())
      local json = input:gsub('("[^\n]-"):','[%1]=')
      curr = load("_ENV = {}; return "..json)()
      if type(curr) == "table" then
        if base == nil then
```

```
          base = curr
        else
          setmetatable(prev, { __index = curr })
        end
        prev = curr
      end
      ::continue::
    end
    return base or {}
  end)()

  return {
    name = "built-in content_blocks syntax extension",
    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.

```
      function self.contentblock(src,suf,type,tit)
        if not self.is_writing then return "" end
        src = src.."."..suf
        suf = suf:lower()
        if type == "onlineimage" then
          return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
                                  "{",self.string(src),"}",
                                  "{",self.uri(src),"}",
                                  "{",self.string(tit or ""),"}"}
        elseif languages_json[suf] then
          return {"\\markdownRendererContentBlockCode{",suf,"}",
                                  "{",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
                       - parsers.more
                       - parsers.spacing
                       - #(parsers.period
                          * onlineimagesuffix
                          * parsers.more
                          * contentblock_tail))^0)
                  * parsers.period
                  * Cs(onlineimagesuffix)
                  * parsers.more
                  + (Cs((parsers.inparens
                        + (parsers.anyescaped
                          - parsers.spacing
                          - parsers.rparent
                          - #(parsers.period
                             * onlineimagesuffix
                             * contentblock_tail)))^0)
```

```
                        * parsers.period
                        * Cs(onlineimagesuffix))
                      ) * Cc("onlineimage")

      -- filename for iA Writer content blocks with mandatory suffix:
      local localfilepath
                    = parsers.slash
                    * Cs((parsers.anyescaped
                         - parsers.tab
                         - parsers.newline
                         - #(parsers.period
                            * parsers.alphanumeric^1
                            * contentblock_tail))^1)
                    * parsers.period
                    * Cs(parsers.alphanumeric^1)
                    * Cc("localfile")

      local ContentBlock
                    = parsers.check_trail_no_rem
                    * (localfilepath + onlineimageurl)
                    * contentblock_tail
                    / writer.contentblock

      self.insert_pattern("Block before Blockquote",
                          ContentBlock, "ContentBlock")
    end
  }
end
```

3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

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

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

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
```

```
                                "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

      local defstart  = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
                                      * (parsers.tab + parsers.space^-
3)
                      + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
                                       * (parsers.tab + parsers.space^-
2)
                      + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
                                       * (parsers.tab + parsers.space^-
1)
                      + parsers.check_trail_length(3) * defstartchar * #parsers.spaci

      local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_

      local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option

      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.

```
9820      function self.fancylist(items,tight,startnum,numstyle,numdelim)
9821        if not self.is_writing then return "" end
9822        local buffer = {}
9823        local num = startnum
9824        for _,item in ipairs(items) do
9825          if item ~= "" then
9826            buffer[#buffer + 1] = self.fancyitem(item,num)
9827          end
9828          if num ~= nil and item ~= "" then
9829            num = num + 1
9830          end
9831        end
9832        local contents = util.intersperse(buffer,"\n")
9833        if tight and options.tightLists then
9834          return {"\\markdownRendererFancyOlBeginTight{",
9835                  numstyle,"}{",numdelim,"}",contents,
9836                  "\n\\markdownRendererFancyOlEndTight "}
9837        else
9838          return {"\\markdownRendererFancyOlBegin{",
9839                  numstyle,"}{",numdelim,"}",contents,
9840                  "\n\\markdownRendererFancyOlEnd "}
9841        end
9842      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.

```
9843      function self.fancyitem(s,num)
```

```
        if num ~= nil then
          return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
                  "\\markdownRendererFancyOlItemEnd "}
        else
          return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEn
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local options = self.options
      local writer = self.writer

      local function combine_markers_and_delims(markers, delims)
        local markers_table = {}
        for _,marker in ipairs(markers) do
          local start_marker
          local continuation_marker
          if type(marker) == "table" then
            start_marker = marker[1]
            continuation_marker = marker[2]
          else
            start_marker = marker
            continuation_marker = marker
          end
          for _,delim in ipairs(delims) do
            table.insert(markers_table, {start_marker, continuation_marker, delim})
          end
        end
        return markers_table
      end

      local function join_table_with_func(func, markers_table)
        local pattern = func(table.unpack(markers_table[1]))
        for i = 2, #markers_table do
          pattern = pattern + func(table.unpack(markers_table[i]))
        end
        return pattern
      end

      local lowercase_letter_marker = R("az")
      local uppercase_letter_marker = R("AZ")

      local roman_marker = function(chars)
        local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
        local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
        return  m^-3
                * (c*m + c*d + d^-1 * c^-3)
```

```
              * (x*c + x*l + l^-1 * x^-3)
              * (i*x + i*v + v^-1 * i^-3)
    end

    local lowercase_roman_marker  = roman_marker({"m", "d", "c", "l", "x", "v", "i"
    local uppercase_roman_marker  = roman_marker({"M", "D", "C", "L", "X", "V", "I"

    local lowercase_opening_roman_marker  = P("i")
    local uppercase_opening_roman_marker  = P("I")

    local digit_marker = parsers.dig * parsers.dig^-8

    local markers = {
      {lowercase_opening_roman_marker, lowercase_roman_marker},
      {uppercase_opening_roman_marker, uppercase_roman_marker},
      lowercase_letter_marker,
      uppercase_letter_marker,
      lowercase_roman_marker,
      uppercase_roman_marker,
      digit_marker
    }

    local delims = {
      parsers.period,
      parsers.rparent
    }

    local markers_table = combine_markers_and_delims(markers, delims)

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

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

    local starter = join_table_with_func(enumerator, markers_table)
```

```
9938        local TightListItem = function(starter)
9939          return  parsers.add_indent(starter, "li")
9940                  * parsers.indented_content_tight
9941        end
9942
9943        local LooseListItem = function(starter)
9944          return  parsers.add_indent(starter, "li")
9945                  * parsers.indented_content_loose
9946                  * remove_indent("li")
9947        end
9948
9949        local function roman2number(roman)
9950          local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
9951          local numeral = 0
9952
9953          local i = 1
9954          local len = string.len(roman)
9955          while i < len do
9956            local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
9957            if z1 < z2 then
9958                numeral = numeral + (z2 - z1)
9959                i = i + 2
9960            else
9961                numeral = numeral + z1
9962                i = i + 1
9963            end
9964          end
9965          if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
9966          return numeral
9967        end
9968
9969        local function sniffstyle(numstr, delimend)
9970          local numdelim
9971          if delimend == ")" then
9972            numdelim = "OneParen"
9973          elseif delimend == "." then
9974            numdelim = "Period"
9975          else
9976            numdelim = "Default"
9977          end
9978
9979          local num
9980          num = numstr:match("^([I])$")
9981          if num then
9982            return roman2number(num), "UpperRoman", numdelim
9983          end
9984          num = numstr:match("^([i])$")
```

```
9985          if num then
9986            return roman2number(string.upper(num)), "LowerRoman", numdelim
9987          end
9988          num = numstr:match("^([A-Z])$")
9989          if num then
9990            return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
9991          end
9992          num = numstr:match("^([a-z])$")
9993          if num then
9994            return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
9995          end
9996          num = numstr:match("^([IVXLCDM]+)")
9997          if num then
9998            return roman2number(num), "UpperRoman", numdelim
9999          end
10000         num = numstr:match("^([ivxlcdm]+)")
10001         if num then
10002           return roman2number(string.upper(num)), "LowerRoman", numdelim
10003         end
10004         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10005       end
10006
10007       local function fancylist(items,tight,start)
10008         local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
10009         return writer.fancylist(items,tight,
10010                                 options.startNumber and startnum or 1,
10011                                 numstyle or "Decimal",
10012                                 numdelim or "Default")
10013       end
10014
10015       local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
10016         local enumerator_start = enumerator(start_marker, continuation_marker, delimi
10017         local enumerator_cont = enumerator(continuation_marker, continuation_marker,
10018         return Cg(enumerator_start, "listtype")
10019              * (Ct( TightListItem(Cb("listtype"))
10020                  * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
10021              * Cc(true)
10022              * -#((parsers.conditionally_indented_blankline^0 / "")
10023                  * parsers.check_minimal_indent * enumerator_cont)
10024              + Ct( LooseListItem(Cb("listtype"))
10025                  * ((parsers.conditionally_indented_blankline^0 / "")
10026                    * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
10027              * Cc(false)
10028              ) * Ct(Cb("listtype")) / fancylist
10029       end
10030
10031       local FancyList = join_table_with_func(FancyListOfType, markers_table)
```

```
10032
10033       local Endline   = parsers.newline
10034                       * (parsers.check_minimal_indent
10035                         * -parsers.EndlineExceptions
10036                         + parsers.check_optional_indent
10037                         * -parsers.EndlineExceptions
10038                         * -starter)
10039                       * parsers.spacechar^0
10040                       / writer.soft_line_break
10041
10042       self.update_rule("OrderedList", FancyList)
10043       self.update_rule("Endline", Endline)
10044     end
10045   }
10046 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.

```
10047 M.extensions.fenced_code = function(blank_before_code_fence,
10048                                       allow_attributes,
10049                                       allow_raw_blocks)
10050   return {
10051     name = "built-in fenced_code syntax extension",
10052     extend_writer = function(self)
10053       local options = self.options
10054
```

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.

```
10055       function self.fencedCode(s, i, attr)
10056         if not self.is_writing then return "" end
10057         s = s:gsub("\n$", "")
10058         local buf = {}
10059         if attr ~= nil then
10060           table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
10061                              self.attributes(attr)})
10062         end
10063         local name = util.cache_verbatim(options.cacheDir, s)
10064         table.insert(buf, {"\\markdownRendererInputFencedCode{",
```

```
                            name,"}{",self.string(i),"}{",self.infostring(i),"}"})
      if attr ~= nil then
        table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd{}")
      end
      return buf
    end

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```
    if allow_raw_blocks then
      function self.rawBlock(s, attr)
        if not self.is_writing then return "" end
        s = s:gsub("\n$", "")
        local name = util.cache_verbatim(options.cacheDir, s)
        return {"\\markdownRendererInputRawBlock{",
                name,"}{", self.string(attr),"}"}
      end
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

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

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

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

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

    local fenceindent

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

```
10109           next(indent_table.trail) ~= nil
10110       end
10111
10112       local function has_indents(indent_table)
10113         return indent_table ~= nil and
10114           indent_table.indents ~= nil and
10115           next(indent_table.indents) ~= nil
10116       end
10117
10118       local function get_last_indent_name(indent_table)
10119         if has_indents(indent_table) then
10120           return indent_table.indents[#indent_table.indents].name
10121         end
10122       end
10123
10124       local function count_fenced_start_indent(_, _, indent_table, trail)
10125         local last_indent_name = get_last_indent_name(indent_table)
10126         fenceindent = 0
10127         if last_indent_name ~= "li" then
10128           fenceindent = #trail
10129         end
10130         return true
10131       end
10132
10133       local fencehead      = function(char, infostring)
10134         return               Cmt(Cb("indent_info") * parsers.check_trail, count_fence
10135                            * Cg(char^3, "fencelength")
10136                            * parsers.optionalspace
10137                            * infostring
10138                            * (parsers.newline + parsers.eof)
10139       end
10140
10141       local fencetail      = function(char)
10142         return               parsers.check_trail_no_rem
10143                            * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10144                            * parsers.optionalspace * (parsers.newline + parsers.eof)
10145                            + parsers.eof
10146       end
10147
10148       local function process_fenced_line(s, i, indent_table, line_content, is_blank)
10149         local remainder = ""
10150         if has_trail(indent_table) then
10151           remainder = indent_table.trail.internal_remainder
10152         end
10153
10154         if is_blank and get_last_indent_name(indent_table) == "li" then
10155           remainder = ""
```

```
10156         end
10157
10158         local str = remainder .. line_content
10159         local index = 1
10160         local remaining = fenceindent
10161
10162         while true do
10163           local c = str:sub(index, index)
10164           if c == " " and remaining > 0 then
10165             remaining = remaining - 1
10166             index = index + 1
10167           elseif c == "\t" and remaining > 3 then
10168             remaining = remaining - 4
10169             index = index + 1
10170           else
10171             break
10172           end
10173         end
10174
10175         return true, str:sub(index)
10176       end
10177
10178       local fencedline = function(char)
10179         return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
10180       end
10181
10182       local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true)
10183
10184       local TildeFencedCode
10185              = fencehead(parsers.tilde, tilde_infostring)
10186              * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10187                    + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
10188              * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
10189
10190       local BacktickFencedCode
10191              = fencehead(parsers.backtick, backtick_infostring)
10192              * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10193                    + (parsers.check_minimal_indent / "") * fencedline(parsers.backtic
10194              * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10195
10196       local infostring_with_attributes
10197                         = Ct(C((parsers.linechar
10198                               - ( parsers.optionalspace
10199                                 * parsers.attributes))^0)
10200                             * parsers.optionalspace
10201                             * Ct(parsers.attributes))
10202
```

```
10203       local FencedCode
10204                = ((TildeFencedCode + BacktickFencedCode)
10205                / function(infostring, code)
10206                    local expanded_code = self.expandtabs(code)
10207
10208                    if allow_raw_blocks then
10209                      local raw_attr = lpeg.match(parsers.raw_attribute,
10210                                                  infostring)
10211                      if raw_attr then
10212                        return writer.rawBlock(expanded_code, raw_attr)
10213                      end
10214                    end
10215
10216                    local attr = nil
10217                    if allow_attributes then
10218                      local match = lpeg.match(infostring_with_attributes,
10219                                               infostring)
10220                      if match then
10221                        infostring, attr = table.unpack(match)
10222                      end
10223                    end
10224                    return writer.fencedCode(expanded_code, infostring, attr)
10225                  end)
10226
10227       self.insert_pattern("Block after Verbatim",
10228                           FencedCode, "FencedCode")
10229
10230       local fencestart
10231       if blank_before_code_fence then
10232         fencestart = parsers.fail
10233       else
10234         fencestart = fencehead(parsers.backtick, backtick_infostring)
10235                    + fencehead(parsers.tilde, tilde_infostring)
10236       end
10237
10238       self.update_rule("EndlineExceptions", function(previous_pattern)
10239         if previous_pattern == nil then
10240           previous_pattern = parsers.EndlineExceptions
10241         end
10242         return previous_pattern + fencestart
10243       end)
10244
10245       self.add_special_character("`")
10246       self.add_special_character("~")
10247     end
10248   }
10249 end
```

3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

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

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```
      function self.div_begin(attributes)
        local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
                              self.attributes(attributes)}
        local end_output = {"\\markdownRendererFencedDivAttributeContextEnd{}"}
        return self.push_attributes("div", attributes, start_output, end_output)
      end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
      function self.div_end()
        return self.pop_attributes("div")
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
      local fenced_div_infostring
                              = C((parsers.linechar
                                 - ( parsers.spacechar^1
                                   * parsers.colon^1))^1)

      local fenced_div_begin = parsers.nonindentspace
                             * parsers.colon^3
                             * parsers.optionalspace
                             * fenced_div_infostring
                             * ( parsers.spacechar^1
                               * parsers.colon^1)^0
                             * parsers.optionalspace
                             * (parsers.newline + parsers.eof)

      local fenced_div_end = parsers.nonindentspace
                           * parsers.colon^3
                           * parsers.optionalspace
                           * (parsers.newline + parsers.eof)
```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```
10284       self.initialize_named_group("fenced_div_level", "0")
10285       self.initialize_named_group("fenced_div_num_opening_indents")
10286
10287       local function increment_div_level()
10288         local function push_indent_table(s, i, indent_table, -- luacheck: ignore s i
10289                                          fenced_div_num_opening_indents, fenced_div_l
10290           fenced_div_level = tonumber(fenced_div_level) + 1
10291           local num_opening_indents = 0
10292           if indent_table.indents ~= nil then
10293             num_opening_indents = #indent_table.indents
10294           end
10295           fenced_div_num_opening_indents[fenced_div_level] = num_opening_indents
10296           return true, fenced_div_num_opening_indents
10297         end
10298
10299         local function increment_level(s, i, fenced_div_level) -- luacheck: ignore s
10300           fenced_div_level = tonumber(fenced_div_level) + 1
10301           return true, tostring(fenced_div_level)
10302         end
10303
10304         return Cg( Cmt( Cb("indent_info")
10305                       * Cb("fenced_div_num_opening_indents")
10306                       * Cb("fenced_div_level"), push_indent_table)
10307                  , "fenced_div_num_opening_indents")
10308              * Cg( Cmt( Cb("fenced_div_level"), increment_level)
10309                  , "fenced_div_level")
10310       end
10311
10312       local function decrement_div_level()
10313         local function pop_indent_table(s, i, fenced_div_indent_table, fenced_div_lev
10314           fenced_div_level = tonumber(fenced_div_level)
10315           fenced_div_indent_table[fenced_div_level] = nil
10316           return true, tostring(fenced_div_level - 1)
10317         end
10318
10319         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
10320                       * Cb("fenced_div_level"), pop_indent_table)
10321                  , "fenced_div_level")
10322       end
10323
```

```
10324
10325       local non_fenced_div_block  = parsers.check_minimal_indent * V("Block")
10326                                   - parsers.check_minimal_indent_and_trail * fenced_d
10327
10328       local non_fenced_div_paragraph  = parsers.check_minimal_indent * V("Paragraph")
10329                                       - parsers.check_minimal_indent_and_trail * fenc
10330
10331       local blank = parsers.minimally_indented_blank
10332
10333       local block_separated  = parsers.block_sep_group(blank)
10334                              * non_fenced_div_block
10335
10336       local loop_body_pair  = parsers.create_loop_body_pair(block_separated,
10337                                                             non_fenced_div_paragraph,
10338                                                             parsers.block_sep_group(b
10339                                                             parsers.par_sep_group(bla
10340
10341       local content_loop  = ( non_fenced_div_block
10342                             * loop_body_pair.block^0
10343                             + non_fenced_div_paragraph
10344                             * block_separated
10345                             * loop_body_pair.block^0
10346                             + non_fenced_div_paragraph
10347                             * loop_body_pair.par^0)
10348                           * blank^0
10349
10350       local FencedDiv = fenced_div_begin
10351                       / function (infostring)
10352                           local attr = lpeg.match(Ct(parsers.attributes), infostring)
10353                           if attr == nil then
10354                             attr = {"." .. infostring}
10355                           end
10356                           return attr
10357                         end
10358                       / writer.div_begin
10359                       * increment_div_level()
10360                       * parsers.skipblanklines
10361                       * Ct(content_loop)
10362                       * parsers.minimally_indented_blank^0
10363                       * parsers.check_minimal_indent_and_trail * fenced_div_end
10364                       * decrement_div_level()
10365                       * (Cc("") / writer.div_end)
10366
10367       self.insert_pattern("Block after Verbatim",
10368                           FencedDiv, "FencedDiv")
10369
10370       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 function check_div_level(s, i, fenced_div_level) -- luacheck: ignore s
          fenced_div_level = tonumber(fenced_div_level)
          return fenced_div_level > 0
        end

        return Cmt(Cb("fenced_div_level"), check_div_level)
      end

      local function check_indent()
        local function compare_indent(s, i, indent_table, -- luacheck: ignore s i
                                      fenced_div_num_opening_indents, fenced_div_leve
          fenced_div_level = tonumber(fenced_div_level)
          local num_current_indents = (indent_table.current_line_indents ~= nil and
                                       #indent_table.current_line_indents) or 0
          local num_opening_indents = fenced_div_num_opening_indents[fenced_div_level
          return num_current_indents == num_opening_indents
        end

        return Cmt( Cb("indent_info")
                  * Cb("fenced_div_num_opening_indents")
                  * Cb("fenced_div_level"), compare_indent)
      end

      local fencestart = is_inside_div()
                       * fenced_div_end
                       * check_indent()

      if not blank_before_div_fence then
        self.update_rule("EndlineExceptions", function(previous_pattern)
          if previous_pattern == nil then
            previous_pattern = parsers.EndlineExceptions
          end
          return previous_pattern + fencestart
        end)
      end
    end
  }
end
```

3.1.7.8 Header Attributes

The extensions.header_attributes function implements the Pandoc header attribute syntax extension.

```
M.extensions.header_attributes = function()
  return {
    name = "built-in header_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local function strip_atx_end(s)
        return s:gsub("%s+#*%s*$","")
      end

      local AtxHeading = Cg(parsers.heading_start, "level")
                       * parsers.optionalspace
                       * (C(((parsers.linechar
                              - (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline))
                             * (parsers.linechar
                               - parsers.lbrace)^0)^1)
                             / strip_atx_end
                             / parsers.parse_heading_text)
                       * Cg(Ct(parsers.newline
                              + (parsers.attributes
                                * parsers.optionalspace
                                * parsers.newline)), "attributes")
                       * Cb("level")
                       * Cb("attributes")
                       / writer.heading

      local function strip_trailing_spaces(s)
        return s:gsub("%s*$","")
      end

      local heading_line  = (parsers.linechar
                            - (parsers.attributes
                              * parsers.optionalspace
                              * parsers.newline))^1
                          - parsers.thematic_break_lines

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

      local SetextHeading  = parsers.freeze_trail * parsers.check_trail_no_rem
```

```
                              * #(heading_text
                                 * (parsers.attributes
                                   * parsers.optionalspace
                                   * parsers.newline)^-1
                                 * parsers.check_minimal_indent * parsers.check_trail *
                              * 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
                              * 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.lineblo

      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.spnlc * parsers.url
                                      * ( parsers.spnlc_sep * parsers.title * (parsers.
                                        * parsers.only_blank
                                        + parsers.spnlc_sep * parsers.title * parsers.o
                                        + Cc("") * (parsers.spnlc * Ct(parsers.attribut
                                        + Cc("") * parsers.only_blank)
```

```
10570
10571       local ReferenceWithAttributes = define_reference_parser
10572                                     / self.register_link
10573
10574       self.update_rule("Reference", ReferenceWithAttributes)
10575
```

The following patterns define direct and indirect links with attributes.

```
10576
10577       local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),
10578                                       / self.defer_link_and_emphasis_processing
10579
10580       self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10581
```

The following patterns define autolinks with attributes.

```
10582       local AutoLinkUrlWithAttributes
10583                       = parsers.auto_link_url
10584                       * Ct(parsers.attributes)
10585                       / self.auto_link_url
10586
10587       self.insert_pattern("Inline before AutoLinkUrl",
10588                           AutoLinkUrlWithAttributes,
10589                           "AutoLinkUrlWithAttributes")
10590
10591       local AutoLinkEmailWithAttributes
10592                       = parsers.auto_link_email
10593                       * Ct(parsers.attributes)
10594                       / self.auto_link_email
10595
10596       self.insert_pattern("Inline before AutoLinkEmail",
10597                           AutoLinkEmailWithAttributes,
10598                           "AutoLinkEmailWithAttributes")
10599
10600       if options.relativeReferences then
10601
10602         local AutoLinkRelativeReferenceWithAttributes
10603                         = parsers.auto_link_relative_reference
10604                         * Ct(parsers.attributes)
10605                         / self.auto_link_url
10606
10607         self.insert_pattern(
10608           "Inline before AutoLinkRelativeReference",
10609           AutoLinkRelativeReferenceWithAttributes,
10610           "AutoLinkRelativeReferenceWithAttributes")
10611
10612       end
10613
```

```
    end
  }
end
```

3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

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

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

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

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

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

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

        local rawnotes = {}

        -- like indirect_link
```

```
10651         local function lookup_note(ref)
10652           return writer.defer_call(function()
10653             local found = rawnotes[self.normalize_tag(ref)]
10654             if found then
10655               return writer.note(
10656                 self.parser_functions.parse_blocks_nested(found))
10657             else
10658               return {"[",
10659                 self.parser_functions.parse_inlines("^" .. ref), "]"}
10660             end
10661           end)
10662         end
10663 
10664         local function register_note(ref,rawnote)
10665           local normalized_tag = self.normalize_tag(ref)
10666           if rawnotes[normalized_tag] == nil then
10667             rawnotes[normalized_tag] = rawnote
10668           end
10669           return ""
10670         end
10671 
10672         local NoteRef = RawNoteRef / lookup_note
10673 
10674         local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10675 
10676         local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10677 
10678         local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10679 
10680         local indented_blocks = function(bl)
10681           return Cs( bl
10682                 * (blank^1 * (parsers.check_optional_indent / "")
10683                   * parsers.check_code_trail * -parsers.blankline * bl)^0)
10684         end
10685 
10686         local NoteBlock
10687                     = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10688                     * parsers.spnlc * indented_blocks(chunk)
10689                     / register_note
10690 
10691         local Reference = NoteBlock + parsers.Reference
10692 
10693         self.update_rule("Reference", Reference)
10694         self.insert_pattern("Inline before LinkAndEmph",
10695                             NoteRef, "NoteRef")
10696       end
10697 
```

```
      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
```

```
10779               table.insert(buffer, "{")
10780             end
10781             table.insert(buffer, column)
10782             if i > 1 then
10783               table.insert(buffer, "}")
10784             end
10785           end
10786           table.insert(buffer, "}")
10787         end
10788         if attributes ~= nil then
10789           table.insert(buffer,
10790                        "\\markdownRendererTableAttributeContextEnd{}")
10791         end
10792         return buffer
10793       end
10794     end, extend_reader = function(self)
10795       local parsers = self.parsers
10796       local writer = self.writer
10797
10798       local table_hline_separator = parsers.pipe + parsers.plus
10799
10800       local table_hline_column = (parsers.dash
10801                                   - #(parsers.dash
10802                                       * (parsers.spacechar
10803                                         + table_hline_separator
10804                                         + parsers.newline)))^1
10805                                 * (parsers.colon * Cc("r")
10806                                   + parsers.dash * Cc("d"))
10807                                 + parsers.colon
10808                                 * (parsers.dash
10809                                   - #(parsers.dash
10810                                       * (parsers.spacechar
10811                                         + table_hline_separator
10812                                         + parsers.newline)))^1
10813                                 * (parsers.colon * Cc("c")
10814                                   + parsers.dash * Cc("l"))
10815
10816       local table_hline = pipe_table_row(false
10817                                          , table_hline_column
10818                                          , table_hline_separator
10819                                          , table_hline_column)
10820
10821       local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10822                                        * parsers.optionalspace * parsers.newline)^0
10823                                     * parsers.check_minimal_indent_and_trail
10824                                     * (P("Table")^-1 * parsers.colon)
10825                                     * parsers.optionalspace
```

```
10826
10827       local function strip_trailing_spaces(s)
10828         return s:gsub("%s*$","")
10829       end
10830
10831       local table_row = pipe_table_row(true
10832                                        , (C((parsers.linechar - parsers.pipe)^1)
10833                                          / strip_trailing_spaces
10834                                          / self.parser_functions.parse_inlines)
10835                                        , parsers.pipe
10836                                        , (C((parsers.linechar - parsers.pipe)^0)
10837                                          / strip_trailing_spaces
10838                                          / self.parser_functions.parse_inlines))
10839
10840       local table_caption
10841       if table_captions then
10842         table_caption = #table_caption_beginning
10843                       * table_caption_beginning
10844         if table_attributes then
10845           table_caption = table_caption
10846                         * (C(((( parsers.linechar
10847                                - (parsers.attributes
10848                                  * parsers.optionalspace
10849                                  * parsers.newline
10850                                  * -#( parsers.optionalspace
10851                                      * parsers.linechar)))
10852                               + ( parsers.newline
10853                                 * #( parsers.optionalspace
10854                                    * parsers.linechar)
10855                                 * C(parsers.optionalspace) / writer.space))
10856                              * (parsers.linechar
10857                                - parsers.lbrace)^0)^1)
10858                             / self.parser_functions.parse_inlines)
10859                         * (parsers.newline
10860                           + ( Ct(parsers.attributes)
10861                             * parsers.optionalspace
10862                             * parsers.newline))
10863         else
10864           table_caption = table_caption
10865                         * C(( parsers.linechar
10866                             + ( parsers.newline
10867                               * #( parsers.optionalspace
10868                                  * parsers.linechar)
10869                               * C(parsers.optionalspace) / writer.space))^1)
10870                         / self.parser_functions.parse_inlines
10871                         * parsers.newline
10872         end
```

```
10873       else
10874         table_caption = parsers.fail
10875       end
10876
10877       local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_inden
10878                         * table_hline * parsers.newline
10879                         * ((parsers.check_minimal_indent / {}) * table_row * parsers.
10880                       / make_pipe_table_rectangular
10881                       * table_caption^-1
10882                       / writer.table
10883
10884       self.insert_pattern("Block after Blockquote",
10885                           PipeTable, "PipeTable")
10886     end
10887   }
10888 end
```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
10889 M.extensions.raw_inline = function()
10890   return {
10891     name = "built-in raw_inline syntax extension",
10892     extend_writer = function(self)
10893       local options = self.options
10894
```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```
10895       function self.rawInline(s, attr)
10896         if not self.is_writing then return "" end
10897         if self.flatten_inlines then return s end
10898         local name = util.cache_verbatim(options.cacheDir, s)
10899         return {"\\markdownRendererInputRawInline{",
10900                 name,"}{", self.string(attr),"}"}
10901       end
10902     end, extend_reader = function(self)
10903       local writer = self.writer
10904
10905       local RawInline = parsers.inticks
10906                       * parsers.raw_attribute
10907                       / writer.rawInline
10908
10909       self.insert_pattern("Inline before Code",
10910                           RawInline, "RawInline")
10911     end
10912   }
```

```
10913 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
10914 M.extensions.strike_through = function()
10915   return {
10916     name = "built-in strike_through syntax extension",
10917     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.

```
10918       function self.strike_through(s)
10919         if self.flatten_inlines then return s end
10920         return {"\\markdownRendererStrikeThrough{",s,"}"}
10921       end
10922     end, extend_reader = function(self)
10923       local parsers = self.parsers
10924       local writer = self.writer
10925
10926       local StrikeThrough = (
10927         parsers.between(parsers.Inline, parsers.doubletildes,
10928                         parsers.doubletildes)
10929       ) / writer.strike_through
10930
10931       self.insert_pattern("Inline after LinkAndEmph",
10932                           StrikeThrough, "StrikeThrough")
10933
10934       self.add_special_character("~")
10935     end
10936   }
10937 end
```

3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
10938 M.extensions.subscripts = function()
10939   return {
10940     name = "built-in subscripts syntax extension",
10941     extend_writer = function(self)
```

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

```
10942       function self.subscript(s)
10943         if self.flatten_inlines then return s end
10944         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.

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

      local Superscript = (
        parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
      ) / writer.superscript

      self.insert_pattern("Inline after LinkAndEmph",
                          Superscript, "Superscript")

      self.add_special_character("^")
    end
  }
end
```

3.1.7.19 TEX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
M.extensions.tex_math = function(tex_math_dollars,
                                 tex_math_single_backslash,
                                 tex_math_double_backslash)
  return {
    name = "built-in tex_math syntax extension",
    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.

```
      function self.display_math(s)
        if self.flatten_inlines then return s end
        return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
      end
```

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

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

      local function between(p, starter, ender)
        return (starter * Cs(p * (p - ender)^0) * ender)
      end

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

      local allowed_before_closing = B( parsers.backslash * parsers.any
                                      + parsers.any * (parsers.any - parsers.backslas

      local allowed_before_closing_no_space = B( parsers.backslash * parsers.any
                                               + parsers.any * (parsers.nonspacechar
```

The following patterns implement the Pandoc dollar math syntax extension.

```
      local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
                                 + parsers.backslash^-1
                                 * parsers.linechar)
                                 - parsers.blankline^2
                                 - parsers.dollar
```

```
11021
11022       local inline_math_opening_dollars = parsers.dollar
11023                                         * #(parsers.nonspacechar)
11024
11025       local inline_math_closing_dollars = allowed_before_closing_no_space
11026                                         * parsers.dollar
11027                                         * -#(parsers.digit)
11028
11029       local inline_math_dollars = between(Cs( dollar_math_content),
11030                                           inline_math_opening_dollars,
11031                                           inline_math_closing_dollars)
11032
11033       local display_math_opening_dollars  = parsers.dollar
11034                                           * parsers.dollar
11035
11036       local display_math_closing_dollars  = parsers.dollar
11037                                           * parsers.dollar
11038
11039       local display_math_dollars = between(Cs( dollar_math_content),
11040                                            display_math_opening_dollars,
11041                                            display_math_closing_dollars)
```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```
11042       local backslash_math_content  = (parsers.newline * (parsers.check_optional_inde
11043                                     + parsers.linechar)
11044                                     - parsers.blankline^2
```

The following patterns implement the Pandoc double backslash math syntax extension.

```
11045       local inline_math_opening_double  = parsers.backslash
11046                                         * parsers.backslash
11047                                         * parsers.lparent
11048
11049       local inline_math_closing_double  = allowed_before_closing
11050                                         * parsers.spacechar^0
11051                                         * parsers.backslash
11052                                         * parsers.backslash
11053                                         * parsers.rparent
11054
11055       local inline_math_double  = between(Cs( backslash_math_content),
11056                                           inline_math_opening_double,
11057                                           inline_math_closing_double)
11058                                 / strip_preceding_whitespaces
11059
11060       local display_math_opening_double = parsers.backslash
11061                                         * parsers.backslash
11062                                         * parsers.lbracket
```

```
11063
11064       local display_math_closing_double = allowed_before_closing
11065                                         * parsers.spacechar^0
11066                                         * parsers.backslash
11067                                         * parsers.backslash
11068                                         * parsers.rbracket
11069
11070       local display_math_double = between(Cs( backslash_math_content),
11071                                           display_math_opening_double,
11072                                           display_math_closing_double)
11073                                 / strip_preceding_whitespaces
```

The following patterns implement the Pandoc single backslash math syntax extension.

```
11074       local inline_math_opening_single  = parsers.backslash
11075                                         * parsers.lparent
11076
11077       local inline_math_closing_single  = allowed_before_closing
11078                                         * parsers.spacechar^0
11079                                         * parsers.backslash
11080                                         * parsers.rparent
11081
11082       local inline_math_single  = between(Cs( backslash_math_content),
11083                                           inline_math_opening_single,
11084                                           inline_math_closing_single)
11085                                 / strip_preceding_whitespaces
11086
11087       local display_math_opening_single = parsers.backslash
11088                                         * parsers.lbracket
11089
11090       local display_math_closing_single = allowed_before_closing
11091                                         * parsers.spacechar^0
11092                                         * parsers.backslash
11093                                         * parsers.rbracket
11094
11095       local display_math_single = between(Cs( backslash_math_content),
11096                                           display_math_opening_single,
11097                                           display_math_closing_single)
11098                                 / strip_preceding_whitespaces
11099
11100       local display_math = parsers.fail
11101
11102       local inline_math = parsers.fail
11103
11104       if tex_math_dollars then
11105         display_math = display_math + display_math_dollars
11106         inline_math = inline_math + inline_math_dollars
11107       end
11108
```

```
      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
        self.add_special_character("$")
      end

      if tex_math_single_backslash or tex_math_double_backslash then
        self.add_special_character("\\")
        self.add_special_character("[")
        self.add_special_character("]")
        self.add_special_character(")")
        self.add_special_character("(")
      end
    end
  }
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.

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

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t`.

```
11143       function self.jekyllData(d, t, p)
11144         if not self.is_writing then return "" end
11145
11146         local buf = {}
11147
11148         local keys = {}
11149         for k, _ in pairs(d) do
11150           table.insert(keys, k)
11151         end
11152         table.sort(keys)
11153
11154         if not p then
11155           table.insert(buf, "\\markdownRendererJekyllDataBegin")
11156         end
11157
11158         if #d > 0 then
11159             table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
11160             table.insert(buf, self.identifier(p or "null"))
11161             table.insert(buf, "}{")
11162             table.insert(buf, #keys)
11163             table.insert(buf, "}")
11164         else
11165             table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11166             table.insert(buf, self.identifier(p or "null"))
11167             table.insert(buf, "}{")
11168             table.insert(buf, #keys)
11169             table.insert(buf, "}")
11170         end
11171
11172         for _, k in ipairs(keys) do
11173           local v = d[k]
11174           local typ = type(v)
11175           k = tostring(k or "null")
11176           if typ == "table" and next(v) ~= nil then
11177             table.insert(
11178               buf,
11179               self.jekyllData(v, t, k)
11180             )
11181           else
11182             k = self.identifier(k)
11183             v = tostring(v)
11184             if typ == "boolean" then
11185               table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11186               table.insert(buf, k)
11187               table.insert(buf, "}{")
11188               table.insert(buf, v)
11189               table.insert(buf, "}")
```

```
11190             elseif typ == "number" then
11191               table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11192               table.insert(buf, k)
11193               table.insert(buf, "}{")
11194               table.insert(buf, v)
11195               table.insert(buf, "}")
11196             elseif typ == "string" then
11197               table.insert(buf, "\\markdownRendererJekyllDataString{")
11198               table.insert(buf, k)
11199               table.insert(buf, "}{")
11200               table.insert(buf, t(v))
11201               table.insert(buf, "}")
11202             elseif typ == "table" then
11203               table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11204               table.insert(buf, k)
11205               table.insert(buf, "}")
11206             else
11207               error(format("Unexpected type %s for value of " ..
11208                            "YAML key %s", typ, k))
11209             end
11210           end
11211         end
11212
11213         if #d > 0 then
11214           table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
11215         else
11216           table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
11217         end
11218
11219         if not p then
11220           table.insert(buf, "\\markdownRendererJekyllDataEnd")
11221         end
11222
11223         return buf
11224       end
11225     end, extend_reader = function(self)
11226       local parsers = self.parsers
11227       local writer = self.writer
11228
11229       local JekyllData
11230                   = Cmt( C((parsers.line - P("---") - P("..."))^0)
11231                        , function(s, i, text) -- luacheck: ignore s i
11232                            local data
11233                            local ran_ok, _ = pcall(function()
11234                              -- TODO: Replace with `require("tinyyaml")` in TeX Liv
11235                              local tinyyaml = require("markdown-tinyyaml")
11236                              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
                    * #(-parsers.blankline)  -- if followed by blank, it's thematic b
                    * JekyllData
                    * (P("---") + P("..."))

      local ExpectedJekyllData
                    = ( P("---")
                      * parsers.blankline / 0
                      * #(-parsers.blankline)  -- if followed by blank, it's thematic
                      )^-1
                    * JekyllData
                    * (P("---") + P("..."))^-1

      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 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.

```
11275   if options.singletonCache and singletonCache.convert then
11276     for k, v in pairs(defaultOptions) do
11277       if type(v) == "table" then
11278         for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11279           if singletonCache.options[k][i] ~= options[k][i] then
11280             goto miss
11281           end
11282         end
11283       elseif singletonCache.options[k] ~= options[k] then
11284         goto miss
11285       end
11286     end
11287     return singletonCache.convert
11288   end
11289   ::miss::
```

Apply built-in syntax extensions based on `options`.

```
11290   local extensions = {}
11291
11292   if options.bracketedSpans then
11293     local bracketed_spans_extension = M.extensions.bracketed_spans()
11294     table.insert(extensions, bracketed_spans_extension)
11295   end
11296
11297   if options.contentBlocks then
11298     local content_blocks_extension = M.extensions.content_blocks(
11299       options.contentBlocksLanguageMap)
11300     table.insert(extensions, content_blocks_extension)
11301   end
11302
11303   if options.definitionLists then
11304     local definition_lists_extension = M.extensions.definition_lists(
11305       options.tightLists)
11306     table.insert(extensions, definition_lists_extension)
11307   end
11308
11309   if options.fencedCode then
11310     local fenced_code_extension = M.extensions.fenced_code(
11311       options.blankBeforeCodeFence,
11312       options.fencedCodeAttributes,
11313       options.rawAttribute)
11314     table.insert(extensions, fenced_code_extension)
11315   end
11316
11317   if options.fencedDivs then
11318     local fenced_div_extension = M.extensions.fenced_divs(
11319       options.blankBeforeDivFence)
11320     table.insert(extensions, fenced_div_extension)
```

```
11321   end
11322
11323   if options.headerAttributes then
11324     local header_attributes_extension = M.extensions.header_attributes()
11325     table.insert(extensions, header_attributes_extension)
11326   end
11327
11328   if options.inlineCodeAttributes then
11329     local inline_code_attributes_extension =
11330       M.extensions.inline_code_attributes()
11331     table.insert(extensions, inline_code_attributes_extension)
11332   end
11333
11334   if options.jekyllData then
11335     local jekyll_data_extension = M.extensions.jekyll_data(
11336       options.expectJekyllData)
11337     table.insert(extensions, jekyll_data_extension)
11338   end
11339
11340   if options.linkAttributes then
11341     local link_attributes_extension =
11342       M.extensions.link_attributes()
11343     table.insert(extensions, link_attributes_extension)
11344   end
11345
11346   if options.lineBlocks then
11347     local line_block_extension = M.extensions.line_blocks()
11348     table.insert(extensions, line_block_extension)
11349   end
11350
11351   if options.mark then
11352     local mark_extension = M.extensions.mark()
11353     table.insert(extensions, mark_extension)
11354   end
11355
11356   if options.pipeTables then
11357     local pipe_tables_extension = M.extensions.pipe_tables(
11358       options.tableCaptions, options.tableAttributes)
11359     table.insert(extensions, pipe_tables_extension)
11360   end
11361
11362   if options.rawAttribute then
11363     local raw_inline_extension = M.extensions.raw_inline()
11364     table.insert(extensions, raw_inline_extension)
11365   end
11366
11367   if options.strikeThrough then
```

```
11368     local strike_through_extension = M.extensions.strike_through()
11369     table.insert(extensions, strike_through_extension)
11370   end
11371
11372   if options.subscripts then
11373     local subscript_extension = M.extensions.subscripts()
11374     table.insert(extensions, subscript_extension)
11375   end
11376
11377   if options.superscripts then
11378     local superscript_extension = M.extensions.superscripts()
11379     table.insert(extensions, superscript_extension)
11380   end
11381
11382   if options.texMathDollars or
11383      options.texMathSingleBackslash or
11384      options.texMathDoubleBackslash then
11385     local tex_math_extension = M.extensions.tex_math(
11386       options.texMathDollars,
11387       options.texMathSingleBackslash,
11388       options.texMathDoubleBackslash)
11389     table.insert(extensions, tex_math_extension)
11390   end
11391
11392   if options.notes or options.inlineNotes then
11393     local notes_extension = M.extensions.notes(
11394       options.notes, options.inlineNotes)
11395     table.insert(extensions, notes_extension)
11396   end
11397
11398   if options.citations then
11399     local citations_extension = M.extensions.citations(options.citationNbsps)
11400     table.insert(extensions, citations_extension)
11401   end
11402
11403   if options.fancyLists then
11404     local fancy_lists_extension = M.extensions.fancy_lists()
11405     table.insert(extensions, fancy_lists_extension)
11406   end
```

Apply user-defined syntax extensions based on `options.extensions`.

```
11407   for _, user_extension_filename in ipairs(options.extensions) do
11408     local user_extension = (function(filename)
```

First, load and compile the contents of the user-defined syntax extension.

```
11409       local pathname = kpse.lookup(filename)
11410       local input_file = assert(io.open(pathname, "r"),
11411         [[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
      .. [[, which is incompatible]])

    assert(user_extension.grammar_version ~= nil,
      [[User-defined syntax extension "]] .. pathname
      .. [[" does not specify mandatory field "grammar_version"]])
    assert(type(user_extension.grammar_version) == "number",
      [[User-defined syntax extension "]] .. pathname
      .. [[" specifies field "grammar_version" of type "]]
      .. type(user_extension.grammar_version)
      .. [[" but "number" was expected]])
    assert(user_extension.grammar_version == metadata.grammar_version,
      [[User-defined syntax extension "]] .. pathname
      .. [[" uses grammar version "]] .. user_extension.grammar_version
      .. [[ but markdown.lua ]] .. metadata.version
      .. [[ uses grammar version ]] .. metadata.grammar_version
      .. [[, which is incompatible]])

    assert(user_extension.finalize_grammar ~= nil,
      [[User-defined syntax extension "]] .. pathname
      .. [[" does not specify mandatory "finalize_grammar" field]])
```

```
      assert(type(user_extension.finalize_grammar) == "function",
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "finalize_grammar" of type "]]
        .. type(user_extension.finalize_grammar)
        .. [[" 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.)

```
      local extension = {
        name = [[user-defined "]] .. pathname .. [[" syntax extension]],
        extend_reader = user_extension.finalize_grammar,
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

Produce a conversion function from markdown to plain TEX.

```
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  local convert = reader.finalize_grammar(extensions)
```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```
  collectgarbage("collect")
```

Update the singleton cache.

```
  if options.singletonCache then
    local singletonCacheOptions = {}
    for k, v in pairs(options) do
      singletonCacheOptions[k] = v
    end
    setmetatable(singletonCacheOptions,
      { __index = function (_, key)
        return defaultOptions[key] end })
    singletonCache.options = singletonCacheOptions
    singletonCache.convert = convert
  end
```

Return the conversion function from markdown to plain TEX.

```
  return convert
end

return M
```

3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```
11491
11492 local input
11493 if input_filename then
11494   local input_file = assert(io.open(input_filename, "r"),
11495     [[Could not open file "]] .. input_filename .. [[" for reading]])
11496   input = assert(input_file:read("*a"))
11497   assert(input_file:close())
11498 else
11499   input = assert(io.read("*a"))
11500 end
11501
```

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

```
11502 local lfs = require("lfs")
11503 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11504   assert(lfs.mkdir(options["cacheDir"]))
11505 end
```

If Kpathsea has not been loaded before or if LuaTEX has not yet been initialized, configure Kpathsea on top of loading it.

```
11506 local kpse
11507 (function()
11508   local should_initialize = package.loaded.kpse == nil
11509                         or tex.initialize ~= nil
11510   kpse = require("kpse")
11511   if should_initialize then
11512     kpse.set_program_name("luatex")
11513   end
11514 end)()
11515 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.

```
11516 if metadata.version ~= md.metadata.version then
11517   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11518        "markdown.lua " .. md.metadata.version .. ".")
11519 end
11520 local convert = md.new(options)
11521 local output = convert(input)
11522
11523 if output_filename then
11524   local output_file = assert(io.open(output_filename, "w"),
11525     [[Could not open file "]] .. output_filename .. [[" for writing]])
11526   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. Futhermore, 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{}%
```

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_renderer_prototype:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
  {
    \str_case:nn
```

```
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
      { #1 }
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block_renderer_prototype: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 }
}
\def\markdownRendererJekyllDataStringPrototype#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 2_ε `\filecontents` macro to plain TeX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

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

```
    |markdownIfOption{frozenCache}{}{@
      |immediate|openout|markdownOutputFileStream@
        |markdownOptionInputTempFileName|relax@
      |markdownInfo{@
        Buffering block-level markdown input into the temporary @
        input file "|markdownOptionInputTempFileName" and scanning @
        for the closing token sequence "#1"}@
    }@
```

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

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

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

```
    |def|markdownReadAndConvertStripPercentSign##1{@
      |markdownIfOption{stripPercentSigns}{@
        |if##1%@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
        |else@
          |expandafter|expandafter|expandafter@
            |markdownReadAndConvertProcessLine@
            |expandafter|expandafter|expandafter##1@
        |fi@
      }{@
        |expandafter@
          |markdownReadAndConvertProcessLine@
          |expandafter##1@
      }@
    }@
```

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

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

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

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

When the ending token sequence appears in the line, make the next newline character close the inputTempFileName file, return the character categories back to the former state, convert the inputTempFileName file from markdown to plain TEX, \input the result of the conversion, and expand the ending control sequence.

```
        |def^^M{@
          |markdownInfo{The ending token sequence was found}@
          |markdownIfOption{frozenCache}{}{@
```

```
        |immediate|closeout|markdownOutputFileStream@
      }@
      |endgroup@
      |markdownInput{@
        |markdownOptionOutputDir@
        /|markdownOptionInputTempFileName@
      }@
      #2}@
  |fi@
```

Repeat with the next line.

```
  ^^M}@
```

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

```
|catcode`|^^I=13@
|def^^I{|markdownReadAndConvertTab}@
```

Make the newline character active at expansion time and make it consume the rest of the line on expansion. Throw away the rest of the first line and pass the second line to the `\markdownReadAndConvertProcessLine` macro.

```
|catcode`|^^M=13@
|def^^M##1^^M{@
  |def^^M####1^^M{@
    |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
  ^^M}@
^^M}@
```

Reset the character categories back to the former state.

```
|endgroup
```

Use the lt3luabridge library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownLuaExecute
  #1
  {
    \int_compare:nNnT
      { \g_luabridge_method_int }
      =
      { \c_luabridge_method_shell_int }
      {
        \sys_if_shell_unrestricted:F
          {
            \sys_if_shell:TF
              {
                \msg_error:nn
                  { markdown }
```

```
                  { restricted-shell-access }
              }
              {
                \msg_error:nn
                  { markdown }
                  { disabled-shell-access }
              }
          }
      }
    \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.

```
\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|markdownInput#1{%
```

Change the category code of the percent sign (`%`) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```
    |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.

```
12256     |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding \markdownFrozenCache⟨*number*⟩ macro, and increment frozenCacheCounter.

```
12257     |markdownIfOption{frozenCache}{%
12258       |ifnum|markdownOptionFrozenCacheCounter=0|relax
12259         |markdownInfo{Reading frozen cache from
12260           "|markdownOptionFrozenCacheFileName"}%
12261         |input|markdownOptionFrozenCacheFileName|relax
12262       |fi
12263       |markdownInfo{Including markdown document number
12264         "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
12265       |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
12266       |global|advance|markdownOptionFrozenCacheCounter by 1|relax
12267     }{%
12268       |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.

```
12269       |openin|markdownInputFileStream&1
12270       |closein|markdownInputFileStream
12271       |markdownPrepareLuaOptions
12272       |markdownLuaExecute{%
12273         |markdownPrepare
12274         local file = assert(io.open("&1", "r"),
12275           [[Could not open file "&1" for reading]])
12276         local input = assert(file:read("*a"))
12277         assert(file:close())
12278         print(convert(input))
12279         |markdownCleanup}%
```

If we are finalizing the frozen cache, increment frozenCacheCounter.

```
12280       |markdownIfOption{finalizeCache}{%
12281         |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
12282     }%
12283     |endgroup
12284   }%
12285 |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 implemenation 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
  { markdown }
  {
```

In our implementation of the `markdown` LaTeX environment, we want to distinguish between the following two cases:

```
\begin{markdown} [smartEllipses]
% This is an optional argument ^
% ...
\end{markdown}
```

```
\begin{markdown}
  [smartEllipses]
% ^ This is link
\end{markdown}
```

Therefore, we cannot use the built-in LaTeX support for environments with optional arguments or packages such as xparse. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` LaTeX environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by TeX via the `\endlinechar` plain TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
    \group_begin:
    \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
    \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
    \peek_regex_replace_once:nnF
      { \ *\[\r*([^]]*)\][^\r]* }
      {
```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.

```
        \c { group_end: }
        \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
        \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the LaTeX environment.

```
        \c { markdownReadAndConvert@markdown } { }
      }
      {
        \group_end:
        \markdownReadAndConvert@markdown { }
      }
  }
  { \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.
```

```
  }
\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@markdown#1<%
    |markdownReadAndConvert<\end{markdown#1}>%
                           <|end<markdown#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. Futhermore, 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 }
    \tl_set:Nn
      \l_tmpa_tl
      {
        \plpartopsep=\partopsep
        \pltopsep=\topsep
      }
    \@@_if_option:nTF
      { startNumber }
      {
        \tl_put_right:Nn
          \l_tmpa_tl
          { \begin{compactenum} }
      }
      {
        \tl_put_right: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_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} }
        }
        { \markdownInput{#3} }
    },
  },
}
\ExplSyntaxOff
```

[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.

```
\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
          }
          {
```

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

```
            \ltx@ifpackageloaded
              { listings }
              { \lstinputlisting[language=\l_tmpa_tl]{#1} }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
                { \markdownRendererInputFencedCode{#1}{}{} }
            }
        }
    }
\ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
\ExplSyntaxOn
\def\markdownLATEXStrongEmphasis#1{%
  \str_if_in:NnTF
    \f@series
    { b }
    { \textnormal{#1} }
    { \textbf{#1} }
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={strongEmphasis={%
  \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
\@ifundefined{chapter}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}},
    headingFive = {\subparagraph{#1}}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
}%
```

3.3.4.2 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
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
```

[35] See https://tug.org/tex4ht/.

```
  }
}
```

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}%  Without prenotes and 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}%  Without prenotes and 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}%  Without prenotes and 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_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
```

```
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block_renderer_prototype: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. Futhermore, 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_renderer_prototype:nn
              { #1 }
              { context }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { context }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block_renderer_prototype: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