A Markdown Interpreter for TeX

Vít Novotný (based on the work of John MacFarlane and Hans Hagen)
witiko@mail.muni.cz

Version 2.1.2
September 14, 2016

Contents

1 Introduction

This document is a reference manual for the Markdown package. It is split into three sections. This section explains the purpose and the background of the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package along with usage notes and examples. It is aimed at the user of the package. Section 3 describes the implementation of the package. It is aimed at the developer of the package and the curious user.

1.1 About Markdown

The Markdown package provides facilities for the conversion of markdown markup to plain TeX. These are provided both in the form of a Lua module and in the form of plain TeX, LaTeX, and ConTeXt macro packages that enable the direct inclusion of markdown documents inside TeX documents.

Architecturally, the package consists of the Lunamark v1.4.0 Lua module by John MacFarlane, which was slimmed down and rewritten for the needs of the package. On top of Lunamark sits code for the plain TeX, LaTeX, and ConTeXt formats by Vít Novotný.

```
1 local metadata = {
2     version   = "2.1.2",
3     comment   = "A module for the conversion from markdown to plain TeX",
4     author    = "John MacFarlane, Hans Hagen, Vít Novotný",
```

```
    copyright = "2009-2016 John MacFarlane, Hans Hagen; 2016 Vít Novotný",
    license   = "LPPL 1.3"
}
if not modules then modules = { } end
modules['markdown'] = metadata
```

1.2 Feedback

Please use the markdown project page on GitHub[1] to report bugs and submit feature requests. Before making a feature request, please ensure that you have thoroughly studied this manual. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question on the TEX-LATEX Stack Exchange[2].

1.3 Acknowledgements

I would like to thank the Faculty of Informatics at the Masaryk University in Brno for providing me with the opportunity to work on this package alongside my studies. I would also like to thank the creator of the Lunamark Lua module, John Macfarlane, for releasing Lunamark under a permissive license that enabled its inclusion into the package.

The TEX part 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.

1.4 Prerequisites

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

1.4.1 Lua Prerequisites

The Lua part of the package requires that the following Lua modules are available from within the LuaTEX engine:

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

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

[1] https://github.com/witiko/markdown/issues
[2] https://tex.stackexchange.com

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

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

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

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

All the abovelisted modules are statically linked into the current version of the LuaTeX engine (see [1, Section 3.3]).

1.4.2 Plain TeX Prerequisites

The plain TeX part of the package requires that the plain TeX format (or its superset) is loaded, all the Lua prerequisites (see Section 1.4.1) and the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the `\markdownOptionCacheDir` macro before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive $\geq$ 2008).

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers (see [1, Section 3.2]).

The Lua File System module is statically linked into the LuaTeX engine (see [1, Section 3.3]).

The plain TeX part of the package also requires that either the LuaTeX `\directlua` primitive or the shell access file stream 18 is available.

1.4.3 LaTeX Prerequisites

The LaTeX part of the package requires that the LaTeX 2_ε format is loaded,

```
13 \NeedsTeXFormat{LaTeX2e}%
```

all the plain TeX prerequisites (see Section 1.4.2), and the following LaTeX 2_ε packages:

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

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

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

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

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

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

1.4.4 ConTEXt prerequisites

The ConTEXt part of the package requires that either the Mark II or the Mark IV format is loaded and all the plain TEX prerequisites (see Section 1.4.2).

2 User Guide

This part of the manual 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 purely abstract. It serves as a means of structuring this manual and as a promise to the user that if they only access the package through the interfaces, the future versions of the package should remain backwards compatible.

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain 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.

```
14 local M = {}
```

2.1.1 Conversion from Markdown to Plain TEX

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

The following example Lua code converts the markdown string `_Hello world!_` to a 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 Options

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

```
15 local defaultOptions = {}
```

`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.

```
16 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.

```
17 defaultOptions.blankBeforeCodeFence = false
```

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

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

```
18 defaultOptions.blankBeforeHeading = false
```

`cacheDir`=⟨*directory*⟩ default: .

The path to the directory containing auxiliary cache files.

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.

```
19 defaultOptions.cacheDir = "."
```

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

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

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

```
20 defaultOptions.citationNbsps = true
```

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

`true` Enable the pandoc citation syntax extension:

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

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

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

`false` Disable the pandoc citation syntax extension.

```
21 defaultOptions.citations = 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.

```
22 defaultOptions.definitionLists = 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.

```
23 defaultOptions.hashEnumerators = false
```

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

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

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

```
24 defaultOptions.hybrid = false
```

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

`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>
  ```
```

`true` Disable the commonmark fenced code block extension.

```
25 defaultOptions.fencedCode = false
```

`footnotes`=true, `false` default: `false`

`true` Enable the pandoc footnote syntax extension:

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

[^1]: Here is the footnote.

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

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

        { some.code }

    The whole paragraph can be indented, or just the
    first line.  In this way, multi-paragraph footnotes
```

```
    work like multi-paragraph list items.

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

`false` Disable the pandoc footnote syntax extension.

```
defaultOptions.footnotes = false
```

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

`true` Preserve all tabs in the input.

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

```
defaultOptions.preserveTabs = false
```

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

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

`false` Preserve all ellipses in the input.

```
defaultOptions.smartEllipses = false
```

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

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

`false` Ignore the number in the items of ordered lists. Each item will only produce a `\markdownRendererOlItem` TEX macro.

```
defaultOptions.startNumber = true
```

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

`true` Lists whose bullets do not consist of multiple paragraphs will be detected and passed to the `\markdownRendererOlBeginTight`, `\markdownRendererOlEndTight`, `\markdownRendererUlBeginTight`, `\markdownRendererUlEndTight`, `\markdownRendererDlBeginTight`, and `\markdownRendererDlEndTight` macros.

`false` Lists whose bullets do not consist of multiple paragraphs will be treated the same way as lists that do.

```
defaultOptions.tightLists = true
```

2.2 Plain TEX Interface

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

```
\def\markdownLastModified{2016/09/13}%
\def\markdownVersion{2.1.2}%
```

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`, and `\markdownInput` 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 otherwise. 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 (see [2, p. 46]). As a corrolary, the `\markdownBegin` macro also ignores them.

The `\markdownBegin` and `\markdownEnd` macros will also consume the rest of the lines at which they appear. In the following example plain TEX code, the characters `c`, `e`, and `f` will not appear in the output.

```
\input markdown
a
b \markdownBegin c
```

```
d
e \markdownEnd   f
g
\bye
```

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

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

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

The \markdownInput macro accepts a single parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TEX.

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

This macro is not subject to the abovelisted 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
```

2.2.2 Options

The plain TEX options are represented by TEX macros. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.2), while some of them are specific to the plain TEX interface.

2.2.2.1 File and directory names

The \markdownOptionHelperScriptFileName macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain TEX in TEX engines without the \directlua primitive. It defaults to \jobname.markdown.lua, where \jobname is the base name of the document being typeset.

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

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

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the conversion from markdown to plain TEX in TEX engines without the `\directlua` primitive. It defaults to `\jobname.markdown.out`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

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

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain TEX in TEX engines without the `\directlua` primitive. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

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

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

```
\def\markdownOptionCacheDir{./_markdown_\jobname}%
```

2.2.2.2 Lua Interface Options

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

```
\let\markdownOptionBlankBeforeBlockquote\undefined
\let\markdownOptionBlankBeforeCodeFence\undefined
\let\markdownOptionBlankBeforeHeading\undefined
\let\markdownOptionCitations\undefined
\let\markdownOptionCitationNbsps\undefined
\let\markdownOptionDefinitionLists\undefined
\let\markdownOptionFootnotes\undefined
\let\markdownOptionFencedCode\undefined
\let\markdownOptionHashEnumerators\undefined
\let\markdownOptionHybrid\undefined
\let\markdownOptionPreserveTabs\undefined
\let\markdownOptionSmartEllipses\undefined
\let\markdownOptionStartNumber\undefined
\let\markdownOptionTightLists\undefined
```

2.2.3 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 2.2.4).

2.2.3.1 Interblock Separator Renderer

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

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

2.2.3.2 Line Break Renderer

The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

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

2.2.3.3 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 `true`. The macro receives no arguments.

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

2.2.3.4 Non-breaking Space Renderer

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

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

2.2.3.5 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}%
\def\markdownRendererRightBrace{%
  \markdownRendererRightBracePrototype}%
\def\markdownRendererDollarSign{%
  \markdownRendererDollarSignPrototype}%
\def\markdownRendererPercentSign{%
  \markdownRendererPercentSignPrototype}%
\def\markdownRendererAmpersand{%
  \markdownRendererAmpersandPrototype}%
\def\markdownRendererUnderscore{%
  \markdownRendererUnderscorePrototype}%
\def\markdownRendererHash{%
```

```
  \markdownRendererHashPrototype}%
\def\markdownRendererCircumflex{%
  \markdownRendererCircumflexPrototype}%
\def\markdownRendererBackslash{%
  \markdownRendererBackslashPrototype}%
\def\markdownRendererTilde{%
  \markdownRendererTildePrototype}%
\def\markdownRendererPipe{%
  \markdownRendererPipePrototype}%
```

2.2.3.6 Code Span Renderer

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

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

2.2.3.7 Link Renderer

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

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

2.2.3.8 Image Renderer

The `\markdownRendererImage` macro represents an image. It receives four 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}%
```

2.2.3.9 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}%
```

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 `false`. The macro receives no arguments.

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

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

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

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

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

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}%
```

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 `false`. The macro receives no arguments.

```
\def\markdownRendererUlEndTight{%
  \markdownRendererUlEndTightPrototype}%
```

2.2.3.10 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). The macro receives no arguments.

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

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 `false`. The macro receives no arguments.

```
\def\markdownRendererOlBeginTight{%
  \markdownRendererOlBeginTightPrototype}%
```

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

```
\def\markdownRendererOlItem{%
  \markdownRendererOlItemPrototype}%
```

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

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

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

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

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

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

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 `false`. The macro receives no arguments.

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

2.2.3.11 Definition List Renderers

The following macros are only produces, when the `definitionLists` option is `true`.

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}%
```

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

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

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}%
```

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

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

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}%
```

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}%
```

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}%
```

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

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

2.2.3.12 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}%
```

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}%
```

2.2.3.13 Block Quote Renderers

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

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

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

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

2.2.3.14 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}%
```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is `true`. The macro receives two arguments that correspond to the filename of a file contaning the code block contents and to the code fence infostring.

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

2.2.3.15 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}%
```

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

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

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

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

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

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

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

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

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

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

2.2.3.16 Horizontal Rule Renderer

The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

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

2.2.3.17 Footnote Renderer

The `\markdownRendererFootnote` macro represents a footnote. This macro will only be produced, when the `footnotes` option is `true`. The macro receives a single argument that corresponds to the footnote text.

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

2.2.3.18 Parenthesized Citations Renderer

The `\markdownRendererCite` macro represents a string of one or more parenthetical citations. This macro will only be produced, when the `citations` option is `true`. 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}%
```

2.2.3.19 Text Citations Renderer

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

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

2.2.4 Token Renderer Prototypes

The following TEX macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the LATEX and ConTEXt implementations (see sections 3.3 and 3.4).

```
\def\markdownRendererInterblockSeparatorPrototype{}%
\def\markdownRendererLineBreakPrototype{}%
\def\markdownRendererEllipsisPrototype{}%
\def\markdownRendererNbspPrototype{}%
\def\markdownRendererLeftBracePrototype{}%
\def\markdownRendererRightBracePrototype{}%
\def\markdownRendererDollarSignPrototype{}%
\def\markdownRendererPercentSignPrototype{}%
```

```
\def\markdownRendererAmpersandPrototype{}%
\def\markdownRendererUnderscorePrototype{}%
\def\markdownRendererHashPrototype{}%
\def\markdownRendererCircumflexPrototype{}%
\def\markdownRendererBackslashPrototype{}%
\def\markdownRendererTildePrototype{}%
\def\markdownRendererPipePrototype{}%
\def\markdownRendererCodeSpanPrototype#1{}%
\def\markdownRendererLinkPrototype#1#2#3#4{}%
\def\markdownRendererImagePrototype#1#2#3#4{}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{}%
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{}%
\def\markdownRendererStrongEmphasisPrototype#1{}%
\def\markdownRendererBlockQuoteBeginPrototype{}%
\def\markdownRendererBlockQuoteEndPrototype{}%
\def\markdownRendererInputVerbatimPrototype#1{}%
\def\markdownRendererInputFencedCodePrototype#1#2{}%
\def\markdownRendererHeadingOnePrototype#1{}%
\def\markdownRendererHeadingTwoPrototype#1{}%
\def\markdownRendererHeadingThreePrototype#1{}%
\def\markdownRendererHeadingFourPrototype#1{}%
\def\markdownRendererHeadingFivePrototype#1{}%
\def\markdownRendererHeadingSixPrototype#1{}%
\def\markdownRendererHorizontalRulePrototype{}%
\def\markdownRendererFootnotePrototype#1{}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
```

2.2.5 Logging Facilities

The \markdownInfo, \markdownWarning, and \markdownError macros provide access to logging to the rest of the macros. Their first argument specifies the text of the info, warning, or error message.

```
\def\markdownInfo#1{}%
\def\markdownWarning#1{}%
```

The \markdownError macro receives a second argument that provides a help text suggesting a remedy to the error.

```
\def\markdownError#1{}%
```

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

2.2.6 Miscellanea

The \markdownLuaRegisterIBCallback and \markdownLuaUnregisterIBCallback macros specify the Lua code for registering and unregistering a callback for changing the contents of the line input buffer before a TEX engine that supports direct Lua access via the \directlua macro starts looking at it. The first argument of the \markdownLuaRegisterIBCallback macro corresponds to the callback function being registered.

Local members defined within \markdownLuaRegisterIBCallback are guaranteed to be visible from \markdownLuaUnregisterIBCallback and the execution of the two macros alternates, so it is not necessary to consider the case, when one of the macros is called twice in a row.

```
\def\markdownLuaRegisterIBCallback#1{%
  local old_callback = callback.find("process_input_buffer")
  callback.register("process_input_buffer", #1)}%
\def\markdownLuaUnregisterIBCallback{%
  callback.register("process_input_buffer", old_callback)}%
```

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 the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.5), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

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

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

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

```
\ifx\markdownMode\undefined
  \ifx\directlua\undefined
    \def\markdownMode{0}%
  \else
    \def\markdownMode{2}%
  \fi
\fi
```

2.3 LATEX Interface

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

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

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

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

2.3.1 Typesetting Markdown

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

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

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

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

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown}
_Hello_ **world** ...
\end{markdown}
% ...
\end{document}
```

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown*}{smartEllipses}
_Hello_ **world** ...
\end{markdown*}
% ...
\end{document}
```

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 $\langle\langle key\rangle=\langle value\rangle\rangle$ pairs. For boolean options, the $\langle=\langle value\rangle\rangle$ part is optional, and $\langle\langle key\rangle\rangle$ will be interpreted as $\langle\langle key\rangle=true\rangle$.

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

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

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

2.3.2.1 Plain TeX Interface Options

The following options map directly to the option macros exposed by the plain TeX interface (see Section 2.2.2).

```
\RequirePackage{keyval}
\define@key{markdownOptions}{helperScriptFileName}{%
  \def\markdownOptionHelperScriptFileName{#1}}%
\define@key{markdownOptions}{inputTempFileName}{%
  \def\markdownOptionInputTempFileName{#1}}%
\define@key{markdownOptions}{outputTempFileName}{%
  \def\markdownOptionOutputTempFileName{#1}}%
\define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
  \def\markdownOptionBlankBeforeBlockquote{#1}}%
\define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
  \def\markdownOptionBlankBeforeCodeFence{#1}}%
\define@key{markdownOptions}{blankBeforeHeading}[true]{%
  \def\markdownOptionBlankBeforeHeading{#1}}%
\define@key{markdownOptions}{citations}[true]{%
```

```
  \def\markdownOptionCitations{#1}%
\define@key{markdownOptions}{citationNbsps}[true]{%
  \def\markdownOptionCitationNbsps{#1}%
\define@key{markdownOptions}{cacheDir}{%
  \def\markdownOptionCacheDir{#1}%
\define@key{markdownOptions}{definitionLists}[true]{%
  \def\markdownOptionDefinitionLists{#1}%
\define@key{markdownOptions}{footnotes}[true]{%
  \def\markdownOptionFootnotes{#1}%
\define@key{markdownOptions}{fencedCode}[true]{%
  \def\markdownOptionFencedCode{#1}%
\define@key{markdownOptions}{hashEnumerators}[true]{%
  \def\markdownOptionHashEnumerators{#1}%
\define@key{markdownOptions}{hybrid}[true]{%
  \def\markdownOptionHybrid{#1}%
\define@key{markdownOptions}{preserveTabs}[true]{%
  \def\markdownOptionPreserveTabs{#1}%
\define@key{markdownOptions}{smartEllipses}[true]{%
  \def\markdownOptionSmartEllipses{#1}%
\define@key{markdownOptions}{startNumber}[true]{%
  \def\markdownOptionStartNumber{#1}%
```

If the `tightLists=false` option is specified, when loading the package, then the paralist package for typesetting tight lists will not be automatically loaded. This precaution is meant to minimize the footprint of this package, since some document-classes (beamer) experience clashes with the paralist package.

```
\define@key{markdownOptions}{tightLists}[true]{%
  \def\markdownOptionTightLists{#1}%
```

The following example LaTeX code showcases a possible configuration of plain TeX interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

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

2.3.2.2 Plain TeX Markdown Token Renderers

The LaTeX interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain TeX interface (see Section 2.2.3).

```
\define@key{markdownRenderers}{interblockSeparator}{%
  \renewcommand\markdownRendererInterblockSeparator{#1}%
```

```
\define@key{markdownRenderers}{lineBreak}{%
  \renewcommand\markdownRendererLineBreak{#1}}%
\define@key{markdownRenderers}{ellipsis}{%
  \renewcommand\markdownRendererEllipsis{#1}}%
\define@key{markdownRenderers}{nbsp}{%
  \renewcommand\markdownRendererNbsp{#1}}%
\define@key{markdownRenderers}{leftBrace}{%
  \renewcommand\markdownRendererLeftBrace{#1}}%
\define@key{markdownRenderers}{rightBrace}{%
  \renewcommand\markdownRendererRightBrace{#1}}%
\define@key{markdownRenderers}{dollarSign}{%
  \renewcommand\markdownRendererDollarSign{#1}}%
\define@key{markdownRenderers}{percentSign}{%
  \renewcommand\markdownRendererPercentSign{#1}}%
\define@key{markdownRenderers}{ampersand}{%
  \renewcommand\markdownRendererAmpersand{#1}}%
\define@key{markdownRenderers}{underscore}{%
  \renewcommand\markdownRendererUnderscore{#1}}%
\define@key{markdownRenderers}{hash}{%
  \renewcommand\markdownRendererHash{#1}}%
\define@key{markdownRenderers}{circumflex}{%
  \renewcommand\markdownRendererCircumflex{#1}}%
\define@key{markdownRenderers}{backslash}{%
  \renewcommand\markdownRendererBackslash{#1}}%
\define@key{markdownRenderers}{tilde}{%
  \renewcommand\markdownRendererTilde{#1}}%
\define@key{markdownRenderers}{pipe}{%
  \renewcommand\markdownRendererPipe{#1}}%
\define@key{markdownRenderers}{codeSpan}{%
  \renewcommand\markdownRendererCodeSpan[1]{#1}}%
\define@key{markdownRenderers}{link}{%
  \renewcommand\markdownRendererLink[4]{#1}}%
\define@key{markdownRenderers}{image}{%
  \renewcommand\markdownRendererImage[4]{#1}}%
\define@key{markdownRenderers}{ulBegin}{%
  \renewcommand\markdownRendererUlBegin{#1}}%
\define@key{markdownRenderers}{ulBeginTight}{%
  \renewcommand\markdownRendererUlBeginTight{#1}}%
\define@key{markdownRenderers}{ulItem}{%
  \renewcommand\markdownRendererUlItem{#1}}%
\define@key{markdownRenderers}{ulItemEnd}{%
  \renewcommand\markdownRendererUlItemEnd{#1}}%
\define@key{markdownRenderers}{ulEnd}{%
  \renewcommand\markdownRendererUlEnd{#1}}%
\define@key{markdownRenderers}{ulEndTight}{%
  \renewcommand\markdownRendererUlEndTight{#1}}%
\define@key{markdownRenderers}{olBegin}{%
```

```
  \renewcommand\markdownRendererOlBegin{#1}}%
\define@key{markdownRenderers}{olBeginTight}{%
  \renewcommand\markdownRendererOlBeginTight{#1}}%
\define@key{markdownRenderers}{olItem}{%
  \renewcommand\markdownRendererOlItem{#1}}%
\define@key{markdownRenderers}{olItemWithNumber}{%
  \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
\define@key{markdownRenderers}{olItemEnd}{%
  \renewcommand\markdownRendererOlItemEnd{#1}}%
\define@key{markdownRenderers}{olEnd}{%
  \renewcommand\markdownRendererOlEnd{#1}}%
\define@key{markdownRenderers}{olEndTight}{%
  \renewcommand\markdownRendererOlEndTight{#1}}%
\define@key{markdownRenderers}{dlBegin}{%
  \renewcommand\markdownRendererDlBegin{#1}}%
\define@key{markdownRenderers}{dlBeginTight}{%
  \renewcommand\markdownRendererDlBeginTight{#1}}%
\define@key{markdownRenderers}{dlItem}{%
  \renewcommand\markdownRendererDlItem[1]{#1}}%
\define@key{markdownRenderers}{dlItemEnd}{%
  \renewcommand\markdownRendererDlItemEnd{#1}}%
\define@key{markdownRenderers}{dlDefinitionBegin}{%
  \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
\define@key{markdownRenderers}{dlDefinitionEnd}{%
  \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
\define@key{markdownRenderers}{dlEnd}{%
  \renewcommand\markdownRendererDlEnd{#1}}%
\define@key{markdownRenderers}{dlEndTight}{%
  \renewcommand\markdownRendererDlEndTight{#1}}%
\define@key{markdownRenderers}{emphasis}{%
  \renewcommand\markdownRendererEmphasis[1]{#1}}%
\define@key{markdownRenderers}{strongEmphasis}{%
  \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
\define@key{markdownRenderers}{blockQuoteBegin}{%
  \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
\define@key{markdownRenderers}{blockQuoteEnd}{%
  \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
\define@key{markdownRenderers}{inputVerbatim}{%
  \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
\define@key{markdownRenderers}{inputFencedCode}{%
  \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
\define@key{markdownRenderers}{headingOne}{%
  \renewcommand\markdownRendererHeadingOne[1]{#1}}%
\define@key{markdownRenderers}{headingTwo}{%
  \renewcommand\markdownRendererHeadingTwo[1]{#1}}%
\define@key{markdownRenderers}{headingThree}{%
  \renewcommand\markdownRendererHeadingThree[1]{#1}}%
```

```
\define@key{markdownRenderers}{headingFour}{%
  \renewcommand\markdownRendererHeadingFour[1]{#1}}%
\define@key{markdownRenderers}{headingFive}{%
  \renewcommand\markdownRendererHeadingFive[1]{#1}}%
\define@key{markdownRenderers}{headingSix}{%
  \renewcommand\markdownRendererHeadingSix[1]{#1}}%
\define@key{markdownRenderers}{horizontalRule}{%
  \renewcommand\markdownRendererHorizontalRule{#1}}%
\define@key{markdownRenderers}{footnote}{%
  \renewcommand\markdownRendererFootnote[1]{#1}}%
\define@key{markdownRenderers}{cite}{%
  \renewcommand\markdownRendererCite[1]{#1}}%
\define@key{markdownRenderers}{textCite}{%
  \renewcommand\markdownRendererTextCite[1]{#1}}%
```

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

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

2.3.2.3 Plain TeX Markdown Token Renderer Prototypes

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

```
\define@key{markdownRendererPrototypes}{interblockSeparator}{%
  \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
\define@key{markdownRendererPrototypes}{lineBreak}{%
  \renewcommand\markdownRendererLineBreakPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ellipsis}{%
  \renewcommand\markdownRendererEllipsisPrototype{#1}}%
\define@key{markdownRendererPrototypes}{nbsp}{%
  \renewcommand\markdownRendererNbspPrototype{#1}}%
\define@key{markdownRendererPrototypes}{leftBrace}{%
  \renewcommand\markdownRendererLeftBracePrototype{#1}}%
\define@key{markdownRendererPrototypes}{rightBrace}{%
  \renewcommand\markdownRendererRightBracePrototype{#1}}%
\define@key{markdownRendererPrototypes}{dollarSign}{%
  \renewcommand\markdownRendererDollarSignPrototype{#1}}%
\define@key{markdownRendererPrototypes}{percentSign}{%
```

```
  \renewcommand\markdownRendererPercentSignPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ampersand}{%
  \renewcommand\markdownRendererAmpersandPrototype{#1}}%
\define@key{markdownRendererPrototypes}{underscore}{%
  \renewcommand\markdownRendererUnderscorePrototype{#1}}%
\define@key{markdownRendererPrototypes}{hash}{%
  \renewcommand\markdownRendererHashPrototype{#1}}%
\define@key{markdownRendererPrototypes}{circumflex}{%
  \renewcommand\markdownRendererCircumflexPrototype{#1}}%
\define@key{markdownRendererPrototypes}{backslash}{%
  \renewcommand\markdownRendererBackslashPrototype{#1}}%
\define@key{markdownRendererPrototypes}{tilde}{%
  \renewcommand\markdownRendererTildePrototype{#1}}%
\define@key{markdownRendererPrototypes}{pipe}{%
  \renewcommand\markdownRendererPipePrototype{#1}}%
\define@key{markdownRendererPrototypes}{codeSpan}{%
  \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{link}{%
  \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{image}{%
  \renewcommand\markdownRendererImagePrototype[4]{#1}}%
\define@key{markdownRendererPrototypes}{ulBegin}{%
  \renewcommand\markdownRendererUlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulBeginTight}{%
  \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulItem}{%
  \renewcommand\markdownRendererUlItemPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulItemEnd}{%
  \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulEnd}{%
  \renewcommand\markdownRendererUlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{ulEndTight}{%
  \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olBegin}{%
  \renewcommand\markdownRendererOlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olBeginTight}{%
  \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olItem}{%
  \renewcommand\markdownRendererOlItemPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olItemWithNumber}{%
  \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{olItemEnd}{%
  \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olEnd}{%
  \renewcommand\markdownRendererOlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{olEndTight}{%
  \renewcommand\markdownRendererOlEndTightPrototype{#1}}%
```

```
\define@key{markdownRendererPrototypes}{dlBegin}{%
  \renewcommand\markdownRendererDlBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlBeginTight}{%
  \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlItem}{%
  \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{dlItemEnd}{%
  \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
  \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
  \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlEnd}{%
  \renewcommand\markdownRendererDlEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{dlEndTight}{%
  \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
\define@key{markdownRendererPrototypes}{emphasis}{%
  \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{strongEmphasis}{%
  \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{blockQuoteBegin}{%
  \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
\define@key{markdownRendererPrototypes}{blockQuoteEnd}{%
  \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
\define@key{markdownRendererPrototypes}{inputVerbatim}{%
  \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{inputFencedCode}{%
  \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
\define@key{markdownRendererPrototypes}{headingOne}{%
  \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingTwo}{%
  \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingThree}{%
  \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingFour}{%
  \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingFive}{%
  \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{headingSix}{%
  \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{horizontalRule}{%
  \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
\define@key{markdownRendererPrototypes}{footnote}{%
  \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{cite}{%
  \renewcommand\markdownRendererCitePrototype[1]{#1}}%
\define@key{markdownRendererPrototypes}{textCite}{%
```

```
\renewcommand\markdownRendererTextCitePrototype[1]{#1}%
```

The following example LaTeX code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```
\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}},    % Render inline code via `\texttt`.
  }
}
```

2.4 ConTeXt Interface

The ConTeXt interface provides a start-stop macro pair for the typesetting of markdown input from within ConTeXt. The rest of the interface is inherited from the plain TeX interface (see Section 2.2).

```
\writestatus{loading}{ConTeXt User Module / markdown}%
\unprotect
```

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

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

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

2.4.1 Typesetting Markdown

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

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

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

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
```

3 Technical Documentation

This part of the manual describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

3.1 Lua Implementation

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

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

```
local upper, gsub, format, length =
  string.upper, string.gsub, string.format, string.len
local concat = table.concat
local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, 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.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the Lua code. 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.
    local file = assert(io.open(name, "w"))
    local result = string
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  return name
end
```

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

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

The `util.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 [3, Chapter 21].

```
function util.expand_tabs_in_line(s, tabstop)
  local tab = tabstop or 4
  local corr = 0
  return (s:gsub("()\t", function(p)
            local sp = tab - (p - 1 + corr) % tab
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
end
```

The `util.walk` method walks a rope `t`, applying a function `f` to each leaf element in order. A rope is an array whose elements may be ropes, strings, numbers, or functions. If a leaf element is a function, call it and get the return value before proceeding.

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

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

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

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

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

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

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

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

```
    return nil
  else
    local l = rope[#rope]
    if type(l) == "table" then
      return util.rope_last(l)
    else
      return l
    end
  end
end
```

Given an array `ary` and a string `x`, the `util.intersperse` method returns an array `new`, such that `ary[i] == new[2*(i-1)+1]` and `new[2*i] == x` for all $1 \leq$ `i` $\leq$ `#ary`.

```
function util.intersperse(ary, x)
  local new = {}
  local l = #ary
  for i,v in ipairs(ary) do
    local n = #new
    new[n + 1] = v
    if i ~= l then
      new[n + 2] = x
    end
  end
  return new
end
```

Given an array `ary` and a function `f`, the `util.map` method returns an array `new`, such that `new[i] == f(ary[i])` for all $1 \leq$ `i` $\leq$ `#ary`.

```
function util.map(ary, f)
  local new = {}
  for i,v in ipairs(ary) do
    new[i] = f(v)
  end
  return new
end
```

Given a table `char_escapes` mapping escapable characters to escaped strings and optionally a table `string_escapes` mapping escapable strings to escaped strings, the `util.escaper` method returns an escaper function that escapes all occurances of escapable strings and characters (in this order).

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

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

Build a string of escapable characters.

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

```
622   end
```

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

```
623   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 $(\texttt{k}, \texttt{v}) \in$ `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.

```
624   if string_escapes then
625     for k,v in pairs(string_escapes) do
626       escapable = P(k) / v + escapable
627     end
628   end
```

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

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

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

```
630   return function(s)
631     return lpeg.match(escape_string, s)
632   end
633 end
```

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

```
634 function util.pathname(dir, file)
635   if #dir == 0 then
636     return file
637   else
638     return dir .. "/" .. file
639   end
640 end
```

3.1.2 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.

```
641 M.writer = {}
```

The `writer.new` method creates and returns a new TEX writer object associated with the Lua interface options (see Section 2.1.2) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `writer->`⟨*member*⟩.

```
642 function M.writer.new(options)
643   local self = {}
644   options = options or {}
```

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

```
645   setmetatable(options, { __index = function (_, key)
646     return defaultOptions[key] end })
```

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

```
647   self.suffix = ".tex"
```

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

```
648   self.space = " "
```

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

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

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

```
650   function self.plain(s)
651     return s
652   end
```

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

```
653   function self.paragraph(s)
654     return s
655   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.

```
656   function self.pack(name)
657     return [[\input"]] .. name .. [["\relax]]
658   end
```

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

```
659   self.interblocksep = "\\markdownRendererInterblockSeparator\n{}"
```

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

```
660   self.eof = [[\relax]]
```

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

```
661   self.linebreak = "\\markdownRendererLineBreak\n{}"
```

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

```
662   self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
663   self.hrule = "\\markdownRendererHorizontalRule{}"
```

Define a table `escaped_chars` containing the mapping from special plain TEX characters (including the active pipe character (`|`) of ConTEXt) to their escaped variants. Define tables `escaped_minimal_chars` and `escaped_minimal_strings` containing the mapping from special plain characters and character strings that need to be escaped even in content that will not be typeset.

```
664   local escaped_chars = {
665     ["{"] = "\\markdownRendererLeftBrace{}",
666     ["}"] = "\\markdownRendererRightBrace{}",
667     ["$"] = "\\markdownRendererDollarSign{}",
668     ["%"] = "\\markdownRendererPercentSign{}",
669     ["&"] = "\\markdownRendererAmpersand{}",
670     ["_"] = "\\markdownRendererUnderscore{}",
671     ["#"] = "\\markdownRendererHash{}",
672     ["^"] = "\\markdownRendererCircumflex{}",
673     ["\\"] = "\\markdownRendererBackslash{}",
674     ["~"] = "\\markdownRendererTilde{}",
675     ["|"] = "\\markdownRendererPipe{}", }
676   local escaped_minimal_chars = {
677     ["{"] = "\\markdownRendererLeftBrace{}",
678     ["}"] = "\\markdownRendererRightBrace{}",
679     ["%"] = "\\markdownRendererPercentSign{}",
680     ["\\"] = "\\markdownRendererBackslash{}", }
681   local escaped_minimal_strings = {
682     ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ", }
```

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

```
683   local escape = util.escaper(escaped_chars)
684   local escape_minimal = util.escaper(escaped_minimal_chars,
685     escaped_minimal_strings)
```

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

```
686   if options.hybrid then
687     self.string = function(s) return s end
688     self.uri = function(u) return u end
689   else
690     self.string = escape
691     self.uri = escape_minimal
692   end
```

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

```
693   function self.code(s)
694     return {"\\markdownRendererCodeSpan{",escape(s),"}"}
695   end
```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, and `tit` to the title of the link.

```
696   function self.link(lab,src,tit)
697     return {"\\markdownRendererLink{",lab,"}",
698                             "{",self.string(src),"}",
699                             "{",self.uri(src),"}",
700                             "{",self.string(tit or ""),"}"}
701   end
```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, and `tit` to the title of the image.

```
702   function self.image(lab,src,tit)
703     return {"\\markdownRendererImage{",lab,"}",
704                              "{",self.string(src),"}",
705                              "{",self.uri(src),"}",
706                              "{",self.string(tit or ""),"}"}
707   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.

```
708   local function ulitem(s)
709     return {"\\markdownRendererUlItem ",s,
710             "\\markdownRendererUlItemEnd "}
711   end
712
713   function self.bulletlist(items,tight)
714     local buffer = {}
715     for _,item in ipairs(items) do
716       buffer[#buffer + 1] = ulitem(item)
717     end
718     local contents = util.intersperse(buffer,"\n")
```

```
719     if tight and options.tightLists then
720       return {"\\markdownRendererUlBeginTight\n",contents,
721         "\n\\markdownRendererUlEndTight "}
722     else
723       return {"\\markdownRendererUlBegin\n",contents,
724         "\n\\markdownRendererUlEnd "}
725     end
726   end
```

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

```
727   local function olitem(s,num)
728     if num ~= nil then
729       return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
730               "\\markdownRendererOlItemEnd "}
731     else
732       return {"\\markdownRendererOlItem ",s,
733               "\\markdownRendererOlItemEnd "}
734     end
735   end
736
737   function self.orderedlist(items,tight,startnum)
738     local buffer = {}
739     local num = startnum
740     for _,item in ipairs(items) do
741       buffer[#buffer + 1] = olitem(item,num)
742       if num ~= nil then
743         num = num + 1
744       end
745     end
746     local contents = util.intersperse(buffer,"\n")
747     if tight and options.tightLists then
748       return {"\\markdownRendererOlBeginTight\n",contents,
749         "\n\\markdownRendererOlEndTight "}
750     else
751       return {"\\markdownRendererOlBegin\n",contents,
752         "\n\\markdownRendererOlEnd "}
753     end
754   end
```

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

```
755   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)
    local buffer = {}
    for _,item in ipairs(items) do
      buffer[#buffer + 1] = dlitem(item.term, item.definitions)
    end
    if tight and options.tightLists then
      return {"\\markdownRendererDlBeginTight\n", buffer,
        "\n\\markdownRendererDlEndTight"}
    else
      return {"\\markdownRendererDlBegin\n", buffer,
        "\n\\markdownRendererDlEnd"}
    end
  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)
    return {"\\markdownRendererEmphasis{",s,"}"}
  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)
    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)
    return {"\\markdownRendererBlockQuoteBegin\n",s,
      "\n\\markdownRendererBlockQuoteEnd "}
  end
```

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

```
  function self.verbatim(s)
    local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
    return {"\\markdownRendererInputVerbatim{",name,"}"}
  end
```

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

```
function self.fencedCode(i, s)
  local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
  return {"\\markdownRendererInputFencedCode{",name,"}{",i,"}"}
end
```

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

```
function self.heading(s,level)
  local cmd
  if level == 1 then
    cmd = "\\markdownRendererHeadingOne"
  elseif level == 2 then
    cmd = "\\markdownRendererHeadingTwo"
  elseif level == 3 then
    cmd = "\\markdownRendererHeadingThree"
  elseif level == 4 then
    cmd = "\\markdownRendererHeadingFour"
  elseif level == 5 then
    cmd = "\\markdownRendererHeadingFive"
  elseif level == 6 then
    cmd = "\\markdownRendererHeadingSix"
  else
    cmd = ""
  end
  return {cmd,"{",s,"}"}
end
```

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

```
function self.note(s)
  return {"\\markdownRendererFootnote{",s,"}"}
end
```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is `true`, 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.

```
818   function self.citations(text_cites, cites)
819     local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
820       "{", #cites, "}"}
821     for _,cite in ipairs(cites) do
822       buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
823         cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"}
824     end
825     return buffer
826   end
827
828   return self
829 end
```

3.1.3 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

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

The `reader.new` method creates and returns a new TEX reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object `writer`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these ⟨*member*⟩s as `reader->`⟨*member*⟩.

```
830 M.reader = {}
831 function M.reader.new(writer, options)
832   local self = {}
833   options = options or {}
```

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

```
834   setmetatable(options, { __index = function (_, key)
835     return defaultOptions[key] end })
```

3.1.3.1 Top Level Helper Functions

Define `normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
836   local function normalize_tag(tag)
```

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

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

```
  local expandtabs
  if options.preserveTabs then
    expandtabs = function(s) return s end
  else
    expandtabs = function(s)
                   if s:find("\t") then
                     return s:gsub("[^\n]*", util.expand_tabs_in_line)
                   else
                     return s
                   end
                 end
  end
```

3.1.3.2 Top Level Parsing Functions

```
  local syntax
  local blocks_toplevel
  local blocks
  local inlines, inlines_no_link, inlines_nbsp

  local function create_parser(name, grammar)
    return function(str)
      local res = lpeg.match(grammar(), str)
      if res == nil then
        error(format("%s failed on:\n%s", name, str:sub(1,20)))
      else
        return res
      end
    end
  end

  local parse_blocks = create_parser("parse_blocks",
    function() return blocks end)
  local parse_blocks_toplevel = create_parser("parse_blocks_toplevel",
    function() return blocks_toplevel end)
  local parse_inlines = create_parser("parse_inlines",
    function() return inlines end)
  local parse_inlines_no_link = create_parser("parse_inlines_no_link",
    function() return inlines_no_link end)
  local parse_inlines_nbsp = create_parser("parse_inlines_nbsp",
    function() return inlines_nbsp end)
```

3.1.3.3 Generic PEG Patterns

```
878   local percent               = P("%")
879   local at                    = P("@")
880   local comma                 = P(",")
881   local asterisk              = P("*")
882   local dash                  = P("-")
883   local plus                  = P("+")
884   local underscore            = P("_")
885   local period                = P(".")
886   local hash                  = P("#")
887   local ampersand             = P("&")
888   local backtick              = P("`")
889   local less                  = P("<")
890   local more                  = P(">")
891   local space                 = P(" ")
892   local squote                = P("'")
893   local dquote                = P('"')
894   local lparent               = P("(")
895   local rparent               = P(")")
896   local lbracket              = P("[")
897   local rbracket              = P("]")
898   local circumflex            = P("^")
899   local slash                 = P("/")
900   local equal                 = P("=")
901   local colon                 = P(":")
902   local semicolon             = P(";")
903   local exclamation           = P("!")
904   local tilde                 = P("~")
905
906   local digit                 = R("09")
907   local hexdigit              = R("09","af","AF")
908   local letter                = R("AZ","az")
909   local alphanumeric          = R("AZ","az","09")
910   local keyword               = letter * alphanumeric^0
911   local internal_punctuation  = S(":;,.#$%&-+?<>~/")
912
913   local doubleasterisks       = P("**")
914   local doubleunderscores     = P("__")
915   local fourspaces            = P("    ")
916
917   local any                   = P(1)
918   local fail                  = any - 1
919   local always                = P("")
920
921   local escapable             = S("\\`*_{}[]()+_.!<>#-~:^@;")
922
923   local anyescaped            = P("\\") / "" * escapable
```

```
                                + any

  local tab                     = P("\t")
  local spacechar               = S("\t ")
  local spacing                 = S(" \n\r\t")
  local newline                 = P("\n")
  local nonspacechar            = any - spacing
  local tightblocksep           = P("\001")

  local specialchar             = S("*_‘&[]<!\\.@-")

  local normalchar              = any -
                                  (specialchar + spacing + tightblocksep)
  local optionalspace           = spacechar^0
  local eof                     = - any
  local nonindentspace          = space^-3 * - spacechar
  local indent                  = space^-3 * tab
                                + fourspaces / ""
  local linechar                = P(1 - newline)

  local blankline               = optionalspace * newline / "\n"
  local blanklines              = blankline^0
  local skipblanklines          = (optionalspace * newline)^0
  local indentedline            = indent    /"" * C(linechar^1 * newline^-1)
  local optionallyindentedline = indent^-1 /"" * C(linechar^1 * newline^-1)
  local sp                      = spacing^0
  local spnl                    = optionalspace * (newline * optionalspace)^-1
  local line                    = linechar^0 * newline
                                + linechar^1 * eof
  local nonemptyline            = line - blankline

  local chunk = line * (optionallyindentedline - blankline)^0

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

3.1.3.4 List PEG Patterns

```
  local bulletchar = C(plus + asterisk + dash)

  local bullet     = ( Cg(bulletchar, "bulletchar") * #spacing * (tab + space^-3)
                     + space * Cg(bulletchar, "bulletchar") * #spacing * (tab + space^-2
```

```
                    + space * space * Cg(bulletchar, "bulletchar") * #spacing * (tab +
                    + space * space * space * Cg(bulletchar, "bulletchar") * #spacing
                    )

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

  local enumerator = C(dig^3 * period) * #spacing
                   + C(dig^2 * period) * #spacing * (tab + space^1)
                   + C(dig * period) * #spacing * (tab + space^-2)
                   + space * C(dig^2 * period) * #spacing
                   + space * C(dig * period) * #spacing * (tab + space^-1)
                   + space * space * C(dig^1 * period) * #spacing
```

3.1.3.5 Code Span PEG Patterns

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

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

  local closeticks  = space^-1 *
                      Cmt(C(backtick^1) * Cb("ticks"), captures_equal_length)

  local intickschar = (any - S(" \n\r`"))
                    + (newline * -blankline)
                    + (space - closeticks)
                    + (backtick^1 - closeticks)

  local inticks     = openticks * space^-1 * C(intickschar^0) * closeticks
% \paragraph{Fenced Code \acro{peg} Patterns}
%  \begin{macrocode}
  local function captures_geq_length(s,i,a,b)
    return #a >= #b and i
  end

  local infostring     = (linechar - (backtick + space^1 * (newline + eof)))^0

  local fenceindent
  local function fencehead(char)
    return               C(nonindentspace) / function(s) fenceindent = #s end
                       * Cg(char^3, "fencelength")
                       * optionalspace * C(infostring) * optionalspace
```

```
                      * (newline + eof)
end

local function fencetail(char)
  return                nonindentspace
                      * Cmt(C(char^3) * Cb("fencelength"),
                            captures_geq_length)
                      * optionalspace * (newline + eof)
                      + eof
end

local function fencedline(char)
  return                C(line - fencetail(char))
                      / function(s)
                            return s:gsub("^" .. string.rep(" ?",
                                fenceindent), "")
                        end
end
```

3.1.3.6 Tag PEG Patterns

```
local leader        = space^-3

-- in balanced brackets, parentheses, quotes:
local bracketed     = P{ lbracket
                         * ((anyescaped - (lbracket + rbracket
                            + blankline^2)) + V(1))^0
                         * rbracket }

local inparens      = P{ lparent
                         * ((anyescaped - (lparent + rparent
                            + blankline^2)) + V(1))^0
                         * rparent }

local squoted       = P{ squote * alphanumeric
                         * ((anyescaped - (squote + blankline^2))
                            + V(1))^0
                         * squote }

local dquoted       = P{ dquote * alphanumeric
                         * ((anyescaped - (dquote + blankline^2))
                            + V(1))^0
                         * dquote }

-- bracketed 'tag' for markdown links, allowing nested brackets:
local tag           = lbracket
                    * Cs((alphanumeric^1
```

```
                          + bracketed
                          + inticks
                          + (anyescaped - (rbracket + blankline^2)))^0)
                      * rbracket

-- url for markdown links, allowing balanced parentheses:
local url           = less * Cs((anyescaped-more)^0) * more
                      + Cs((inparens + (anyescaped-spacing-rparent))^1)

-- quoted text possibly with nested quotes:
local title_s       = squote * Cs(((anyescaped-squote) + squoted)^0) *
                      squote

local title_d       = dquote * Cs(((anyescaped-dquote) + dquoted)^0) *
                      dquote

local title_p       = lparent
                      * Cs((inparens + (anyescaped-rparent))^0)
                      * rparent

local title         = title_d + title_s + title_p

local optionaltitle = spnl * title * spacechar^0
                      + Cc("")
```

3.1.3.7 Citation PEG Patterns

```
local citation_name = Cs(dash^-1) * at
                      * Cs(alphanumeric
                          * (alphanumeric + internal_punctuation
                              - comma - semicolon)^0)

local citation_body_prenote
                      = Cs((alphanumeric^1
                          + bracketed
                          + inticks
                          + (anyescaped
                              - (rbracket + blankline^2))
                          - (spnl * dash^-1 * at))^0)

local citation_body_postnote
                      = Cs((alphanumeric^1
                          + bracketed
                          + inticks
                          + (anyescaped
                              - (rbracket + semicolon + blankline^2))
                          - (spnl * rbracket))^0)
```

```
1100
1101  local citation_body_chunk
1102                      = citation_body_prenote
1103                      * spnl * citation_name
1104                      * (comma * spnl)^-1
1105                      * citation_body_postnote
1106
1107  local citation_body = citation_body_chunk
1108                      * (semicolon * spnl * citation_body_chunk)^0
1109
1110  local citation_headless_body_postnote
1111                      = Cs((alphanumeric^1
1112                           + bracketed
1113                           + inticks
1114                           + (anyescaped
1115                                - (rbracket + at + semicolon + blankline^2))
1116                           - (spnl * rbracket))^0)
1117
1118  local citation_headless_body
1119                      = citation_headless_body_postnote
1120                      * (sp * semicolon * spnl * citation_body_chunk)^0
```

3.1.3.8 Footnote PEG Patterns

```
1121  local rawnotes = {}
1122
1123  local function strip_first_char(s)
1124    return s:sub(2)
1125  end
1126
1127  -- like indirect_link
1128  local function lookup_note(ref)
1129    return function()
1130      local found = rawnotes[normalize_tag(ref)]
1131      if found then
1132        return writer.note(parse_blocks_toplevel(found))
1133      else
1134        return {"[", parse_inlines("^" .. ref), "]"}
1135      end
1136    end
1137  end
1138
1139  local function register_note(ref,rawnote)
1140    rawnotes[normalize_tag(ref)] = rawnote
1141    return ""
1142  end
1143
```

```
local RawNoteRef = #(lbracket * circumflex) * tag / strip_first_char

local NoteRef    = RawNoteRef / lookup_note

local NoteBlock

if options.footnotes then
  NoteBlock = leader * RawNoteRef * colon * spnl *
              indented_blocks(chunk) / register_note
else
  NoteBlock = fail
end
```

3.1.3.9 Link and Image PEG Patterns

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

-- add a reference to the list
local function register_link(tag,url,title)
    references[normalize_tag(tag)] = { url = url, title = title }
    return ""
end

-- parse a reference definition:  [foo]: /bar "title"
local define_reference_parser =
  leader * tag * colon * spacechar^0 * url * optionaltitle * blankline^1

-- lookup link reference and return either
-- the link or nil and fallback text.
local function lookup_reference(label,sps,tag)
    local tagpart
    if not tag then
        tag = label
        tagpart = ""
    elseif tag == "" then
        tag = label
        tagpart = "[]"
    else
        tagpart = {"[", parse_inlines(tag), "]"}
    end
    if sps then
      tagpart = {sps, tagpart}
    end
    local r = references[normalize_tag(tag)]
    if r then
      return r
```

```
      else
        return nil, {"[", parse_inlines(label), "]", tagpart}
      end
  end

  -- lookup link reference and return a link, if the reference is found,
  -- or a bracketed label otherwise.
  local function indirect_link(label,sps,tag)
    return function()
      local r,fallback = lookup_reference(label,sps,tag)
      if r then
        return writer.link(parse_inlines_no_link(label), r.url, r.title)
      else
        return fallback
      end
    end
  end

  -- lookup image reference and return an image, if the reference is found,
  -- or a bracketed label otherwise.
  local function indirect_image(label,sps,tag)
    return function()
      local r,fallback = lookup_reference(label,sps,tag)
      if r then
        return writer.image(writer.string(label), r.url, r.title)
      else
        return {"!", fallback}
      end
    end
  end
```

3.1.3.10 Spacing PEG Patterns

```
  local bqstart      = more
  local headerstart  = hash
                     + (line * (equal^1 + dash^1) * optionalspace * newline)
  local fencestart   = fencehead(backtick) + fencehead(tilde)

  if options.blankBeforeBlockquote then
    bqstart = fail
  end

  if options.blankBeforeHeading then
    headerstart = fail
  end

  if not options.fencedCode or options.blankBeforeCodeFence then
```

```
    fencestart = fail
end
```

3.1.3.11 String PEG Rules

```
local Inline    = V("Inline")

local Str       = normalchar^1 / writer.string

local Symbol    = (specialchar - tightblocksep) / writer.string
```

3.1.3.12 Ellipsis PEG Rules

```
local Ellipsis  = P("...") / writer.ellipsis

local Smart     = Ellipsis
```

3.1.3.13 Inline Code Block PEG Rules

```
local Code      = inticks / writer.code
```

3.1.3.14 Spacing PEG Rules

```
local Endline   = newline * -( -- newline, but not before...
                      blankline -- paragraph break
                    + tightblocksep  -- nested list
                    + eof       -- end of document
                    + bqstart
                    + headerstart
                    + fencestart
                  ) * spacechar^0 / writer.space

local Space     = spacechar^2 * Endline / writer.linebreak
                + spacechar^1 * Endline^-1 * eof / ""
                + spacechar^1 * Endline^-1 * optionalspace / writer.space

local NonbreakingEndline
                = newline * -( -- newline, but not before...
                      blankline -- paragraph break
                    + tightblocksep  -- nested list
                    + eof       -- end of document
                    + bqstart
                    + headerstart
                    + fencestart
                  ) * spacechar^0 / writer.nbsp

local NonbreakingSpace
                = spacechar^2 * Endline / writer.linebreak
```

```
                 + spacechar^1 * Endline^-1 * eof / ""
                 + spacechar^1 * Endline^-1 * optionalspace / writer.nbsp

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

3.1.3.15 Emphasis PEG Rules

```
local Strong = ( between(Inline, doubleasterisks, doubleasterisks)
               + between(Inline, doubleunderscores, doubleunderscores)
               ) / writer.strong

local Emph   = ( between(Inline, asterisk, asterisk)
               + between(Inline, underscore, underscore)
               ) / writer.emphasis
```

3.1.3.16 Link PEG Rules

```
local urlchar = anyescaped - newline - more

local AutoLinkUrl   = less
                    * C(alphanumeric^1 * P("://") * urlchar^1)
                    * more
                    / function(url)
                      return writer.link(writer.string(url), url)
                    end

local AutoLinkEmail = less
                    * C((alphanumeric + S("-._+"))^1 * P("@") * urlchar^1)
                    * more
                    / function(email)
                      return writer.link(writer.string(email),
                                         "mailto:"..email)
                    end

local DirectLink    = (tag / parse_inlines_no_link)  -- no links inside links
                    * spnl
                    * lparent
                    * (url + Cc(""))  -- link can be empty [foo]()
                    * optionaltitle
                    * rparent
                    / writer.link

local IndirectLink = tag * (C(spnl) * tag)^-1 / indirect_link
```

```
1310    -- parse a link or image (direct or indirect)
1311    local Link          = DirectLink + IndirectLink
```

3.1.3.17 Image PEG Rules

```
1312    local DirectImage   = exclamation
1313                        * (tag / parse_inlines)
1314                        * spnl
1315                        * lparent
1316                        * (url + Cc("")  -- link can be empty [foo]()
1317                        * optionaltitle
1318                        * rparent
1319                        / writer.image
1320
1321    local IndirectImage  = exclamation * tag * (C(spnl) * tag)^-1 /
1322                           indirect_image
1323
1324    local Image         = DirectImage + IndirectImage
```

3.1.3.18 Miscellaneous Inline PEG Rules

```
1325    -- avoid parsing long strings of * or _ as emph/strong
1326    local UlOrStarLine  = asterisk^4 + underscore^4 / writer.string
1327
1328    local EscapedChar   = S("\\") * C(escapable) / writer.string
```

3.1.3.19 Citations PEG Rules

```
1329    local function citations(text_cites, raw_cites)
1330        local function normalize(str)
1331            if str == "" then
1332                str = nil
1333            else
1334                str = (options.citationNbsps and parse_inlines_nbsp or
1335                  parse_inlines)(str)
1336            end
1337            return str
1338        end
1339
1340        local cites = {}
1341        for i = 1,#raw_cites,4 do
1342            cites[#cites+1] = {
1343                prenote = normalize(raw_cites[i]),
1344                suppress_author = raw_cites[i+1] == "-",
1345                name = writer.string(raw_cites[i+2]),
1346                postnote = normalize(raw_cites[i+3]),
1347            }
1348        end
```

```
1349        return writer.citations(text_cites, cites)
1350    end
1351
1352    local TextCitations = Ct(Cc("")
1353                          * citation_name
1354                          * ((spnl
1355                              * lbracket
1356                              * citation_headless_body
1357                              * rbracket) + Cc(""))) /
1358                            function(raw_cites)
1359                                return citations(true, raw_cites)
1360                            end
1361
1362    local ParenthesizedCitations
1363                          = Ct(lbracket
1364                          * citation_body
1365                          * rbracket) /
1366                            function(raw_cites)
1367                                return citations(false, raw_cites)
1368                            end
1369
1370    local Citations     = TextCitations + ParenthesizedCitations
```

3.1.3.20 Code Block PEG Rules

```
1371    local Block           = V("Block")
1372
1373    local Verbatim        = Cs( (blanklines
1374                              * ((indentedline - blankline))^1)^1
1375                              ) / expandtabs / writer.verbatim
1376
1377    local TildeFencedCode
1378                          = fencehead(tilde)
1379                          * Cs(fencedline(tilde)^0)
1380                          * fencetail(tilde)
1381
1382    local BacktickFencedCode
1383                          = fencehead(backtick)
1384                          * Cs(fencedline(backtick)^0)
1385                          * fencetail(backtick)
1386
1387    local FencedCode      = (TildeFencedCode + BacktickFencedCode)
1388                          / function(infostring, code)
1389                                return writer.fencedCode(
1390                                    writer.string(infostring),
1391                                    expandtabs(code))
1392                            end
```

3.1.3.21 Blockquote PEG Patterns

```
1393  -- strip off leading > and indents, and run through blocks
1394  local Blockquote     = Cs((
1395            ((leader * more * space^-1)/"" * linechar^0 * newline)^1
1396          * (-blankline * linechar^1 * newline)^0
1397          * (blankline^0 / "")
1398          )^1) / parse_blocks_toplevel / writer.blockquote
1399
1400  local function lineof(c)
1401      return (leader * (P(c) * optionalspace)^3 * (newline * blankline^1
1402          + newline^-1 * eof))
1403  end
```

3.1.3.22 Horizontal Rule PEG Rules

```
1404  local HorizontalRule = ( lineof(asterisk)
1405                         + lineof(dash)
1406                         + lineof(underscore)
1407                         ) / writer.hrule
```

3.1.3.23 List PEG Rules

```
1408  local starter = bullet + enumerator
1409
1410  -- we use \001 as a separator between a tight list item and a
1411  -- nested list under it.
1412  local NestedList            = Cs((optionallyindentedline - starter)^1)
1413                              / function(a) return "\001"..a end
1414
1415  local ListBlockLine         = optionallyindentedline
1416                                - blankline - (indent^-1 * starter)
1417
1418  local ListBlock             = line * ListBlockLine^0
1419
1420  local ListContinuationBlock = blanklines * (indent / "") * ListBlock
1421
1422  local function TightListItem(starter)
1423      return -HorizontalRule
1424             * (Cs(starter / "" * ListBlock * NestedList^-1) /
1425                parse_blocks)
1426             * -(blanklines * indent)
1427  end
1428
1429  local function LooseListItem(starter)
1430      return -HorizontalRule
1431             * Cs( starter / "" * ListBlock * Cc("\n")
1432               * (NestedList + ListContinuationBlock^0)
```

```
              * (blanklines / "\n\n")
              ) / parse_blocks
end

local BulletList = ( Ct(TightListItem(bullet)^1)
                     * Cc(true) * skipblanklines * -bullet
                   + Ct(LooseListItem(bullet)^1)
                     * Cc(false) * skipblanklines ) /
                       writer.bulletlist

local function orderedlist(items,tight,startNumber)
  if options.startNumber then
    startNumber = tonumber(startNumber) or 1  -- fallback for '#'
  else
    startNumber = nil
  end
  return writer.orderedlist(items,tight,startNumber)
end

local OrderedList = Cg(enumerator, "listtype") *
                    ( Ct(TightListItem(Cb("listtype")) *
                         TightListItem(enumerator)^0)
                      * Cc(true) * skipblanklines * -enumerator
                    + Ct(LooseListItem(Cb("listtype")) *
                         LooseListItem(enumerator)^0)
                      * Cc(false) * skipblanklines
                    ) * Cb("listtype") / orderedlist

local defstartchar = S("~:")
local defstart     = ( defstartchar * #spacing * (tab + space^-3)
                     + space * defstartchar * #spacing * (tab + space^-2)
                     + space * space * defstartchar * #spacing *
                       (tab + space^-1)
                     + space * space * space * defstartchar * #spacing
                     )

local dlchunk = Cs(line * (indentedline - blankline)^0)

local function definition_list_item(term, defs, tight)
  return { term = parse_inlines(term), definitions = defs }
end

local DefinitionListItemLoose = C(line) * skipblanklines
                              * Ct((defstart *
                                    indented_blocks(dlchunk) /
                                    parse_blocks_toplevel)^1)
                              * Cc(false)
```

```
                          / definition_list_item

local DefinitionListItemTight = C(line)
                          * Ct((defstart * dlchunk /
                                          parse_blocks)^1)
                          * Cc(true)
                          / definition_list_item

local DefinitionList =  ( Ct(DefinitionListItemLoose^1) * Cc(false)
                        +  Ct(DefinitionListItemTight^1)
                           * (skipblanklines *
                              -DefinitionListItemLoose * Cc(true))
                        ) / writer.definitionlist
```

3.1.3.24 Blank Line PEG Rules

```
local Reference      = define_reference_parser / register_link
local Blank          = blankline / ""
                     + NoteBlock
                     + Reference
                     + (tightblocksep / "\n")
```

3.1.3.25 Paragraph PEG Rules

```
local Paragraph      = nonindentspace * Ct(Inline^1) * newline
                     * ( blankline^1
                       + #hash
                       + #(leader * more * space^-1)
                       )
                     / writer.paragraph

local ToplevelParagraph
                     = nonindentspace * Ct(Inline^1) * (newline
                     * ( blankline^1
                       + #hash
                       + #(leader * more * space^-1)
                       + eof
                       )
                     + eof )
                     / writer.paragraph

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

3.1.3.26 Heading PEG Rules

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

```

```
-- parse setext header ending and return level
local HeadingLevel = equal^1 * Cc(1) + dash^1 * Cc(2)

local function strip_atx_end(s)
  return s:gsub("[#%s]*\n$","")
end

-- parse atx header
local AtxHeading = Cg(HeadingStart,"level")
                   * optionalspace
                   * (C(line) / strip_atx_end / parse_inlines)
                   * Cb("level")
                   / writer.heading

-- parse setext header
local SetextHeading = #(line * S("=-"))
                   * Ct(line / parse_inlines)
                   * HeadingLevel
                   * optionalspace * newline
                   / writer.heading

local Heading = AtxHeading + SetextHeading
```

3.1.3.27 Top Level PEG Specification

```
syntax =
  { "Blocks",

    Blocks                = Blank^0 *
                            Block^-1 *
                            (Blank^0 / function()
                              return writer.interblocksep
                             end * Block)^0 *
                            Blank^0 *
                            eof,

    Blank                 = Blank,

    Block                 = V("Blockquote")
                          + V("Verbatim")
                          + V("FencedCode")
                          + V("HorizontalRule")
                          + V("BulletList")
                          + V("OrderedList")
                          + V("Heading")
                          + V("DefinitionList")
                          + V("Paragraph")
```

```
                          + V("Plain"),

    Blockquote            = Blockquote,
    Verbatim              = Verbatim,
    FencedCode            = FencedCode,
    HorizontalRule        = HorizontalRule,
    BulletList            = BulletList,
    OrderedList           = OrderedList,
    Heading               = Heading,
    DefinitionList        = DefinitionList,
    DisplayHtml           = DisplayHtml,
    Paragraph             = Paragraph,
    Plain                 = Plain,

    Inline                = V("Str")
                          + V("Space")
                          + V("Endline")
                          + V("UlOrStarLine")
                          + V("Strong")
                          + V("Emph")
                          + V("NoteRef")
                          + V("Citations")
                          + V("Link")
                          + V("Image")
                          + V("Code")
                          + V("AutoLinkUrl")
                          + V("AutoLinkEmail")
                          + V("EscapedChar")
                          + V("Smart")
                          + V("Symbol"),

    Str                   = Str,
    Space                 = Space,
    Endline               = Endline,
    UlOrStarLine          = UlOrStarLine,
    Strong                = Strong,
    Emph                  = Emph,
    NoteRef               = NoteRef,
    Citations             = Citations,
    Link                  = Link,
    Image                 = Image,
    Code                  = Code,
    AutoLinkUrl           = AutoLinkUrl,
    AutoLinkEmail         = AutoLinkEmail,
    InlineHtml            = InlineHtml,
    HtmlEntity            = HtmlEntity,
    EscapedChar           = EscapedChar,
```

```
      Smart                 = Smart,
      Symbol                = Symbol,
    }

  if not options.definitionLists then
    syntax.DefinitionList = fail
  end

  if not options.fencedCode then
    syntax.FencedCode = fail
  end

  if not options.citations then
    syntax.Citations = fail
  end

  if not options.footnotes then
    syntax.NoteRef = fail
  end

  if not options.smartEllipses then
    syntax.Smart = fail
  end

  local blocks_toplevel_t = util.table_copy(syntax)
  blocks_toplevel_t.Paragraph = ToplevelParagraph
  blocks_toplevel = Ct(blocks_toplevel_t)

  blocks = Ct(syntax)

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

  local inlines_no_link_t = util.table_copy(inlines_t)
  inlines_no_link_t.Link = fail
  inlines_no_link = Ct(inlines_no_link_t)

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

3.1.3.28 Exported Conversion Function

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

```
  function self.convert(input)
    references = {}
```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.2). The `cacheDir` option is disregarded.

```
    local opt_string = {}
    for k,_ in pairs(defaultOptions) do
      local v = options[k]
      if k ~= "cacheDir" then
        opt_string[#opt_string+1] = k .. "=" .. tostring(v)
      end
    end
    table.sort(opt_string)
    local salt = table.concat(opt_string, ",") .. "," .. metadata.version
```

Produce the cache file, transform its filename via the `writer->pack` method, and return the result.

```
    local name = util.cache(options.cacheDir, input, salt, function(input)
        return util.rope_to_string(parse_blocks_toplevel(input)) .. writer.eof
      end, ".md" .. writer.suffix)
    return writer.pack(name)
  end
  return self
end
```

3.1.4 Conversion from Markdown to Plain TEX

The `new` method returns the `reader->convert` function of a reader object associated with the Lua interface options (see Section 2.1.2) `options` and with a writer object associated with `options`.

```
function M.new(options)
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  return reader.convert
end

return M
```

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

```
\def\markdownInfo#1{%
  \message{(l.\the\inputlineno) markdown.tex info: #1.}}%
\def\markdownWarning#1{%
  \message{(l.\the\inputlineno) markdown.tex warning: #1}}%
\def\markdownError#1#2{%
  \errhelp{#2.}%
  \errmessage{(l.\the\inputlineno) markdown.tex error: #1}}%
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererLineBreakPrototype{\hfil\break}%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`{}%
\def\markdownRendererRightBracePrototype{\char`}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\char`&}%
\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\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
```

```
\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{{\it#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input"#1"\relax}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \markdownRendererInputVerbatimPrototype{#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\markdownRendererHorizontalRulePrototype{}%
\def\markdownRendererFootnotePrototype#1{#1}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
```

3.2.3 Lua Snippets

The `\markdownLuaOptions` macro expands to a Lua table that contains the plain TEX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.2). Note that the boolean options are not sanitized and expect the plain TEX option macros to expand to either `true` or `false`.

```
\def\markdownLuaOptions{{%
\ifx\markdownOptionBlankBeforeBlockquote\undefined\else
  blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
\fi
\ifx\markdownOptionBlankBeforeCodeFence\undefined\else
  blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
\fi
\ifx\markdownOptionBlankBeforeHeading\undefined\else
  blankBeforeHeading = \markdownOptionBlankBeforeHeading,
\fi
\ifx\markdownOptionCacheDir\undefined\else
  cacheDir = "\markdownOptionCacheDir",
\fi
```

```
\ifx\markdownOptionCitations\undefined\else
  citations = \markdownOptionCitations,
\fi
\ifx\markdownOptionCitationNbsps\undefined\else
  citationNbsps = \markdownOptionCitationNbsps,
\fi
\ifx\markdownOptionDefinitionLists\undefined\else
  definitionLists = \markdownOptionDefinitionLists,
\fi
\ifx\markdownOptionFootnotes\undefined\else
  footnotes = \markdownOptionFootnotes,
\fi
\ifx\markdownOptionFencedCode\undefined\else
  fencedCode = \markdownOptionFencedCode,
\fi
\ifx\markdownOptionHashEnumerators\undefined\else
  hashEnumerators = \markdownOptionHashEnumerators,
\fi
\ifx\markdownOptionHybrid\undefined\else
  hybrid = \markdownOptionHybrid,
\fi
\ifx\markdownOptionPreserveTabs\undefined\else
  preserveTabs = \markdownOptionPreserveTabs,
\fi
\ifx\markdownOptionSmartEllipses\undefined\else
  smartEllipses = \markdownOptionSmartEllipses,
\fi
\ifx\markdownOptionStartNumber\undefined\else
  startNumber = \markdownOptionStartNumber,
\fi
\ifx\markdownOptionTightLists\undefined\else
  tightLists = \markdownOptionTightLists,
\fi}
}%
```

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 `\markdownOptionCacheDir` directory exists.

```
local lfs = require("lfs")
local cacheDir = "\markdownOptionCacheDir"
if lfs.isdir(cacheDir) == true then else
  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)
}%
```

3.2.4 Lua Shell Escape Bridge

The following TEX code is intended for TEX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` and `\markdownReadAndConvert` macros defined here and in Section 3.2.5 are meant to be transparent to the remaining code.

The package assumes that although the user is not using the LuaTEX engine, their TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the TEXLua interpreter (see [1, Section 3.1.1]).

```

\ifnum\markdownMode<2\relax
\ifnum\markdownMode=0\relax
  \markdownInfo{Using mode 0: Shell escape via write18}%
\else
  \markdownInfo{Using mode 1: Shell escape via os.execute}%
\fi
```

The macro `\markdownLuaExecuteFileStream` contains the number of the output file stream that will be used to store the helper Lua script in the file named `\markdownOptionHelperScriptFileName` during the expansion of the macro `\markdownLuaExecute`, and to store the markdown input in the file named `\markdownOptionInputTempFileName` during the expansion of the macro `\markdownReadAndConvert`.

```
\csname newwrite\endcsname\markdownLuaExecuteFileStream
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (`1`), disabled (`0`), or restricted (`2`).

Inherit the value of the the `\pdfshellescape` (LuaTEX, PdfTEX) or the `\shellescape` (XETEX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```
\ifx\pdfshellescape\undefined
  \ifx\shellescape\undefined
    \ifnum\markdownMode=0\relax
      \def\markdownExecuteShellEscape{1}%
    \else
      \def\markdownExecuteShellEscape{%
        \directlua{tex.sprint(status.shell_escape or "1")}}%
```

```
    \fi
  \else
    \let\markdownExecuteShellEscape\shellescape
  \fi
\else
  \let\markdownExecuteShellEscape\pdfshellescape
\fi
```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `markdown.execute` method otherwise.

```
\ifnum\markdownMode=0\relax
  \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
\else
  \def\markdownExecuteDirect#1{%
    \directlua{os.execute("\luaescapestring{#1}")}}%
\fi
```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

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

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TEX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```
\def\markdownLuaExecute#1{%
```

Create the file `\markdownOptionHelperScriptFileName` and fill it with the input Lua code prepended with kpathsea initialization, so that Lua modules from the TEX distribution are available.

```
  \immediate\openout\markdownLuaExecuteFileStream=%
    \markdownOptionHelperScriptFileName
  \markdownInfo{Writing a helper Lua script to the file
    "\markdownOptionHelperScriptFileName"}%
  \immediate\write\markdownLuaExecuteFileStream{%
    local kpse = require('kpse')
    kpse.set_program_name('luatex') #1}%
  \immediate\closeout\markdownLuaExecuteFileStream
```

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

```
\markdownInfo{Executing a helper Lua script from the file
  "\markdownOptionHelperScriptFileName" and storing the result in the
  file "\markdownOptionOutputTempFileName"}%
\markdownExecute{texlua "\markdownOptionHelperScriptFileName" >
  "\markdownOptionOutputTempFileName"}%
```

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

```
  \input\markdownOptionOutputTempFileName\relax}%
```

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.

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

Open the `\markdownOptionInputTempFileName` file for writing.

```
    |immediate|openout|markdownLuaExecuteFileStream%
      |markdownOptionInputTempFileName%
    |markdownInfo{Buffering 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 `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Note the use of the comments to ensure that the entire macro is at a single line and therefore no (active) newline symbols are produced.

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

When the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` file.

```
      |ifx|relax##3|relax%
        |immediate|write|markdownLuaExecuteFileStream{##1}%
      |else%
```

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

```
        |def^^M{%
          |markdownInfo{The ending token sequence was found}%
          |immediate|closeout|markdownLuaExecuteFileStream%
          |endgroup%
          |markdownInput|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{%
        |markdownReadAndConvertProcessLine####1#1#1|relax}%
      ^^M}%
    ^^M}%
```

Reset the character categories back to the former state.

```
|endgroup
```

3.2.5 Direct Lua Access

The following TEX code is intended for TEX engines that provide direct access to Lua (LuaTEX). The `\markdownLuaExecute` and `\markdownReadAndConvert` defined here and in Section 3.2.4 are meant to be transparent to the remaining code. This corresponds to the `\markdownMode` value of `2`.

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

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `\tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.4,

```
\def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%
```

In the definition of the direct Lua access version of the `\markdownReadAndConvert` macro, we will be using the hash symbol (`#`), the underscore symbol (`_`), the circumflex symbol (`^`), the dollar sign (`$`), the backslash symbol (`\`), the percent sign (`%`), and the braces (`{}`) as a part of the Lua syntax.

```
\begingroup
```

To this end, we will make the underscore symbol, the dollar sign, and circumflex symbols ordinary characters,

```
    \catcode`\_=12%
    \catcode`\$=12%
    \catcode`\^=12%
```

swap the category code of the hash symbol with the slash symbol (`/`).

```
    \catcode`\/=6%
    \catcode`\#=12%
```

swap the category code of the percent sign with the at symbol (`@`).

```
    \catcode`\@=14%
    \catcode`\%=12%
```

swap the category code of the backslash symbol with the pipe symbol (`|`),

```
    \catcode`|=0@
    \catcode`\\=12@
```

Braces are a part of the plain TEX syntax, but they are not removed during expansion, so we do not need to bother with changing their category codes.

```
    |gdef|markdownReadAndConvert/1/2{@
```

Make the `\markdownReadAndConvertAfter` macro store the token sequence that will be inserted into the document after the ending token sequence has been found.

```
      |def|markdownReadAndConvertAfter{/2}@
      |markdownInfo{Buffering markdown input and scanning for the
        closing token sequence "/1"}@
      |directlua{@
```

Set up an empty Lua table that will serve as our buffer.

```
        |markdownPrepare
        local buffer = {}
```

Create a regex that will match the ending input sequence. Escape any special regex characters (like a star inside `\end{markdown*}`) inside the input.

```
1909          local ending_sequence = "^.-" .. ([[/1]]):gsub(
1910            "([%(%)%.%%%+%-%*%?%[%]%^%$])", "%%%1")
```

Register a callback that will notify you about new lines of input.

```
1911          |markdownLuaRegisterIBCallback{function(line)
```

When the ending token sequence appears on a line, unregister the callback, convert the contents of our buffer from markdown to plain TEX, and insert the result into the input line buffer of TEX.

```
1912            if line:match(ending_sequence) then
1913              |markdownLuaUnregisterIBCallback;
1914              local input = table.concat(buffer, "\n") .. "\n"
1915              local output = convert(input)
1916              return [[\markdownInfo{The ending token sequence was found}]] ..
1917                output .. [[\markdownReadAndConvertAfter]]
```

When the ending token sequence does not appear on a line, store the line in our buffer, and insert either `\fi`, if this is the first line of input, or an empty token list to the input line buffer of TEX.

```
1918            else
1919              buffer[#buffer+1] = line
1920              return [[\]] .. (#buffer == 1 and "fi" or "relax")
1921            end
1922          end}}@
```

Insert `\iffalse` after the `\markdownReadAndConvert` macro in order to consume the rest of the first line of input.

```
1923        |iffalse}@
```

Reset the character categories back to the former state.

```
1924    |endgroup
1925  \fi
```

3.2.6 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.

```
1926  \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.

```
1927    \catcode`|=0%
1928    \catcode`\\=12%
1929    |gdef|markdownInput#1{%
```

```
    |markdownInfo{Including markdown document "#1"}%
    |markdownLuaExecute{%
      |markdownPrepare
      local input = assert(io.open("#1","r")):read("*a")
```

Since the Lua converter expects UNIX line endings, normalize the input.

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

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 (see [4, Section 9]). As a consequence, we can directly reuse the existing plain TeX implementation.

```
\input markdown
\def\markdownVersionSpace{ }%
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Logging Facilities

The LaTeX implementation redefines the plain TeX logging macros (see Section 3.2.1) to use the LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

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

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\markdownInput` macro. The `\markdownInput` 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.

```
\renewenvironment{markdown}{%
  \markdownReadAndConvert@markdown{}}\relax
\renewenvironment{markdown*}[1]{%
```

```
  \markdownSetup{#1}%
  \markdownReadAndConvert@markdown*}\relax
\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
```

3.3.3 Options

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

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

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

```
\define@key{markdownOptions}{renderers}{%
  \setkeys{markdownRenderers}{#1}%
  \def\KV@prefix{KV@markdownOptions@}}%
\define@key{markdownOptions}{rendererPrototypes}{%
  \setkeys{markdownRendererPrototypes}{#1}%
  \def\KV@prefix{KV@markdownOptions@}}%
```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\RequirePackage{url}
\RequirePackage{graphicx}
```

If the `\markdownOptionTightLists` macro expands to `false`, do not load the paralist package. This is necessary for LaTeX 2$_\varepsilon$ document classes that do not play nice with paralist, such as beamer.

```
\RequirePackage{ifthen}
\ifx\markdownOptionTightLists\undefined
  \RequirePackage{paralist}
\else
  \ifthenelse{\equal{\markdownOptionTightLists}{false}}{}{
    \RequirePackage{paralist}}
\fi
```

```
\RequirePackage{fancyvrb}
\markdownSetup{rendererPrototypes={
  lineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  pipe = {\textbar},
  codeSpan = {\texttt{#1}},
  link = {#1\footnote{\ifx\empty#4\empty\else#4:
    \fi\texttt<\url{#3}\texttt>}},
  image = {\begin{figure}
      \begin{center}%
        \includegraphics{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
      \label{fig:#1}%
    \end{figure}},
  ulBegin = {\begin{itemize}},
  ulBeginTight = {\begin{compactitem}},
  ulItem = {\item},
  ulEnd = {\end{itemize}},
  ulEndTight = {\end{compactitem}},
  olBegin = {\begin{enumerate}},
  olBeginTight = {\begin{compactenum}},
  olItem = {\item},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  olEndTight = {\end{compactenum}},
  dlBegin = {\begin{description}},
  dlBeginTight = {\begin{compactdesc}},
  dlItem = {\item[#1]},
  dlEnd = {\end{description}},
  dlEndTight = {\end{compactdesc}},
  emphasis = {\emph{#1}},
  strongEmphasis = {%
    \ifx\alert\undefined
      \textbf{\emph{#1}}%
    \else % Beamer support
      \alert{\emph{#1}}%
    \fi},
  blockQuoteBegin = {\begin{quotation}},
```

```
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  inputFencedCode = {%
    \ifx\relax#2\relax
      \VerbatimInput{#1}%
    \else
      \ifx\minted@jobname\undefined
        \ifx\lst@version\undefined
          \markdownRendererInputFencedCode{#1}{}%
```

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

```
        \else
          \lstinputlisting[language=#2]{#1}%
        \fi
```

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

```
      \else
        \inputminted{#2}{#1}%
      \fi
    \fi},
  horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  footnote = {\footnote{#1}}}}

\ifx\chapter\undefined
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}},
    headingFive = {\subparagraph{#1}}}}
\else
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}},
    headingSix = {\subparagraph{#1}}}}
\fi
```

There is a basic implementation for citations that uses the LaTeX `\cite` macro. There is also a more advanced implementation that uses the BibLaTeX `\autocites` and `\textcites` macros. This implementation will be used, when BibLaTeX is loaded.

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\def\markdownLaTeXBasicCitations#1#2#3#4{%
```

```
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#2\relax\else#2~\fi\cite[#3]{#4}%
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \expandafter\@gobble
  \fi\markdownLaTeXBasicCitations}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% 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}%
    \ifx\autocites\undefined
      \expandafter
      \markdownLaTeXBasicCitations
    \else
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    \fi},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \ifx\textcites\undefined
      \expandafter
      \markdownLaTeXBasicTextCitations
    \else
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    \fi}}}
```

3.3.5 Miscellanea

Unlike base LuaTeX, which only allows for a single registered function per a callback (see [1, Section 8.1]), the LaTeX 2$_\varepsilon$ format disables the `callback.register` method and exposes the `luatexbase.add_to_callback` and `luatexbase.remove_from_callback` methods that enable the user code to hook several functions on a single callback (see [4, Section 73.4]).

To make our code function with the LaTeX 2$_\varepsilon$ format, we need to redefine the `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros accordingly.

```
\let\markdownLuaRegisterIBCallbackPrevious
  \markdownLuaRegisterIBCallback
\let\markdownLuaUnregisterIBCallbackPrevious
  \markdownLuaUnregisterIBCallback
\renewcommand\markdownLuaRegisterIBCallback[1]{%
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.add_to_callback("process_input_buffer", #1, %
      "The markdown input processor")
  else
    \markdownLuaRegisterIBCallbackPrevious{#1}
  end}
\renewcommand\markdownLuaUnregisterIBCallback{%
  if luatexbase and luatexbase.add_to_callback then
    luatexbase.remove_from_callback("process_input_buffer",%
      "The markdown input processor")
  else
    \markdownLuaUnregisterIBCallbackPrevious;
  end}
```

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.

```
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
```

When there is no Lua support, then just load the plain TEX implementation.

```
\ifx\directlua\undefined
  \input markdown
\else
```

When there is Lua support, check if we can set the `process_input_buffer` LuaTEX callback.

```
  \directlua{%
    local function unescape(str)
      return (str:gsub("|", string.char(92))) end
    local old_callback = callback.find("process_input_buffer")
    callback.register("process_input_buffer", function() end)
    local new_callback = callback.find("process_input_buffer")
```

If we can not, we are probably using ConTeXt Mark IV. In ConTeXt Mark IV, the `process_input_buffer` callback is currently frozen (inaccessible from the user code) and, due to the lack of available documentation, it is unclear to me how to emulate it. As a workaround, we will force the plain TEX implementation to use the Lua shell escape bridge (see Section 3.2.4) by setting the `\markdownMode` macro to the value of `1`.

```
    if new_callback == false then
      tex.print(unescape([[|def|markdownMode{1}|input markdown]]))
```

If we can set the `process_input_buffer` LuaTEX callback, then just load the plain TEX implementation.

```
    else
      callback.register("process_input_buffer", old_callback)
      tex.print(unescape("|input markdown"))
    end}%
\fi
```

If the shell escape bridge is being used, define the `\markdownMakeOther` macro, so that the pipe character (`|`) is inactive during the scanning. This is necessary, since the character is active in ConTEXt.

```
\ifnum\markdownMode<2\relax
  \def\markdownMakeOther{%
    \catcode`|=12}%
\fi
```

3.4.1 Logging Facilities

The ConTEXt implementation redefines the plain TEX logging macros (see Section 3.2.1) to use the ConTEXt `\writestatus` macro.

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

3.4.2 Typesetting Markdown

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

```
\begingroup
```

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

```
  \catcode`\|=0%
  \catcode`\\=12%
  |gdef|startmarkdown{%
    |markdownReadAndConvert{\stopmarkdown}%
                           {|stopmarkdown}}%
|endgroup
```

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\def\markdownRendererLineBreakPrototype{\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}>}}%
\def\markdownRendererImagePrototype#1#2#3#4{%
  \placefigure[][fig:#1]{#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\em#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
\def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
\def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \ifx\relax#2\relax
    \typefile{#1}%
  \else
```

The code fence infostring is used as a name from the ConTEXt `\definetyping` command. This allows the user to set up code highlighting mapping as follows:

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

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

```
2224      \typefile[#2][]{#1}%
2225    \fi}%
2226 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
2227 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
2228 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
2229 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
2230 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
2231 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
2232 \def\markdownRendererHorizontalRulePrototype{%
2233   \blackrule[height=1pt, width=\hsize]}%
2234 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
2235 \stopmodule\protect
```

References

1. LUATEX DEVELOPMENT TEAM. *LuaTEX reference manual (0.98.3)* [online]. 2016 [visited on 2016-08-30]. Available from: http://www.luatex.org/svn/trunk/manual/luatex.pdf.
2. KNUTH, Donald Ervin. *The TEXbook*. 3rd ed. Addison-Westley, 1986. ISBN 0-201-13447-0.
3. IERUSALIMSCHY, Roberto. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. ISBN 978-85-903798-5-0.
4. BRAAMS, Johannes; CARLISLE, David; JEFFREY, Alan; LAMPORT, Leslie; MITTELBACH, Frank; ROWLEY, Chris; SCHÖPF, Rainer. *The LaTeX2ε Sources* [online]. 2016 [visited on 2016-06-02]. Available from: http://mirrors.ctan.org/macros/latex/base/source2e.pdf.