

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.5.0
April 10, 2017

Contents

1	Introduction	1	2.3	W _E T _E X Interface	26
1.1	About Markdown	1	2.4	ConT _E Xt Interface	35
1.2	Feedback	2			
1.3	Acknowledgements	2	3	Technical Documentation	36
1.4	Prerequisites	2	3.1	Lua Implementation	36
2	User Guide	5	3.2	Plain T _E X Implementation	82
2.1	Lua Interface	5	3.3	W _E T _E X Implementation	90
2.2	Plain T _E X Interface	12	3.4	ConT _E Xt Implementation	96

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 T_EX. These are provided both in the form of a Lua module and in the form of plain T_EX, W_ET_EX, and ConT_EXt macro packages that enable the direct inclusion of markdown documents inside T_EX documents.

Architecturally, the package consists of the Lunamark v0.5.0 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 T_EX, W_ET_EX, and ConT_EXt formats by Vít Novotný.

```
1 local metadata = {
2   version      = "2.5.0",
3   comment      = "A module for the conversion from markdown to plain TeX",
4   author       = "John MacFarlane, Hans Hagen, Vít Novotný",
5   copyright    = "2009-2017 John MacFarlane, Hans Hagen; " ..
```

```

6             "2016-2017 Vít Novotný",
7     license   = "LPPL 1.3"
8 }
9 if not modules then modules = { } end
10 modules['markdown'] = metadata

```

1.2 Feedback

Please use the markdown project page on GitHub¹ 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 T_EX-~~E~~_TX Stack Exchange².

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 T_EX part of the package draws inspiration from several sources including the source code of ~~E~~T_EX 2_ε, the minted package by Geoffrey M. Poore – which likewise tackles the issue of interfacing with an external interpreter from T_EX, 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 LuaT_EX 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 LuaT_EX \geq 0.72.0 (T_EXLive \geq 2013).

```

11     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 LuaT_EX (T_EXLive \geq 2008).

```
12      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 LuaT_EX (T_EXLive \geq 2008).

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

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

1.4.2 Plain T_EX Prerequisites

The plain T_EX part of the package requires that the plain T_EX 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 T_EX 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 LuaT_EX (T_EXLive \geq 2008).

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

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

The plain T_EX part of the package also requires that either the LuaT_EX `\directlua` primitive or the shell access file stream 18 is available in your T_EX engine. If only the shell access file stream is available in your T_EX engine (as is the case with pdfT_EX and X_ET_EX) or if you enforce the use of shell using the `\markdownMode` macro, then note the following:

- Unless your T_EX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.
- You will need to avoid the use of the `-output-directory` T_EX parameter when typesetting a document. The parameter causes auxiliary files to be written to a specified output directory, but the shell will be executed in the current directory. Things will not work out.

1.4.3 L^AT_EX Prerequisites

The L^AT_EX part of the package requires that the L^AT_EX 2_ε format is loaded,

```
14 \NeedsTeXFormat{LaTeX2e}%
```

all the plain T_EX prerequisites (see Section 1.4.2), and the following L^AT_EX 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*` L^AT_EX 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).

csvsimple A package that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.4).

1.4.4 ConT_EXt prerequisites

The ConT_EXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain T_EX prerequisites (see Section 1.4.2), and the following modules:

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.4).

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.

```
15 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.

```
16 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.

```
17 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.

```
18 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.

```
19 defaultOptions.blankBeforeHeading = false
```

`breakableBlockquotes=true, false` default: false

- `true` A blank line separates block quotes.
- `false` Blank lines in the middle of a block quote are ignored.

```
20 defaultOptions.breakableBlockquotes = false
```

`cacheDir=<directory name>` 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.

```
21 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.

22 `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.

23 `defaultOptions.citations = false`

`codeSpans=true, false`

default: true

- `true` Enable the code span syntax:

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

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

"This is a quote."

24 `defaultOptions.codeSpans = true`

`contentBlocks=true, false`

default: false

`true` Enable the iA Writer content blocks syntax extension [2]:

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

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

```
25 defaultOptions.contentBlocks = false
```

`contentBlocksLanguageMap=<filename>`

default: markdown-languages.json

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks. See Section 2.2.3.9 for more information.

```
26 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`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.

```
27 defaultOptions.definitionLists = 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>
'''
```

`false`

Disable the commonmark fenced code block extension.

28 `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.

```
29 defaultOptions.footnotes = 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.

```
30 defaultOptions.hashEnumerators = false
```

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

`true` Enable the recognition of HTML tags, block elements, comments, HTML instructions, and entities in the input. Tags, block elements (along with contents), HTML instructions, and comments will be ignored and HTML entities will be replaced with the corresponding Unicode codepoints.

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

```
31 defaultOptions.html = 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 interpreted by \TeX . This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
32 defaultOptions.hybrid = false
```

`inlineFootnotes=true, false` default: false

`true` Enable the pandoc inline footnote syntax extension:

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

`false` Disable the pandoc inline footnote syntax extension.

```
33 defaultOptions.inlineFootnotes = false
```

`preserveTabs=true, false` default: false

`true` Preserve all tabs in the input.

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

```
34 defaultOptions.preserveTabs = false
```

`smartEllipses=true, false` default: false

`true` Convert any ellipses in the input to the `\markdownRenderEllipsis` \TeX macro.

`false` Preserve all ellipses in the input.

```
35 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 `\markdownRenderOItemWithNumber` \TeX macro.

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

```
36 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.

```
37 defaultOptions.tightLists = true
```

`underscores=true, false`

default: true

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

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

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

```
38 defaultOptions.underscores = true
```

2.2 Plain T_EX Interface

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

```
39 \def\markdownLastModified{2017/04/10}%
40 \def\markdownVersion{2.5.0}%
```

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

```
\input markdown
```

It is expected that the special plain T_EX characters have the expected category codes, when `\inputting` 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.

```
41 \let\markdownBegin\relax
42 \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 corollary, 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 [3, p. 46]). As a corollary, 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 .

```
43 \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.

```
44 \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.

```
45 \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.

```
46 \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 \TeX package. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
47 \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.

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

```
48 \let\markdownOptionBlankBeforeBlockquote\undefined
49 \let\markdownOptionBlankBeforeCodeFence\undefined
50 \let\markdownOptionBlankBeforeHeading\undefined
51 \let\markdownOptionBreakableBlockquotes\undefined
52 \let\markdownOptionCitations\undefined
53 \let\markdownOptionCitationNbsps\undefined
54 \let\markdownOptionContentBlocks\undefined
55 \let\markdownOptionContentBlocksLanguageMap\undefined
56 \let\markdownOptionDefinitionLists\undefined
57 \let\markdownOptionFootnotes\undefined
58 \let\markdownOptionFencedCode\undefined
59 \let\markdownOptionHashEnumerators\undefined
60 \let\markdownOptionHtml\undefined
61 \let\markdownOptionHybrid\undefined
62 \let\markdownOptionInlineFootnotes\undefined
63 \let\markdownOptionPreserveTabs\undefined
64 \let\markdownOptionSmartEllipses\undefined
65 \let\markdownOptionStartNumber\undefined
66 \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.

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

2.2.3.2 Line Break Renderer The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

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

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

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

2.2.3.4 Non-breaking Space Renderer The `\markdownRendererNbsp` macro represents a non-breaking space.

```
73 \def\markdownRendererNbsp{%
74   \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`.

```
75 \def\markdownRendererLeftBrace{%
76   \markdownRendererLeftBracePrototype}%
77 \def\markdownRendererRightBrace{%
78   \markdownRendererRightBracePrototype}%
79 \def\markdownRendererDollarSign{%
80   \markdownRendererDollarSignPrototype}%
81 \def\markdownRendererPercentSign{%
82   \markdownRendererPercentSignPrototype}%
83 \def\markdownRendererAmpersand{%
84   \markdownRendererAmpersandPrototype}%
85 \def\markdownRendererUnderscore{%
86   \markdownRendererUnderscorePrototype}%
87 \def\markdownRendererHash{%
88   \markdownRendererHashPrototype}%
89 \def\markdownRendererCircumflex{%
90   \markdownRendererCircumflexPrototype}%
91 \def\markdownRendererBackslash{%
92   \markdownRendererBackslashPrototype}%
```



```

93 \def\markdownRendererTilde{%
94   \markdownRendererTildePrototype}%
95 \def\markdownRendererPipe{%
96   \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.

```

97 \def\markdownRendererCodeSpan{%
98   \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.

```

99 \def\markdownRendererLink{%
100   \markdownRendererLinkPrototype}%

```

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

```

101 \def\markdownRendererImage{%
102   \markdownRendererImagePrototype}%

```

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

```

103 \def\markdownRendererContentBlock{%
104   \markdownRendererContentBlockPrototype}%

```

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

```

105 \def\markdownRendererContentBlockOnlineImage{%
106   \markdownRendererContentBlockOnlineImagePrototype}%

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension *s*. If any `markdown-languages.json` file found by `kpathsea`³ contains a

³Local files take precedence. Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

record (k, v) , then a non-online-image content block with the filename extension s , $s:\text{lower}() = k$ is considered to be in a known programming language v .

The macro receives four arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local TeX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by [2] is a good starting point.

```
107 \def\markdownRendererContentBlockCode{%
108   \markdownRendererContentBlockCodePrototype}%
```

2.2.3.10 Bullet List Renderers The `\markdownRendererUllBegin` 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.

```
109 \def\markdownRendererUllBegin{%
110   \markdownRendererUllBeginPrototype}%
```

The `\markdownRendererUllBeginTight` 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.

```
111 \def\markdownRendererUllBeginTight{%
112   \markdownRendererUllBeginTightPrototype}%
```

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

```
113 \def\markdownRendererUllItem{%
114   \markdownRendererUllItemPrototype}%
```

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

```
115 \def\markdownRendererUllItemEnd{%
116   \markdownRendererUllItemEndPrototype}%
```

The `\markdownRendererUllEnd` 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.

```
117 \def\markdownRendererUllEnd{%
118   \markdownRendererUllEndPrototype}%
```

The `\markdownRendererUllEndTight` 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.

```

119 \def\markdownRendererU1EndTight{%
120   \markdownRendererU1EndTightPrototype}%

```

2.2.3.11 Ordered List Renderers The `\markdownRendererO1Begin` 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.

```

121 \def\markdownRendererO1Begin{%
122   \markdownRendererO1BeginPrototype}%

```

The `\markdownRendererO1BeginTight` 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.

```

123 \def\markdownRendererO1BeginTight{%
124   \markdownRendererO1BeginTightPrototype}%

```

The `\markdownRendererO1Item` 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.

```

125 \def\markdownRendererO1Item{%
126   \markdownRendererO1ItemPrototype}%

```

The `\markdownRendererO1ItemEnd` macro represents the end of an item in an ordered list. The macro receives no arguments.

```

127 \def\markdownRendererO1ItemEnd{%
128   \markdownRendererO1ItemEndPrototype}%

```

The `\markdownRendererO1ItemWithNumber` 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.

```

129 \def\markdownRendererO1ItemWithNumber{%
130   \markdownRendererO1ItemWithNumberPrototype}%

```

The `\markdownRendererO1End` 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.

```

131 \def\markdownRendererO1End{%
132   \markdownRendererO1EndPrototype}%

```

The `\markdownRendererO1EndTight` 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.

```

133 \def\markdownRendererO1EndTight{%
134   \markdownRendererO1EndTightPrototype}%

```

2.2.3.12 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.

```
135 \def\markdownRendererDlBegin{%  
136   \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.

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

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

```
139 \def\markdownRendererDlItem{%  
140   \markdownRendererDlItemPrototype}%
```

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

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

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

```
143 \def\markdownRendererDlDefinitionBegin{%  
144   \markdownRendererDlDefinitionBeginPrototype}%
```

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

```
145 \def\markdownRendererDlDefinitionEnd{%  
146   \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.

```
147 \def\markdownRendererDlEnd{%  
148   \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.

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

2.2.3.13 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.

```
151 \def\markdownRendererEmphasis{%  
152   \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.

```
153 \def\markdownRendererStrongEmphasis{%  
154   \markdownRendererStrongEmphasisPrototype}%
```

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

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

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

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

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

```
159 \def\markdownRendererInputVerbatim{%  
160   \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 containing the code block contents and to the code fence infostring.

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

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

```
163 \def\markdownRendererHeadingOne{%  
164   \markdownRendererHeadingOnePrototype}%
```

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

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

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

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

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

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

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

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

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

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

2.2.3.17 Horizontal Rule Renderer The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

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

2.2.3.18 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.

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

2.2.3.19 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.

```
179 \def\markdownRendererCite{%
180   \markdownRendererCitePrototype}%
```

2.2.3.20 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.

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

2.2.4 Token Renderer Prototypes

The following T_EX 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 L^AT_EX and ConT_EXt implementations (see sections 3.3 and 3.4).

```
183 \def\markdownRendererInterblockSeparatorPrototype{%
184 \def\markdownRendererLineBreakPrototype{%
185 \def\markdownRendererEllipsisPrototype{%
186 \def\markdownRendererNbspPrototype{%
187 \def\markdownRendererLeftBracePrototype{%
188 \def\markdownRendererRightBracePrototype{%
189 \def\markdownRendererDollarSignPrototype{%
190 \def\markdownRendererPercentSignPrototype{%
191 \def\markdownRendererAmpersandPrototype{%
192 \def\markdownRendererUnderscorePrototype{%
193 \def\markdownRendererHashPrototype{%
194 \def\markdownRendererCircumflexPrototype{%
195 \def\markdownRendererBackslashPrototype{%
196 \def\markdownRendererTildePrototype{%
197 \def\markdownRendererPipePrototype{%
198 \def\markdownRendererCodeSpanPrototype#1{%
199 \def\markdownRendererLinkPrototype#1#2#3#4{%
200 \def\markdownRendererImagePrototype#1#2#3#4{%
201 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
202 \def\markdownRendererContentBlockOnlineImagePrototype#1#2#3#4{%
203 \def\markdownRendererContentBlockCodePrototype#1#2#3#4{%
204 \def\markdownRendererUlBeginPrototype{%
205 \def\markdownRendererUlBeginTightPrototype{%
206 \def\markdownRendererUlItemPrototype{%
207 \def\markdownRendererUlItemEndPrototype{%
208 \def\markdownRendererUlEndPrototype{%
209 \def\markdownRendererUlEndTightPrototype{%
210 \def\markdownRendererOlBeginPrototype{%
211 \def\markdownRendererOlBeginTightPrototype{%
212 \def\markdownRendererOlItemPrototype{%
213 \def\markdownRendererOlItemWithNumberPrototype#1{%
214 \def\markdownRendererOlItemEndPrototype{%
215 \def\markdownRendererOlEndPrototype{%
216 \def\markdownRendererOlEndTightPrototype{%
```



```

217 \def\markdownRendererDlBeginPrototype{}%
218 \def\markdownRendererDlBeginTightPrototype{}%
219 \def\markdownRendererDlItemPrototype#1{}%
220 \def\markdownRendererDlItemEndPrototype{}%
221 \def\markdownRendererDlDefinitionBeginPrototype{}%
222 \def\markdownRendererDlDefinitionEndPrototype{}%
223 \def\markdownRendererDlEndPrototype{}%
224 \def\markdownRendererDlEndTightPrototype{}%
225 \def\markdownRendererEmphasisPrototype#1{}%
226 \def\markdownRendererStrongEmphasisPrototype#1{}%
227 \def\markdownRendererBlockQuoteBeginPrototype{}%
228 \def\markdownRendererBlockQuoteEndPrototype{}%
229 \def\markdownRendererInputVerbatimPrototype#1{}%
230 \def\markdownRendererInputFencedCodePrototype#1#2{}%
231 \def\markdownRendererHeadingOnePrototype#1{}%
232 \def\markdownRendererHeadingTwoPrototype#1{}%
233 \def\markdownRendererHeadingThreePrototype#1{}%
234 \def\markdownRendererHeadingFourPrototype#1{}%
235 \def\markdownRendererHeadingFivePrototype#1{}%
236 \def\markdownRendererHeadingSixPrototype#1{}%
237 \def\markdownRendererHorizontalRulePrototype{}%
238 \def\markdownRendererFootnotePrototype#1{}%
239 \def\markdownRendererCitePrototype#1{}%
240 \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.

```

241 \def\markdownInfo#1{}%
242 \def\markdownWarning#1{}%

```

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

```

243 \def\markdownError#1#2{}%

```

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

2.2.6 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a \TeX engine that does not support direct Lua access is starting to buffer a text. The plain \TeX implementation changes the category code of plain \TeX special characters to other, but there may be

other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```
244 \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 T_EX 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.

```
245 \let\markdownReadAndConvert\relax
```

```
246 \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*.

```
247 \catcode'\|=0\catcode'\=12%
248 |gdef|markdownBegin{%
249   |markdownReadAndConvert{\markdownEnd}%
250                               {\|markdownEnd}}%
251 |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 3.2.6), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

The `\markdownMode` macro specifies how the plain T_EX 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 T_EX implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

```
252 \ifx\markdownMode\undefined
253   \ifx\directlua\undefined
254     \def\markdownMode{0}%
255   \else
256     \def\markdownMode{2}%
257   \fi
258 \fi
```

The following macros are no longer a part of the plain \TeX interface and are only defined for backwards compatibility:

```
259 \def\markdownLuaRegisterIBCallback#1{\relax}%
260 \def\markdownLuaUnregisterIBCallback#1{\relax}%
```

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[\langle options \rangle]{markdown}
```

where *\langle options \rangle* are the \LaTeX interface options (see Section 2.3.2). Note that *\langle options \rangle* 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 $\text{\LaTeX} 2_{\epsilon}$ 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 influence this markdown document fragment.

```
261 \newenvironment{markdown}\relax\relax
262 \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:

<code>\documentclass{article}</code>	<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>	<code>\usepackage{markdown}</code>
<code>\begin{document}</code>	<code>\begin{document}</code>
<code>% ...</code>	<code>% ...</code>
<code>\begin{markdown}</code>	<code>\begin{markdown*}{smartEllipses}</code>
<code>_Hello_ **world** ...</code>	<code>_Hello_ **world** ...</code>
<code>\end{markdown}</code>	<code>\end{markdown*}</code>
<code>% ...</code>	<code>% ...</code>
<code>\end{document}</code>	<code>\end{document}</code>

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 influence this markdown document.

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

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

2.3.2 Options

The \LaTeX options are represented by a comma-delimited list of $\langle\langle key \rangle\rangle=\langle\langle value \rangle\rangle$ pairs. For boolean options, the $\langle\langle value \rangle\rangle$ part is optional, and $\langle\langle key \rangle\rangle$ will be interpreted as $\langle\langle key \rangle\rangle=true$.

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.

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

2.3.2.1 Plain T_EX Interface Options The following options map directly to the option macros exposed by the plain T_EX interface (see Section 2.2.2).

```

265 \RequirePackage{keyval}
266 \define@key{markdownOptions}{helperScriptFileName}{%
267   \def\markdownOptionHelperScriptFileName{#1}}%
268 \define@key{markdownOptions}{inputTempFileName}{%
269   \def\markdownOptionInputTempFileName{#1}}%
270 \define@key{markdownOptions}{outputTempFileName}{%
271   \def\markdownOptionOutputTempFileName{#1}}%
272 \define@key{markdownOptions}{blankBeforeBlockquote}[true]{%
273   \def\markdownOptionBlankBeforeBlockquote{#1}}%
274 \define@key{markdownOptions}{blankBeforeCodeFence}[true]{%
275   \def\markdownOptionBlankBeforeCodeFence{#1}}%
276 \define@key{markdownOptions}{blankBeforeHeading}[true]{%
277   \def\markdownOptionBlankBeforeHeading{#1}}%
278 \define@key{markdownOptions}{breakableBlockquotes}[true]{%
279   \def\markdownOptionBreakableBlockquotes{#1}}%
280 \define@key{markdownOptions}{cacheDir}{%
281   \def\markdownOptionCacheDir{#1}}%
282 \define@key{markdownOptions}{citations}[true]{%
283   \def\markdownOptionCitations{#1}}%
284 \define@key{markdownOptions}{citationNbsps}[true]{%
285   \def\markdownOptionCitationNbsps{#1}}%
286 \define@key{markdownOptions}{contentBlocks}[true]{%
287   \def\markdownOptionContentBlocks{#1}}%
288 \define@key{markdownOptions}{codeSpans}[true]{%
289   \def\markdownOptionCodeSpans{#1}}%
290 \define@key{markdownOptions}{contentBlocksLanguageMap}{%
291   \def\markdownOptionContentBlocksLanguageMap{#1}}%
292 \define@key{markdownOptions}{definitionLists}[true]{%
293   \def\markdownOptionDefinitionLists{#1}}%
294 \define@key{markdownOptions}{footnotes}[true]{%
295   \def\markdownOptionFootnotes{#1}}%
296 \define@key{markdownOptions}{fencedCode}[true]{%
297   \def\markdownOptionFencedCode{#1}}%
298 \define@key{markdownOptions}{hashEnumerators}[true]{%
299   \def\markdownOptionHashEnumerators{#1}}%
300 \define@key{markdownOptions}{html}[true]{%
301   \def\markdownOptionHtml{#1}}%
302 \define@key{markdownOptions}{hybrid}[true]{%
303   \def\markdownOptionHybrid{#1}}%
304 \define@key{markdownOptions}{inlineFootnotes}[true]{%
305   \def\markdownOptionInlineFootnotes{#1}}%
306 \define@key{markdownOptions}{preserveTabs}[true]{%
307   \def\markdownOptionPreserveTabs{#1}}%
308 \define@key{markdownOptions}{smartEllipses}[true]{%
309   \def\markdownOptionSmartEllipses{#1}}%

```

```

310 \define@key{markdownOptions}{startNumber}[true]{%
311   \def\markdownOptionStartNumber{#1}}%
312 \define@key{markdownOptions}{tightLists}[true]{%
313   \def\markdownOptionTightLists{#1}}%
314 \define@key{markdownOptions}{underscores}[true]{%
315   \def\markdownOptionUnderscores{#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).

```

316 \define@key{markdownRenderers}{interblockSeparator}{%
317   \renewcommand\markdownRendererInterblockSeparator{#1}}%
318 \define@key{markdownRenderers}{lineBreak}{%
319   \renewcommand\markdownRendererLineBreak{#1}}%
320 \define@key{markdownRenderers}{ellipsis}{%
321   \renewcommand\markdownRendererEllipsis{#1}}%
322 \define@key{markdownRenderers}{nbsp}{%
323   \renewcommand\markdownRendererNbsp{#1}}%
324 \define@key{markdownRenderers}{leftBrace}{%
325   \renewcommand\markdownRendererLeftBrace{#1}}%
326 \define@key{markdownRenderers}{rightBrace}{%
327   \renewcommand\markdownRendererRightBrace{#1}}%
328 \define@key{markdownRenderers}{dollarSign}{%
329   \renewcommand\markdownRendererDollarSign{#1}}%
330 \define@key{markdownRenderers}{percentSign}{%
331   \renewcommand\markdownRendererPercentSign{#1}}%
332 \define@key{markdownRenderers}{ampersand}{%
333   \renewcommand\markdownRendererAmpersand{#1}}%
334 \define@key{markdownRenderers}{underscore}{%
335   \renewcommand\markdownRendererUnderscore{#1}}%
336 \define@key{markdownRenderers}{hash}{%
337   \renewcommand\markdownRendererHash{#1}}%
338 \define@key{markdownRenderers}{circumflex}{%
339   \renewcommand\markdownRendererCircumflex{#1}}%

```

```

340 \define@key{markdownRenderers}{backslash}{%
341   \renewcommand\markdownRendererBackslash{#1}}%
342 \define@key{markdownRenderers}{tilde}{%
343   \renewcommand\markdownRendererTilde{#1}}%
344 \define@key{markdownRenderers}{pipe}{%
345   \renewcommand\markdownRendererPipe{#1}}%
346 \define@key{markdownRenderers}{codeSpan}{%
347   \renewcommand\markdownRendererCodeSpan[1]{#1}}%
348 \define@key{markdownRenderers}{link}{%
349   \renewcommand\markdownRendererLink[4]{#1}}%
350 \define@key{markdownRenderers}{contentBlock}{%
351   \renewcommand\markdownRendererContentBlock[4]{#1}}%
352 \define@key{markdownRenderers}{contentBlockOnlineImage}{%
353   \renewcommand\markdownRendererContentBlockOnlineImage[4]{#1}}%
354 \define@key{markdownRenderers}{contentBlockCode}{%
355   \renewcommand\markdownRendererContentBlockCode[5]{#1}}%
356 \define@key{markdownRenderers}{image}{%
357   \renewcommand\markdownRendererImage[4]{#1}}%
358 \define@key{markdownRenderers}{ulBegin}{%
359   \renewcommand\markdownRendererUlBegin{#1}}%
360 \define@key{markdownRenderers}{ulBeginTight}{%
361   \renewcommand\markdownRendererUlBeginTight{#1}}%
362 \define@key{markdownRenderers}{ulItem}{%
363   \renewcommand\markdownRendererUlItem{#1}}%
364 \define@key{markdownRenderers}{ulItemEnd}{%
365   \renewcommand\markdownRendererUlItemEnd{#1}}%
366 \define@key{markdownRenderers}{ulEnd}{%
367   \renewcommand\markdownRendererUlEnd{#1}}%
368 \define@key{markdownRenderers}{ulEndTight}{%
369   \renewcommand\markdownRendererUlEndTight{#1}}%
370 \define@key{markdownRenderers}{olBegin}{%
371   \renewcommand\markdownRendererOlBegin{#1}}%
372 \define@key{markdownRenderers}{olBeginTight}{%
373   \renewcommand\markdownRendererOlBeginTight{#1}}%
374 \define@key{markdownRenderers}{olItem}{%
375   \renewcommand\markdownRendererOlItem{#1}}%
376 \define@key{markdownRenderers}{olItemWithNumber}{%
377   \renewcommand\markdownRendererOlItemWithNumber[1]{#1}}%
378 \define@key{markdownRenderers}{olItemEnd}{%
379   \renewcommand\markdownRendererOlItemEnd{#1}}%
380 \define@key{markdownRenderers}{olEnd}{%
381   \renewcommand\markdownRendererOlEnd{#1}}%
382 \define@key{markdownRenderers}{olEndTight}{%
383   \renewcommand\markdownRendererOlEndTight{#1}}%
384 \define@key{markdownRenderers}{dlBegin}{%
385   \renewcommand\markdownRendererDlBegin{#1}}%
386 \define@key{markdownRenderers}{dlBeginTight}{%

```

```

387 \renewcommand\markdownRendererDlBeginTight{#1}}%
388 \define@key{markdownRenderers}{dlItem}{%
389 \renewcommand\markdownRendererDlItem[1]{#1}}%
390 \define@key{markdownRenderers}{dlItemEnd}{%
391 \renewcommand\markdownRendererDlItemEnd{#1}}%
392 \define@key{markdownRenderers}{dlDefinitionBegin}{%
393 \renewcommand\markdownRendererDlDefinitionBegin{#1}}%
394 \define@key{markdownRenderers}{dlDefinitionEnd}{%
395 \renewcommand\markdownRendererDlDefinitionEnd{#1}}%
396 \define@key{markdownRenderers}{dlEnd}{%
397 \renewcommand\markdownRendererDlEnd{#1}}%
398 \define@key{markdownRenderers}{dlEndTight}{%
399 \renewcommand\markdownRendererDlEndTight{#1}}%
400 \define@key{markdownRenderers}{emphasis}{%
401 \renewcommand\markdownRendererEmphasis[1]{#1}}%
402 \define@key{markdownRenderers}{strongEmphasis}{%
403 \renewcommand\markdownRendererStrongEmphasis[1]{#1}}%
404 \define@key{markdownRenderers}{blockquoteBegin}{%
405 \renewcommand\markdownRendererBlockQuoteBegin{#1}}%
406 \define@key{markdownRenderers}{blockquoteEnd}{%
407 \renewcommand\markdownRendererBlockQuoteEnd{#1}}%
408 \define@key{markdownRenderers}{inputVerbatim}{%
409 \renewcommand\markdownRendererInputVerbatim[1]{#1}}%
410 \define@key{markdownRenderers}{inputFencedCode}{%
411 \renewcommand\markdownRendererInputFencedCode[2]{#1}}%
412 \define@key{markdownRenderers}{headingOne}{%
413 \renewcommand\markdownRendererHeadingOne[1]{#1}}%
414 \define@key{markdownRenderers}{headingTwo}{%
415 \renewcommand\markdownRendererHeadingTwo[1]{#1}}%
416 \define@key{markdownRenderers}{headingThree}{%
417 \renewcommand\markdownRendererHeadingThree[1]{#1}}%
418 \define@key{markdownRenderers}{headingFour}{%
419 \renewcommand\markdownRendererHeadingFour[1]{#1}}%
420 \define@key{markdownRenderers}{headingFive}{%
421 \renewcommand\markdownRendererHeadingFive[1]{#1}}%
422 \define@key{markdownRenderers}{headingSix}{%
423 \renewcommand\markdownRendererHeadingSix[1]{#1}}%
424 \define@key{markdownRenderers}{horizontalRule}{%
425 \renewcommand\markdownRendererHorizontalRule{#1}}%
426 \define@key{markdownRenderers}{footnote}{%
427 \renewcommand\markdownRendererFootnote[1]{#1}}%
428 \define@key{markdownRenderers}{cite}{%
429 \renewcommand\markdownRendererCite[1]{#1}}%
430 \define@key{markdownRenderers}{textCite}{%
431 \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 T_EX Markdown Token Renderer Prototypes The \TeX 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 T_EX interface (see Section 2.2.4).

```
432 \define@key{markdownRendererPrototypes}{interblockSeparator}{%
433   \renewcommand\markdownRendererInterblockSeparatorPrototype{#1}}%
434 \define@key{markdownRendererPrototypes}{lineBreak}{%
435   \renewcommand\markdownRendererLineBreakPrototype{#1}}%
436 \define@key{markdownRendererPrototypes}{ellipsis}{%
437   \renewcommand\markdownRendererEllipsisPrototype{#1}}%
438 \define@key{markdownRendererPrototypes}{nbsp}{%
439   \renewcommand\markdownRendererNbspPrototype{#1}}%
440 \define@key{markdownRendererPrototypes}{leftBrace}{%
441   \renewcommand\markdownRendererLeftBracePrototype{#1}}%
442 \define@key{markdownRendererPrototypes}{rightBrace}{%
443   \renewcommand\markdownRendererRightBracePrototype{#1}}%
444 \define@key{markdownRendererPrototypes}{dollarSign}{%
445   \renewcommand\markdownRendererDollarSignPrototype{#1}}%
446 \define@key{markdownRendererPrototypes}{percentSign}{%
447   \renewcommand\markdownRendererPercentSignPrototype{#1}}%
448 \define@key{markdownRendererPrototypes}{ampersand}{%
449   \renewcommand\markdownRendererAmpersandPrototype{#1}}%
450 \define@key{markdownRendererPrototypes}{underscore}{%
451   \renewcommand\markdownRendererUnderscorePrototype{#1}}%
452 \define@key{markdownRendererPrototypes}{hash}{%
453   \renewcommand\markdownRendererHashPrototype{#1}}%
454 \define@key{markdownRendererPrototypes}{circumflex}{%
455   \renewcommand\markdownRendererCircumflexPrototype{#1}}%
456 \define@key{markdownRendererPrototypes}{backslash}{%
457   \renewcommand\markdownRendererBackslashPrototype{#1}}%
458 \define@key{markdownRendererPrototypes}{tilde}{%
459   \renewcommand\markdownRendererTildePrototype{#1}}%
460 \define@key{markdownRendererPrototypes}{pipe}{%
461   \renewcommand\markdownRendererPipePrototype{#1}}%
462 \define@key{markdownRendererPrototypes}{codeSpan}{%
```



```

463 \renewcommand\markdownRendererCodeSpanPrototype[1]{#1}}%
464 \define@key{markdownRendererPrototypes}{link}{%
465 \renewcommand\markdownRendererLinkPrototype[4]{#1}}%
466 \define@key{markdownRendererPrototypes}{contentBlock}{%
467 \renewcommand\markdownRendererContentBlockPrototype[4]{#1}}%
468 \define@key{markdownRendererPrototypes}{contentBlockOnlineImage}{%
469 \renewcommand\markdownRendererContentBlockOnlineImagePrototype[4]{#1}}%
470 \define@key{markdownRendererPrototypes}{contentBlockCode}{%
471 \renewcommand\markdownRendererContentBlockCodePrototype[5]{#1}}%
472 \define@key{markdownRendererPrototypes}{image}{%
473 \renewcommand\markdownRendererImagePrototype[4]{#1}}%
474 \define@key{markdownRendererPrototypes}{ulBegin}{%
475 \renewcommand\markdownRendererUlBeginPrototype{#1}}%
476 \define@key{markdownRendererPrototypes}{ulBeginTight}{%
477 \renewcommand\markdownRendererUlBeginTightPrototype{#1}}%
478 \define@key{markdownRendererPrototypes}{ulItem}{%
479 \renewcommand\markdownRendererUlItemPrototype{#1}}%
480 \define@key{markdownRendererPrototypes}{ulItemEnd}{%
481 \renewcommand\markdownRendererUlItemEndPrototype{#1}}%
482 \define@key{markdownRendererPrototypes}{ulEnd}{%
483 \renewcommand\markdownRendererUlEndPrototype{#1}}%
484 \define@key{markdownRendererPrototypes}{ulEndTight}{%
485 \renewcommand\markdownRendererUlEndTightPrototype{#1}}%
486 \define@key{markdownRendererPrototypes}{olBegin}{%
487 \renewcommand\markdownRendererOlBeginPrototype{#1}}%
488 \define@key{markdownRendererPrototypes}{olBeginTight}{%
489 \renewcommand\markdownRendererOlBeginTightPrototype{#1}}%
490 \define@key{markdownRendererPrototypes}{olItem}{%
491 \renewcommand\markdownRendererOlItemPrototype{#1}}%
492 \define@key{markdownRendererPrototypes}{olItemWithNumber}{%
493 \renewcommand\markdownRendererOlItemWithNumberPrototype[1]{#1}}%
494 \define@key{markdownRendererPrototypes}{olItemEnd}{%
495 \renewcommand\markdownRendererOlItemEndPrototype{#1}}%
496 \define@key{markdownRendererPrototypes}{olEnd}{%
497 \renewcommand\markdownRendererOlEndPrototype{#1}}%
498 \define@key{markdownRendererPrototypes}{olEndTight}{%
499 \renewcommand\markdownRendererOlEndTightPrototype{#1}}%
500 \define@key{markdownRendererPrototypes}{dlBegin}{%
501 \renewcommand\markdownRendererDlBeginPrototype{#1}}%
502 \define@key{markdownRendererPrototypes}{dlBeginTight}{%
503 \renewcommand\markdownRendererDlBeginTightPrototype{#1}}%
504 \define@key{markdownRendererPrototypes}{dlItem}{%
505 \renewcommand\markdownRendererDlItemPrototype[1]{#1}}%
506 \define@key{markdownRendererPrototypes}{dlItemEnd}{%
507 \renewcommand\markdownRendererDlItemEndPrototype{#1}}%
508 \define@key{markdownRendererPrototypes}{dlDefinitionBegin}{%
509 \renewcommand\markdownRendererDlDefinitionBeginPrototype{#1}}%

```

```

510 \define@key{markdownRendererPrototypes}{dlDefinitionEnd}{%
511   \renewcommand\markdownRendererDlDefinitionEndPrototype{#1}}%
512 \define@key{markdownRendererPrototypes}{dlEnd}{%
513   \renewcommand\markdownRendererDlEndPrototype{#1}}%
514 \define@key{markdownRendererPrototypes}{dlEndTight}{%
515   \renewcommand\markdownRendererDlEndTightPrototype{#1}}%
516 \define@key{markdownRendererPrototypes}{emphasis}{%
517   \renewcommand\markdownRendererEmphasisPrototype[1]{#1}}%
518 \define@key{markdownRendererPrototypes}{strongEmphasis}{%
519   \renewcommand\markdownRendererStrongEmphasisPrototype[1]{#1}}%
520 \define@key{markdownRendererPrototypes}{blockQuoteBegin}{%
521   \renewcommand\markdownRendererBlockQuoteBeginPrototype{#1}}%
522 \define@key{markdownRendererPrototypes}{blockQuoteEnd}{%
523   \renewcommand\markdownRendererBlockQuoteEndPrototype{#1}}%
524 \define@key{markdownRendererPrototypes}{inputVerbatim}{%
525   \renewcommand\markdownRendererInputVerbatimPrototype[1]{#1}}%
526 \define@key{markdownRendererPrototypes}{inputFencedCode}{%
527   \renewcommand\markdownRendererInputFencedCodePrototype[2]{#1}}%
528 \define@key{markdownRendererPrototypes}{headingOne}{%
529   \renewcommand\markdownRendererHeadingOnePrototype[1]{#1}}%
530 \define@key{markdownRendererPrototypes}{headingTwo}{%
531   \renewcommand\markdownRendererHeadingTwoPrototype[1]{#1}}%
532 \define@key{markdownRendererPrototypes}{headingThree}{%
533   \renewcommand\markdownRendererHeadingThreePrototype[1]{#1}}%
534 \define@key{markdownRendererPrototypes}{headingFour}{%
535   \renewcommand\markdownRendererHeadingFourPrototype[1]{#1}}%
536 \define@key{markdownRendererPrototypes}{headingFive}{%
537   \renewcommand\markdownRendererHeadingFivePrototype[1]{#1}}%
538 \define@key{markdownRendererPrototypes}{headingSix}{%
539   \renewcommand\markdownRendererHeadingSixPrototype[1]{#1}}%
540 \define@key{markdownRendererPrototypes}{horizontalRule}{%
541   \renewcommand\markdownRendererHorizontalRulePrototype{#1}}%
542 \define@key{markdownRendererPrototypes}{footnote}{%
543   \renewcommand\markdownRendererFootnotePrototype[1]{#1}}%
544 \define@key{markdownRendererPrototypes}{cite}{%
545   \renewcommand\markdownRendererCitePrototype[1]{#1}}%
546 \define@key{markdownRendererPrototypes}{textCite}{%
547   \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 ConT_EXt Interface

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

```
548 \writestatus{loading}{ConTEXt User Module / markdown}%  
549 \unprotect
```

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

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

It is expected that the special plain T_EX 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.

```
550 \let\startmarkdown\relax  
551 \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 T_EX interface.

The following example ConT_EXt 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).

```
552 local upper, gsub, format, length =
553   string.upper, string.gsub, string.format, string.len
554 local concat = table.concat
555 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
556   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
557   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 plain \TeX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
558 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.

```
559 function util.err(msg, exit_code)
560   io.stderr:write("markdown.lua: " .. msg .. "\n")
561   os.exit(exit_code or 1)
562 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.

```
563 function util.cache(dir, string, salt, transform, suffix)
564   local digest = md5.sumhexa(string .. (salt or ""))
565   local name = util.pathname(dir, digest .. suffix)
566   local file = io.open(name, "r")
567   if file == nil then -- If no cache entry exists, then create a new one.
568     local file = assert(io.open(name, "w"))
```

```

569     local result = string
570     if transform ~= nil then
571         result = transform(result)
572     end
573     assert(file:write(result))
574     assert(file:close())
575 end
576 return name
577 end

```

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

```

578 function util.table_copy(t)
579     local u = { }
580     for k, v in pairs(t) do u[k] = v end
581     return setmetatable(u, getmetatable(t))
582 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 [4, Chapter 21].

```

583 function util.expand_tabs_in_line(s, tabstop)
584     local tab = tabstop or 4
585     local corr = 0
586     return (s:gsub("()\t", function(p)
587         local sp = tab - (p - 1 + corr) % tab
588         corr = corr - 1 + sp
589         return string.rep(" ", sp)
590     end))
591 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.

```

592 function util.walk(t, f)
593     local typ = type(t)
594     if typ == "string" then
595         f(t)
596     elseif typ == "table" then
597         local i = 1
598         local n
599         n = t[i]
600         while n do
601             util.walk(n, f)
602             i = i + 1
603             n = t[i]
604         end
605     elseif typ == "function" then

```

```

606     local ok, val = pcall(t)
607     if ok then
608         util.walk(val,f)
609     end
610 else
611     f(tostring(t))
612 end
613 end

```

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

```

614 function util.flatten(ary)
615     local new = {}
616     for _,v in ipairs(ary) do
617         if type(v) == "table" then
618             for _,w in ipairs(util.flatten(v)) do
619                 new[#new + 1] = w
620             end
621         else
622             new[#new + 1] = v
623         end
624     end
625     return new
626 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.

```

627 function util.rope_to_string(rope)
628     local buffer = {}
629     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
630     return table.concat(buffer)
631 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.

```

632 function util.rope_last(rope)
633     if #rope == 0 then
634         return nil
635     else
636         local l = rope[#rope]
637         if type(l) == "table" then
638             return util.rope_last(l)
639         else
640             return l
641         end
642     end
643 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 $1 \leq i \leq \#ary$.

```

644 function util.intersperse(ary, x)
645   local new = {}
646   local l = #ary
647   for i,v in ipairs(ary) do
648     local n = #new
649     new[n + 1] = v
650     if i ~= l then
651       new[n + 2] = x
652     end
653   end
654   return new
655 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 $1 \leq i \leq \#ary$.

```

656 function util.map(ary, f)
657   local new = {}
658   for i,v in ipairs(ary) do
659     new[i] = f(v)
660   end
661   return new
662 end

```

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

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

```

663 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

664   local char_escapes_list = ""
665   for i,_ in pairs(char_escapes) do
666     char_escapes_list = char_escapes_list .. i
667   end

```

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

```

668   local escapable = S(char_escapes_list) / char_escapes

```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{(k,v) \in \text{string_escapes}} P(k) / v + \text{escapable}$$

capture that replaces any occurrence of the string `k` with the string `v` for each $(k, v) \in \text{string_escapes}$. Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corollary, the strings always take precedence over the characters.

```
669 if string_escapes then
670   for k,v in pairs(string_escapes) do
671     escapable = P(k) / v + escapable
672   end
673 end
```

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

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

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

```
675 return function(s)
676   return lpeg.match(escape_string, s)
677 end
678 end
```

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

```
679 function util.pathname(dir, file)
680   if #dir == 0 then
681     return file
682   else
683     return dir .. "/" .. file
684   end
685 end
```

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
686 local entities = {}
687
688 local character_entities = {
689   ["quot"] = 0x0022,
690   ["amp"] = 0x0026,
691   ["apos"] = 0x0027,
692   ["lt"] = 0x003C,
693   ["gt"] = 0x003E,
694   ["nbsp"] = 160,
695   ["iexcl"] = 0x00A1,
696   ["cent"] = 0x00A2,
697   ["pound"] = 0x00A3,
```



```

698 ["curren"] = 0x00A4,
699 ["yen"] = 0x00A5,
700 ["brvbar"] = 0x00A6,
701 ["sect"] = 0x00A7,
702 ["uml"] = 0x00A8,
703 ["copy"] = 0x00A9,
704 ["ordf"] = 0x00AA,
705 ["laquo"] = 0x00AB,
706 ["not"] = 0x00AC,
707 ["shy"] = 173,
708 ["reg"] = 0x00AE,
709 ["macr"] = 0x00AF,
710 ["deg"] = 0x00B0,
711 ["plusmn"] = 0x00B1,
712 ["sup2"] = 0x00B2,
713 ["sup3"] = 0x00B3,
714 ["acute"] = 0x00B4,
715 ["micro"] = 0x00B5,
716 ["para"] = 0x00B6,
717 ["middot"] = 0x00B7,
718 ["cedil"] = 0x00B8,
719 ["sup1"] = 0x00B9,
720 ["ordm"] = 0x00BA,
721 ["raquo"] = 0x00BB,
722 ["frac14"] = 0x00BC,
723 ["frac12"] = 0x00BD,
724 ["frac34"] = 0x00BE,
725 ["quest"] = 0x00BF,
726 ["Agrave"] = 0x00C0,
727 ["Aacute"] = 0x00C1,
728 ["Acirc"] = 0x00C2,
729 ["Atilde"] = 0x00C3,
730 ["Auml"] = 0x00C4,
731 ["Aring"] = 0x00C5,
732 ["AElig"] = 0x00C6,
733 ["Ccedil"] = 0x00C7,
734 ["Egrave"] = 0x00C8,
735 ["Eacute"] = 0x00C9,
736 ["Ecirc"] = 0x00CA,
737 ["Euml"] = 0x00CB,
738 ["Igrave"] = 0x00CC,
739 ["Iacute"] = 0x00CD,
740 ["Icirc"] = 0x00CE,
741 ["Iuml"] = 0x00CF,
742 ["ETH"] = 0x00D0,
743 ["Ntilde"] = 0x00D1,
744 ["Ograve"] = 0x00D2,

```

```

745 ["Oacute"] = 0x00D3,
746 ["Ocirc"] = 0x00D4,
747 ["Otilde"] = 0x00D5,
748 ["Ouml"] = 0x00D6,
749 ["times"] = 0x00D7,
750 ["Oslash"] = 0x00D8,
751 ["Ugrave"] = 0x00D9,
752 ["Uacute"] = 0x00DA,
753 ["Ucirc"] = 0x00DB,
754 ["Uuml"] = 0x00DC,
755 ["Yacute"] = 0x00DD,
756 ["THORN"] = 0x00DE,
757 ["szlig"] = 0x00DF,
758 ["agrave"] = 0x00E0,
759 ["aacute"] = 0x00E1,
760 ["acirc"] = 0x00E2,
761 ["atilde"] = 0x00E3,
762 ["auml"] = 0x00E4,
763 ["aring"] = 0x00E5,
764 ["aelig"] = 0x00E6,
765 ["ccedil"] = 0x00E7,
766 ["egrave"] = 0x00E8,
767 ["eacute"] = 0x00E9,
768 ["ecirc"] = 0x00EA,
769 ["euml"] = 0x00EB,
770 ["igrave"] = 0x00EC,
771 ["iacute"] = 0x00ED,
772 ["icirc"] = 0x00EE,
773 ["iuml"] = 0x00EF,
774 ["eth"] = 0x00F0,
775 ["ntilde"] = 0x00F1,
776 ["ograve"] = 0x00F2,
777 ["oacute"] = 0x00F3,
778 ["ocirc"] = 0x00F4,
779 ["otilde"] = 0x00F5,
780 ["ouml"] = 0x00F6,
781 ["divide"] = 0x00F7,
782 ["oslash"] = 0x00F8,
783 ["ugrave"] = 0x00F9,
784 ["uacute"] = 0x00FA,
785 ["ucirc"] = 0x00FB,
786 ["uuml"] = 0x00FC,
787 ["yacute"] = 0x00FD,
788 ["thorn"] = 0x00FE,
789 ["yuml"] = 0x00FF,
790 ["OElig"] = 0x0152,
791 ["oelig"] = 0x0153,

```

```

792 ["Scaron"] = 0x0160,
793 ["scaron"] = 0x0161,
794 ["Yuml"] = 0x0178,
795 ["fnof"] = 0x0192,
796 ["circ"] = 0x02C6,
797 ["tilde"] = 0x02DC,
798 ["Alpha"] = 0x0391,
799 ["Beta"] = 0x0392,
800 ["Gamma"] = 0x0393,
801 ["Delta"] = 0x0394,
802 ["Epsilon"] = 0x0395,
803 ["Zeta"] = 0x0396,
804 ["Eta"] = 0x0397,
805 ["Theta"] = 0x0398,
806 ["Iota"] = 0x0399,
807 ["Kappa"] = 0x039A,
808 ["Lambda"] = 0x039B,
809 ["Mu"] = 0x039C,
810 ["Nu"] = 0x039D,
811 ["Xi"] = 0x039E,
812 ["Omicron"] = 0x039F,
813 ["Pi"] = 0x03A0,
814 ["Rho"] = 0x03A1,
815 ["Sigma"] = 0x03A3,
816 ["Tau"] = 0x03A4,
817 ["Upsilon"] = 0x03A5,
818 ["Phi"] = 0x03A6,
819 ["Chi"] = 0x03A7,
820 ["Psi"] = 0x03A8,
821 ["Omega"] = 0x03A9,
822 ["alpha"] = 0x03B1,
823 ["beta"] = 0x03B2,
824 ["gamma"] = 0x03B3,
825 ["delta"] = 0x03B4,
826 ["epsilon"] = 0x03B5,
827 ["zeta"] = 0x03B6,
828 ["eta"] = 0x03B7,
829 ["theta"] = 0x03B8,
830 ["iota"] = 0x03B9,
831 ["kappa"] = 0x03BA,
832 ["lambda"] = 0x03BB,
833 ["mu"] = 0x03BC,
834 ["nu"] = 0x03BD,
835 ["xi"] = 0x03BE,
836 ["omicron"] = 0x03BF,
837 ["pi"] = 0x03C0,
838 ["rho"] = 0x03C1,

```

```

839 ["sigmaf"] = 0x03C2,
840 ["sigma"] = 0x03C3,
841 ["tau"] = 0x03C4,
842 ["upsilon"] = 0x03C5,
843 ["phi"] = 0x03C6,
844 ["chi"] = 0x03C7,
845 ["psi"] = 0x03C8,
846 ["omega"] = 0x03C9,
847 ["thetasym"] = 0x03D1,
848 ["upsih"] = 0x03D2,
849 ["piv"] = 0x03D6,
850 ["ensp"] = 0x2002,
851 ["emsp"] = 0x2003,
852 ["thinsp"] = 0x2009,
853 ["ndash"] = 0x2013,
854 ["mdash"] = 0x2014,
855 ["lsquo"] = 0x2018,
856 ["rsquo"] = 0x2019,
857 ["sbquo"] = 0x201A,
858 ["ldquo"] = 0x201C,
859 ["rdquo"] = 0x201D,
860 ["bdquo"] = 0x201E,
861 ["dagger"] = 0x2020,
862 ["Dagger"] = 0x2021,
863 ["bull"] = 0x2022,
864 ["hellip"] = 0x2026,
865 ["permil"] = 0x2030,
866 ["prime"] = 0x2032,
867 ["Prime"] = 0x2033,
868 ["lsaquo"] = 0x2039,
869 ["rsaquo"] = 0x203A,
870 ["oline"] = 0x203E,
871 ["frasl"] = 0x2044,
872 ["euro"] = 0x20AC,
873 ["image"] = 0x2111,
874 ["weierp"] = 0x2118,
875 ["real"] = 0x211C,
876 ["trade"] = 0x2122,
877 ["alefsym"] = 0x2135,
878 ["larr"] = 0x2190,
879 ["uarr"] = 0x2191,
880 ["rarr"] = 0x2192,
881 ["darr"] = 0x2193,
882 ["harr"] = 0x2194,
883 ["crarr"] = 0x21B5,
884 ["lArr"] = 0x21D0,
885 ["uArr"] = 0x21D1,

```

```

886 ["rArr"] = 0x21D2,
887 ["dArr"] = 0x21D3,
888 ["hArr"] = 0x21D4,
889 ["forall"] = 0x2200,
890 ["part"] = 0x2202,
891 ["exist"] = 0x2203,
892 ["empty"] = 0x2205,
893 ["nabla"] = 0x2207,
894 ["isin"] = 0x2208,
895 ["notin"] = 0x2209,
896 ["ni"] = 0x220B,
897 ["prod"] = 0x220F,
898 ["sum"] = 0x2211,
899 ["minus"] = 0x2212,
900 ["lowast"] = 0x2217,
901 ["radic"] = 0x221A,
902 ["prop"] = 0x221D,
903 ["infin"] = 0x221E,
904 ["ang"] = 0x2220,
905 ["and"] = 0x2227,
906 ["or"] = 0x2228,
907 ["cap"] = 0x2229,
908 ["cup"] = 0x222A,
909 ["int"] = 0x222B,
910 ["there4"] = 0x2234,
911 ["sim"] = 0x223C,
912 ["cong"] = 0x2245,
913 ["asymp"] = 0x2248,
914 ["ne"] = 0x2260,
915 ["equiv"] = 0x2261,
916 ["le"] = 0x2264,
917 ["ge"] = 0x2265,
918 ["sub"] = 0x2282,
919 ["sup"] = 0x2283,
920 ["nsup"] = 0x2284,
921 ["sube"] = 0x2286,
922 ["supe"] = 0x2287,
923 ["oplus"] = 0x2295,
924 ["otimes"] = 0x2297,
925 ["perp"] = 0x22A5,
926 ["sdot"] = 0x22C5,
927 ["lceil"] = 0x2308,
928 ["rceil"] = 0x2309,
929 ["lfloor"] = 0x230A,
930 ["rfloor"] = 0x230B,
931 ["lang"] = 0x27E8,
932 ["rang"] = 0x27E9,

```

```

933  ["loz"] = 0x25CA,
934  ["spades"] = 0x2660,
935  ["clubs"] = 0x2663,
936  ["hearts"] = 0x2665,
937  ["diams"] = 0x2666,
938 }

```

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

```

939 function entities.dec_entity(s)
940   return unicode.utf8.char(tonumber(s))
941 end

```

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

```

942 function entities.hex_entity(s)
943   return unicode.utf8.char(tonumber("0x"..s))
944 end

```

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

```

945 function entities.char_entity(s)
946   local n = character_entities[s]
947   return unicode.utf8.char(n)
948 end

```

3.1.3 Plain T_EX Writer

This section documents the `writer` object, which implements the routines for producing the T_EX output. The object is an amalgamate of the generic, T_EX, L^AT_EX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark 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.

```

949 M.writer = {}

```

The `writer.new` method creates and returns a new T_EX 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`.

```

950 function M.writer.new(options)

```

```

951  local self = {}
952  options = options or {}
    Make the options table inherit from the defaultOptions table.
953  setmetatable(options, { __index = function (_, key)
954      return defaultOptions[key] end })
    Define writer->suffix as the suffix of the produced cache files.
955  self.suffix = ".tex"
    Define writer->space as the output format of a space character.
956  self.space = " "
    Define writer->nbsp as the output format of a non-breaking space character.
957  self.nbsp = "\\markdownRendererNbsp{}"
    Define writer->plain as a function that will transform an input plain text block s
    to the output format.
958  function self.plain(s)
959      return s
960  end
    Define writer->paragraph as a function that will transform an input paragraph s
    to the output format.
961  function self.paragraph(s)
962      return s
963  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.
964  function self.pack(name)
965      return [[\input"']] .. name .. [[\"relax]]
966  end
    Define writer->interblocksep as the output format of a block element separator.
967  self.interblocksep = "\\markdownRendererInterblockSeparator\n{}"
    Define writer->eof as the end of file marker in the output format.
968  self.eof = [[\relax]]
    Define writer->linebreak as the output format of a forced line break.
969  self.linebreak = "\\markdownRendererLineBreak\n{}"
    Define writer->ellipsis as the output format of an ellipsis.
970  self.ellipsis = "\\markdownRendererEllipsis{}"
    Define writer->hrule as the output format of a horizontal rule.
971  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.

```

972 local escaped_chars = {
973   ["{"] = "\\markdownRendererLeftBrace{}",
974   ["}"] = "\\markdownRendererRightBrace{}",
975   ["$"] = "\\markdownRendererDollarSign{}",
976   ["%"] = "\\markdownRendererPercentSign{}",
977   ["&"] = "\\markdownRendererAmpersand{}",
978   ["_"] = "\\markdownRendererUnderscore{}",
979   ["#"] = "\\markdownRendererHash{}",
980   ["^"] = "\\markdownRendererCircumflex{}",
981   ["\\"] = "\\markdownRendererBackslash{}",
982   ["~"] = "\\markdownRendererTilde{}",
983   ["|"] = "\\markdownRendererPipe{}", }
984 local escaped_minimal_chars = {
985   ["{"] = "\\markdownRendererLeftBrace{}",
986   ["}"] = "\\markdownRendererRightBrace{}",
987   ["%"] = "\\markdownRendererPercentSign{}",
988   ["\\"] = "\\markdownRendererBackslash{}", }
989 local escaped_minimal_strings = {
990   ["^^"] = "\\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`.

```

991 local escape = util.escaper(escaped_chars)
992 local escape_minimal = util.escaper(escaped_minimal_chars,
993   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.

```

994 if options.hybrid then
995   self.string = function(s) return s end
996   self.uri = function(u) return u end
997 else
998   self.string = escape
999   self.uri = escape_minimal
1000 end

```

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

```

1001 function self.code(s)

```



```

1002     return {"\\markdownRendererCodeSpan{" ,escape(s),""}
1003 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.

```

1004 function self.link(lab,src,tit)
1005     return {"\\markdownRendererLink{" ,lab,""}
1006             {"",self.string(src),""},
1007             {"",self.uri(src),""},
1008             {"",self.string(tit or ""),""}
1009 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.

```

1010 function self.image(lab,src,tit)
1011     return {"\\markdownRendererImage{" ,lab,""}
1012             {"",self.string(src),""},
1013             {"",self.uri(src),""},
1014             {"",self.string(tit or ""),""}
1015 end

```

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

```

1016 local languages_json = (function()
1017     local kpse = require('kpse')
1018     kpse.set_program_name('luatex')
1019     local base, prev, curr
1020     for _, file in ipairs{kpse.lookup(options.contentBlocksLanguageMap,
1021                                     { all=true })} do
1022         json = assert(io.open(file, "r")):read("*all")
1023         :gsub('(["^\n]-"):','[%1]=')
1024         curr = (function()
1025             local _ENV={ json=json, load=load } -- run in sandbox
1026             return load("return "..json)()
1027         end)()
1028         if type(curr) == "table" then
1029             if base == nil then
1030                 base = curr
1031             else
1032                 setmetatable(prev, { __index = curr })
1033             end
1034             prev = curr
1035         end

```

```

1036 end
1037 return base or {}
1038 end)()

```

Define `writer->contentblock` as a function that will transform an input iA Writer content block to the output format, where `src` corresponds to the URI prefix, `suf` to the URI extension, `type` to the type of the content block (`localfile` or `onlineimage`), and `tit` to the title of the content block.

```

1039 function self.contentblock(src,suf,type,tit)
1040     src = src.." "..suf
1041     suf = suf:lower()
1042     if type == "onlineimage" then
1043         return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
1044                 "{" ,self.string(src),"} ",
1045                 "{" ,self.uri(src),"} ",
1046                 "{" ,self.string(tit or ""),"} "}
1047     elseif languages_json[suf] then
1048         return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
1049                 "{" ,self.string(languages_json[suf]),"} ",
1050                 "{" ,self.string(src),"} ",
1051                 "{" ,self.uri(src),"} ",
1052                 "{" ,self.string(tit or ""),"} "}
1053     else
1054         return {"\\markdownRendererContentBlock{" ,suf,"} ",
1055                 "{" ,self.string(src),"} ",
1056                 "{" ,self.uri(src),"} ",
1057                 "{" ,self.string(tit or ""),"} "}
1058     end
1059 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.

```

1060 local function ulitem(s)
1061     return {"\\markdownRendererUlItem " ,s,
1062             "\\markdownRendererUlItemEnd "}
1063 end
1064
1065 function self.bulletlist(items,tight)
1066     local buffer = {}
1067     for _,item in ipairs(items) do
1068         buffer[#buffer + 1] = ulitem(item)
1069     end
1070     local contents = util.intersperse(buffer,"\\n")
1071     if tight and options.tightLists then
1072         return {"\\markdownRendererUlBeginTight\\n",contents,
1073                 "\\n\\markdownRendererUlEndTight "}

```

```

1074     else
1075         return {"\\markdownRenderUlBegin\n",contents,
1076             "\n\\markdownRenderUlEnd "}
1077     end
1078 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.

```

1079 local function olitem(s,num)
1080     if num ~= nil then
1081         return {"\\markdownRenderOlItemWithNumber{" ,num,"} ",s,
1082             "\\markdownRenderOlItemEnd "}
1083     else
1084         return {"\\markdownRenderOlItem ",s,
1085             "\\markdownRenderOlItemEnd "}
1086     end
1087 end
1088
1089 function self.orderedlist(items,tight,startnum)
1090     local buffer = {}
1091     local num = startnum
1092     for _,item in ipairs(items) do
1093         buffer[#buffer + 1] = olitem(item,num)
1094         if num ~= nil then
1095             num = num + 1
1096         end
1097     end
1098     local contents = util.intersperse(buffer,"\n")
1099     if tight and options.tightLists then
1100         return {"\\markdownRenderOlBeginTight\n",contents,
1101             "\n\\markdownRenderOlEndTight "}
1102     else
1103         return {"\\markdownRenderOlBegin\n",contents,
1104             "\n\\markdownRenderOlEnd "}
1105     end
1106 end

```

Define `writer->inline_html` and `writer->display_html` as functions that will transform an inline or block HTML element respectively to the output format, where `html` is the HTML input.

```

1107 function self.inline_html(html) return "" end
1108 function self.display_html(html) return "" 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.

```

1109  local function dlitem(term, defs)
1110      local retVal = {"\\markdownRendererDlItem{",term,""}
1111      for _, def in ipairs(defs) do
1112          retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
1113                               "\\markdownRendererDlDefinitionEnd "}
1114      end
1115      retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
1116      return retVal
1117  end
1118
1119  function self.definitionlist(items,tight)
1120      local buffer = {}
1121      for _,item in ipairs(items) do
1122          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
1123      end
1124      if tight and options.tightLists then
1125          return {"\\markdownRendererDlBeginTight\\n", buffer,
1126                "\\n\\markdownRendererDlEndTight"}
1127      else
1128          return {"\\markdownRendererDlBegin\\n", buffer,
1129                "\\n\\markdownRendererDlEnd"}
1130      end
1131  end

```

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

```

1132  function self.emphasis(s)
1133      return {"\\markdownRendererEmphasis{",s,""}
1134  end

```

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

```

1135  function self.strong(s)
1136      return {"\\markdownRendererStrongEmphasis{",s,""}
1137  end

```

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

```

1138  function self.blockquote(s)
1139      return {"\\markdownRendererBlockQuoteBegin\\n",s,
1140            "\\n\\markdownRendererBlockQuoteEnd "}
1141  end

```

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

```

1142  function self.verbatim(s)

```

```

1143     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
1144     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
1145 end

```

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

```

1146 function self.fencedCode(i, s)
1147     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
1148     return {"\\markdownRendererInputFencedCode{" ,name,"}{" ,i,"}"}
1149 end

```

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

```

1150 function self.heading(s,level)
1151     local cmd
1152     if level == 1 then
1153         cmd = "\\markdownRendererHeadingOne"
1154     elseif level == 2 then
1155         cmd = "\\markdownRendererHeadingTwo"
1156     elseif level == 3 then
1157         cmd = "\\markdownRendererHeadingThree"
1158     elseif level == 4 then
1159         cmd = "\\markdownRendererHeadingFour"
1160     elseif level == 5 then
1161         cmd = "\\markdownRendererHeadingFive"
1162     elseif level == 6 then
1163         cmd = "\\markdownRendererHeadingSix"
1164     else
1165         cmd = ""
1166     end
1167     return {cmd,"{" ,s,"}"}
1168 end

```

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

```

1169 function self.note(s)
1170     return {"\\markdownRendererFootnote{" ,s,"}"}
1171 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.

```

1172 function self.citations(text_cites, cites)
1173   local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
1174     "{", #cites, "}"}
1175   for _,cite in ipairs(cites) do
1176     buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
1177       cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
1178   end
1179   return buffer
1180 end
1181
1182 return self
1183 end

```

3.1.4 Parsers

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

```

1184 local parsers = {}

```

3.1.4.1 Basic Parsers

```

1185 parsers.percent = P("%")
1186 parsers.at = P("@")
1187 parsers.comma = P(",")
1188 parsers.asterisk = P("*")
1189 parsers.dash = P("-")
1190 parsers.plus = P("+")
1191 parsers.underscore = P("_")
1192 parsers.period = P(".")
1193 parsers.hash = P("#")
1194 parsers.ampersand = P("&")
1195 parsers.backtick = P("`")
1196 parsers.less = P("<")
1197 parsers.more = P(">")
1198 parsers.space = P(" ")
1199 parsers.squote = P("'")
1200 parsers.dquote = P('"')
1201 parsers.lparent = P("(")
1202 parsers.rparent = P(")")
1203 parsers.lbracket = P("[")
1204 parsers.rbracket = P("]")
1205 parsers.circumflex = P("^")

```

```

1206 parsers.slash           = P("/")
1207 parsers.equal            = P("=")
1208 parsers.colon            = P(":")
1209 parsers.semicolon        = P(";")
1210 parsers.exclamation       = P("!")
1211 parsers.tilde            = P("~")
1212 parsers.tab              = P("\t")
1213 parsers.newline          = P("\n")
1214 parsers.tightblocksep    = P("\001")
1215
1216 parsers.digit            = R("09")
1217 parsers.hexdigit         = R("09", "af", "AF")
1218 parsers.letter           = R("AZ", "az")
1219 parsers.alphanumeric     = R("AZ", "az", "09")
1220 parsers.keyword          = parsers.letter
1221                          * parsers.alphanumeric^0
1222 parsers.internal_punctuation = S(".;,.$%&-+?<>~/")
1223
1224 parsers.doubleasterisks  = P("**")
1225 parsers.doubleunderscores = P("__")
1226 parsers.fourspaces       = P("    ")
1227
1228 parsers.any              = P(1)
1229 parsers.fail             = parsers.any - 1
1230
1231 parsers.escapable        = S("\\'*_{}[]() +_!<>#-~:~@;")
1232 parsers.anyescaped       = P("\\") / " " * parsers.escapable
1233 + parsers.any
1234
1235 parsers.spacechar        = S("\t ")
1236 parsers.spacing          = S(" \n\r\t")
1237 parsers.nonspacechar     = parsers.any - parsers.spacing
1238 parsers.optionalspace    = parsers.spacechar^0
1239
1240 parsers.specialchar      = S("*_ '&[]<!\. @-~")
1241
1242 parsers.normalchar       = parsers.any - (parsers.specialchar
1243                                           + parsers.spacing
1244                                           + parsers.tightblocksep)
1245 parsers.eof              = -parsers.any
1246 parsers.nonindentSPACE   = parsers.space^-3 * - parsers.spacechar
1247 parsers.indent           = parsers.space^-3 * parsers.tab
1248                          + parsers.fourspaces / " "
1249 parsers.linechar          = P(1 - parsers.newline)
1250
1251 parsers.blankline        = parsers.optionalspace
1252                          * parsers.newline / "\n"

```

```

1253 parsers.blanklines      = parsers.blankline^0
1254 parsers.skipblanklines   = (parsers.optionalspace * parsers.newline)^0
1255 parsers.indentedline     = parsers.indent /""
1256                          * C(parsers.linechar^1 * parsers.newline^-1)
1257 parsers.optionallyindentedline = parsers.indent^-1 /""
1258                          * C(parsers.linechar^1 * parsers.newline^-1)
1259 parsers.sp                = parsers.spacing^0
1260 parsers.spnl              = parsers.optionalspace
1261                          * (parsers.newline * parsers.optionalspace)^-1
1262 parsers.line              = parsers.linechar^0 * parsers.newline
1263                          + parsers.linechar^1 * parsers.eof
1264 parsers.nonemptyline      = parsers.line - parsers.blankline
1265
1266 parsers.chunk              = parsers.line * (parsers.optionallyindentedline
1267                          - parsers.blankline)^0
1268
1269 -- block followed by 0 or more optionally
1270 -- indented blocks with first line indented.
1271 parsers.indented_blocks = function(bl)
1272   return Cs( bl
1273             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
1274             * (parsers.blankline^1 + parsers.eof) )
1275 end

```

3.1.4.2 Parsers Used for Markdown Lists

```

1276 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
1277
1278 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
1279                  * (parsers.tab + parsers.space^-3)
1280                  + parsers.space * parsers.bulletchar * #parsers.spacing
1281                  * (parsers.tab + parsers.space^-2)
1282                  + parsers.space * parsers.space * parsers.bulletchar
1283                  * #parsers.spacing
1284                  * (parsers.tab + parsers.space^-1)
1285                  + parsers.space * parsers.space * parsers.space
1286                  * parsers.bulletchar * #parsers.spacing
1287                  )

```

3.1.4.3 Parsers Used for Markdown Code Spans

```

1288 parsers.openticks  = Cg(parsers.backtick^1, "ticks")
1289
1290 local function captures_equal_length(s,i,a,b)
1291   return #a == #b and i
1292 end
1293
1294 parsers.closeticks = parsers.space^-1

```



```

1295             * Cmt(C(parsers.backtick^1)
1296                 * Cb("ticks"), captures_equal_length)
1297
1298 parsers.intickschar = (parsers.any - S(" \n\r'"))
1299                     + (parsers.newline * -parsers.blankline)
1300                     + (parsers.space - parsers.closeticks)
1301                     + (parsers.backtick^1 - parsers.closeticks)
1302
1303 parsers.inticks      = parsers.openticks * parsers.space^-1
1304                     * C(parsers.intickschar^0) * parsers.closeticks

```

3.1.4.4 Parsers Used for Fenced Code Blocks

```

1305 local function captures_geq_length(s,i,a,b)
1306   return #a >= #b and i
1307 end
1308
1309 parsers.infostring    = (parsers.linechar - (parsers.backtick
1310     + parsers.space^1 * (parsers.newline + parsers.eof)))^0
1311
1312 local fenceindent
1313 parsers.fencehead     = function(char)
1314   return              C(parsers.nonindentospace) / function(s) fenceindent = #s end
1315                     * Cg(char^3, "fencelength")
1316                     * parsers.optionalspace * C(parsers.infostring)
1317                     * parsers.optionalspace * (parsers.newline + parsers.eof)
1318 end
1319
1320 parsers.fencetail     = function(char)
1321   return              parsers.nonindentospace
1322                     * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
1323                     * parsers.optionalspace * (parsers.newline + parsers.eof)
1324                     + parsers.eof
1325 end
1326
1327 parsers.fencedline    = function(char)
1328   return              C(parsers.line - parsers.fencetail(char))
1329                     / function(s)
1330                       return s:gsub("^" .. string.rep(" ?", fenceindent), "")
1331                     end
1332 end

```

3.1.4.5 Parsers Used for Markdown Tags and Links

```

1333 parsers.leader        = parsers.space^-3
1334
1335 -- content in balanced brackets, parentheses, or quotes:
1336 parsers.bracketed     = P{ parsers.lbracket

```

```

1337             * ((parsers.anyescaped - (parsers.lbracket
1338                                     + parsers.rbracket
1339                                     + parsers.blankline^2)
1340             ) + V(1))^0
1341             * parsers.rbracket }
1342
1343 parsers.inparens = P{ parsers.lparent
1344             * ((parsers.anyescaped - (parsers.lparent
1345                                     + parsers.rparent
1346                                     + parsers.blankline^2)
1347             ) + V(1))^0
1348             * parsers.rparent }
1349
1350 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
1351             * ((parsers.anyescaped - (parsers.squote
1352                                     + parsers.blankline^2)
1353             ) + V(1))^0
1354             * parsers.squote }
1355
1356 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
1357             * ((parsers.anyescaped - (parsers.dquote
1358                                     + parsers.blankline^2)
1359             ) + V(1))^0
1360             * parsers.dquote }
1361
1362 -- bracketed tag for markdown links, allowing nested brackets:
1363 parsers.tag      = parsers.lbracket
1364             * Cs((parsers.alphanumeric^1
1365                 + parsers.bracketed
1366                 + parsers.inticks
1367                 + (parsers.anyescaped
1368                     - (parsers.rbracket + parsers.blankline^2)))^0)
1369             * parsers.rbracket
1370
1371 -- url for markdown links, allowing nested brackets:
1372 parsers.url      = parsers.less * Cs((parsers.anyescaped
1373                                     - parsers.more)^0)
1374                                     * parsers.more
1375             + Cs((parsers.inparens + (parsers.anyescaped
1376                                     - parsers.spacing
1377                                     - parsers.rparent))^1)
1378
1379 -- quoted text, possibly with nested quotes:
1380 parsers.title_s  = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
1381                                     + parsers.squoted)^0)
1382                                     * parsers.squote
1383

```

```

1384 parsers.title_d      = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
1385                                     + parsers.dquoted)^0)
1386                                     * parsers.dquote
1387
1388 parsers.title_p        = parsers.lparent
1389                         * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
1390                         * parsers.rparent
1391
1392 parsers.title          = parsers.title_d + parsers.title_s + parsers.title_p
1393
1394 parsers.optionaltitle
1395                         = parsers.spnl * parsers.title * parsers.spacechar^0
1396                         + Cc("")

```

3.1.4.6 Parsers Used for iA Writer Content Blocks

```

1397 parsers.contentblock_tail
1398                         = parsers.optionaltitle
1399                         * (parsers.newline + parsers.eof)
1400
1401 -- case insensitive online image suffix:
1402 parsers.onlineimagesuffix
1403                         = (function(...)
1404                             local parser = nil
1405                             for _,suffix in ipairs({...}) do
1406                                 local pattern=nil
1407                                 for i=1,#suffix do
1408                                     local char=suffix:sub(i,i)
1409                                     char = S(char:lower()..char:upper())
1410                                     if pattern == nil then
1411                                         pattern = char
1412                                     else
1413                                         pattern = pattern * char
1414                                     end
1415                                 end
1416                                 if parser == nil then
1417                                     parser = pattern
1418                                 else
1419                                     parser = parser + pattern
1420                                 end
1421                             end
1422                             return parser
1423                         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
1424
1425 -- online image url for iA Writer content blocks with mandatory suffix,
1426 -- allowing nested brackets:
1427 parsers.onlineimageurl

```

```

1428 = (parsers.less
1429 * Cs((parsers.anyescaped
1430 - parsers.more
1431 - #(parsers.period
1432 * parsers.onlineimagesuffix
1433 * parsers.more
1434 * parsers.contentblock_tail))^0)
1435 * parsers.period
1436 * Cs(parsers.onlineimagesuffix)
1437 * parsers.more
1438 + (Cs((parsers.inparens
1439 + (parsers.anyescaped
1440 - parsers.spacing
1441 - parsers.rparent
1442 - #(parsers.period
1443 * parsers.onlineimagesuffix
1444 * parsers.contentblock_tail))))^0)
1445 * parsers.period
1446 * Cs(parsers.onlineimagesuffix))
1447 ) * Cc("onlineimage")
1448
1449 -- filename for iA Writer content blocks with mandatory suffix:
1450 parsers.localfilepath
1451 = parsers.slash
1452 * Cs((parsers.anyescaped
1453 - parsers.tab
1454 - parsers.newline
1455 - #(parsers.period
1456 * parsers.alphanumeric^1
1457 * parsers.contentblock_tail))^1)
1458 * parsers.period
1459 * Cs(parsers.alphanumeric^1)
1460 * Cc("localfile")

```

3.1.4.7 Parsers Used for Citations

```

1461 parsers.citation_name = Cs(parsers.dash^-1) * parsers.at
1462 * Cs(parsers.alphanumeric
1463 * (parsers.alphanumeric + parsers.internal_punctuation
1464 - parsers.comma - parsers.semicolon)^0)
1465
1466 parsers.citation_body_prenote
1467 = Cs((parsers.alphanumeric^1
1468 + parsers.bracketed
1469 + parsers.inticks
1470 + (parsers.anyescaped
1471 - (parsers.rbracket + parsers.blankline^2))

```

```

1472             - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
1473
1474 parsers.citation_body_postnote
1475     = Cs((parsers.alphanumeric^1
1476         + parsers.bracketed
1477         + parsers.inticks
1478         + (parsers.anyescaped
1479           - (parsers.rbracket + parsers.semicolon
1480             + parsers.blankline^2))
1481         - (parsers.spnl * parsers.rbracket))^0)
1482
1483 parsers.citation_body_chunk
1484     = parsers.citation_body_prenote
1485       * parsers.spnl * parsers.citation_name
1486       * (parsers.comma * parsers.spnl)^-1
1487       * parsers.citation_body_postnote
1488
1489 parsers.citation_body
1490     = parsers.citation_body_chunk
1491       * (parsers.semicolon * parsers.spnl
1492         * parsers.citation_body_chunk)^0
1493
1494 parsers.citation_headless_body_postnote
1495     = Cs((parsers.alphanumeric^1
1496         + parsers.bracketed
1497         + parsers.inticks
1498         + (parsers.anyescaped
1499           - (parsers.rbracket + parsers.at
1500             + parsers.semicolon + parsers.blankline^2))
1501         - (parsers.spnl * parsers.rbracket))^0)
1502
1503 parsers.citation_headless_body
1504     = parsers.citation_headless_body_postnote
1505       * (parsers.sp * parsers.semicolon * parsers.spnl
1506         * parsers.citation_body_chunk)^0

```

3.1.4.8 Parsers Used for Footnotes

```

1507 local function strip_first_char(s)
1508     return s:sub(2)
1509 end
1510
1511 parsers.RawNoteRef = #(parsers.lbracket * parsers.circumflex)
1512                      * parsers.tag / strip_first_char

```

3.1.4.9 Parsers Used for HTML

```

1513 -- case-insensitive match (we assume s is lowercase). must be single byte encoding

```

```

1514 parsers.keyword_exact = function(s)
1515   local parser = P(0)
1516   for i=1,#s do
1517     local c = s:sub(i,i)
1518     local m = c .. upper(c)
1519     parser = parser * S(m)
1520   end
1521   return parser
1522 end
1523
1524 parsers.block_keyword =
1525   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
1526   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
1527   parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
1528   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
1529   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
1530   parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
1531   parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
1532   parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
1533   parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
1534   parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
1535   parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
1536   parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
1537   parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
1538   parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
1539   parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
1540   parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
1541   parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
1542   parsers.keyword_exact("td") + parsers.keyword_exact("tr")
1543
1544 -- There is no reason to support bad html, so we expect quoted attributes
1545 parsers.htmlattributevalue
1546   = parsers.squote * (parsers.any - (parsers.blankline
1547   + parsers.squote))^0
1548   * parsers.squote
1549   + parsers.dquote * (parsers.any - (parsers.blankline
1550   + parsers.dquote))^0
1551   * parsers.dquote
1552
1553 parsers.htmlattribute = parsers.spacing^1
1554   * (parsers.alphanumeric + S("_-"))^1
1555   * parsers.sp * parsers.equal * parsers.sp
1556   * parsers.htmlattributevalue
1557
1558 parsers.htmlcomment = P("<!--") * (parsers.any - P("-->"))^0 * P("-->")
1559
1560 parsers.htmlinstruction = P("<?") * (parsers.any - P("?>" ))^0 * P("?>" )

```

```

1561
1562 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
1563         * parsers.sp * parsers.more
1564
1565 parsers.openelt_exact = function(s)
1566     return parsers.less * parsers.sp * parsers.keyword_exact(s)
1567         * parsers.htmlattribute^0 * parsers.sp * parsers.more
1568 end
1569
1570 parsers.openelt_block = parsers.sp * parsers.block_keyword
1571         * parsers.htmlattribute^0 * parsers.sp * parsers.more
1572
1573 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
1574         * parsers.keyword * parsers.sp * parsers.more
1575
1576 parsers.closeelt_exact = function(s)
1577     return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
1578         * parsers.sp * parsers.more
1579 end
1580
1581 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
1582         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
1583         * parsers.more
1584
1585 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
1586         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
1587         * parsers.more
1588
1589 parsers.displaytext = (parsers.any - parsers.less)^1
1590
1591 -- return content between two matched HTML tags
1592 parsers.in_matched = function(s)
1593     return { parsers.openelt_exact(s)
1594         * (V(1) + parsers.displaytext
1595         + (parsers.less - parsers.closeelt_exact(s)))^0
1596         * parsers.closeelt_exact(s) }
1597 end
1598
1599 local function parse_matched_tags(s,pos)
1600     local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
1601     return lpeg.match(parsers.in_matched(t),s,pos-1)
1602 end
1603
1604 parsers.in_matched_block_tags = parsers.less
1605         * Cmt(#parsers.openelt_block, parse_matched_tags)
1606
1607 parsers.displayhtml = parsers.htmlcomment

```

```

1608             + parsers.emptyelt_block
1609             + parsers.openelt_exact("hr")
1610             + parsers.in_matched_block_tags
1611             + parsers.htmlinstruction
1612
1613 parsers.inlinehtml = parsers.emptyelt_any
1614             + parsers.htmlcomment
1615             + parsers.htmlinstruction
1616             + parsers.openelt_any
1617             + parsers.closeelt_any

```

3.1.4.10 Parsers Used for HTML entities

```

1618 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
1619             * C(parsers.hexdigit^1) * parsers.semicolon
1620 parsers.decentity = parsers.ampersand * parsers.hash
1621             * C(parsers.digit^1) * parsers.semicolon
1622 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
1623             * parsers.semicolon

```

3.1.4.11 Helpers for References

```

1624 -- parse a reference definition: [foo]: /bar "title"
1625 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
1626             * parsers.spacechar^0 * parsers.url
1627             * parsers.optionaltitle * parsers.blankline^1

```

3.1.4.12 Inline Elements

```

1628 parsers.Inline      = V("Inline")
1629
1630 -- parse many p between starter and ender
1631 parsers.between = function(p, starter, ender)
1632   local ender2 = B(parsers.nonspacechar) * ender
1633   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
1634 end
1635
1636 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

3.1.4.13 Block Elements

```

1637 parsers.Block      = V("Block")
1638
1639 parsers.OnlineImageURL
1640             = parsers.leader
1641             * parsers.onlineimageurl
1642             * parsers.optionaltitle
1643

```



```

1644 parsers.LocalFilePath
1645         = parsers.leader
1646         * parsers.localfilepath
1647         * parsers.optionaltitle
1648
1649 parsers.TildeFencedCode
1650         = parsers.fencehead(parsers.tilde)
1651         * Cs(parsers.fencedline(parsers.tilde)^0)
1652         * parsers.fencetail(parsers.tilde)
1653
1654 parsers.BacktickFencedCode
1655         = parsers.fencehead(parsers.backtick)
1656         * Cs(parsers.fencedline(parsers.backtick)^0)
1657         * parsers.fencetail(parsers.backtick)
1658
1659 parsers.lineof = function(c)
1660     return (parsers.leader * (P(c) * parsers.optionalspace)^3
1661           * (parsers.newline * parsers.blankline^1
1662             + parsers.newline^-1 * parsers.eof))
1663 end

```

3.1.4.14 Lists

```

1664 parsers.defstartchar = S("~:")
1665 parsers.defstart      = ( parsers.defstartchar * #parsers.spacing
1666                          * (parsers.tab + parsers.space^-3)
1667                          + parsers.space * parsers.defstartchar * #parsers.spacing
1668                          * (parsers.tab + parsers.space^-2)
1669                          + parsers.space * parsers.space * parsers.defstartchar
1670                          * #parsers.spacing
1671                          * (parsers.tab + parsers.space^-1)
1672                          + parsers.space * parsers.space * parsers.space
1673                          * parsers.defstartchar * #parsers.spacing
1674                          )
1675
1676 parsers.dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

```

3.1.4.15 Headings

```

1677 -- parse Atx heading start and return level
1678 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
1679                     * -parsers.hash / length
1680
1681 -- parse setext header ending and return level
1682 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
1683
1684 local function strip_atx_end(s)
1685     return s:gsub("#%s*\n$", "")

```

```
1686 end
```

3.1.5 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 T_EX 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`.

```
1687 M.reader = {}
1688 function M.reader.new(writer, options)
1689     local self = {}
1690     options = options or {}
```

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

```
1691     setmetatable(options, { __index = function (_, key)
1692         return defaultOptions[key] end })
```

3.1.5.1 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.

```
1693     local function normalize_tag(tag)
1694         return unicode.utf8.lower(
1695             gsub(util.rope_to_string(tag), "[\n\r\t]+", " "))
1696     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.

```
1697     local expandtabs
1698     if options.preserveTabs then
1699         expandtabs = function(s) return s end
1700     else
1701         expandtabs = function(s)
1702             if s:find("\t") then
1703                 return s:gsub("[^\n]*", util.expand_tabs_in_line)
```

```

1704             else
1705                 return s
1706             end
1707         end
1708     end

```

The `larsers` (as in “local `parsers`”) hash table stores PEG patterns that depend on the received `options`, which impedes their reuse between different `reader` objects.

```

1709     local larsers = {}

```

3.1.5.2 Top-Level Parser Functions

```

1710     local function create_parser(name, grammar)
1711         return function(str)
1712             local res = lpeg.match(grammar(), str)
1713             if res == nil then
1714                 error(format("%s failed on:\n%s", name, str:sub(1,20)))
1715             else
1716                 return res
1717             end
1718         end
1719     end
1720
1721     local parse_blocks
1722     = create_parser("parse_blocks",
1723                     function()
1724                         return larsers.blocks
1725                     end)
1726
1727     local parse_blocks_toplevel
1728     = create_parser("parse_blocks_toplevel",
1729                     function()
1730                         return larsers.blocks_toplevel
1731                     end)
1732
1733     local parse_inlines
1734     = create_parser("parse_inlines",
1735                     function()
1736                         return larsers.inlines
1737                     end)
1738
1739     local parse_inlines_no_link
1740     = create_parser("parse_inlines_no_link",
1741                     function()
1742                         return larsers.inlines_no_link
1743                     end)
1744
1745     local parse_inlines_no_inline_note

```

```

1746     = create_parser("parse_inlines_no_inline_note",
1747                     function()
1748                         return larsers.inlines_no_inline_note
1749                     end)
1750
1751 local parse_inlines_nbsp
1752     = create_parser("parse_inlines_nbsp",
1753                     function()
1754                         return larsers.inlines_nbsp
1755                     end)

```

3.1.5.3 Parsers Used for Markdown Lists (local)

```

1756 if options.hashEnumerators then
1757     larsers.dig = parsers.digit + parsers.hash
1758 else
1759     larsers.dig = parsers.digit
1760 end
1761
1762 larsers.enumerator = C(larsers.dig^3 * parsers.period) * #parsers.spacing
1763                   + C(larsers.dig^2 * parsers.period) * #parsers.spacing
1764                   * (parsers.tab + parsers.space^1)
1765                   + C(larsers.dig * parsers.period) * #parsers.spacing
1766                   * (parsers.tab + parsers.space^2)
1767                   + parsers.space * C(larsers.dig^2 * parsers.period)
1768                   * #parsers.spacing
1769                   + parsers.space * C(larsers.dig * parsers.period)
1770                   * #parsers.spacing
1771                   * (parsers.tab + parsers.space^1)
1772                   + parsers.space * parsers.space * C(larsers.dig^1
1773                   * parsers.period) * #parsers.spacing

```

3.1.5.4 Parsers Used for Blockquotes (local)

```

1774 -- strip off leading > and indents, and run through blocks
1775 larsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-1)/""
1776                          * parsers.linechar^0 * parsers.newline)^1
1777                          * (-(parsers.leader * parsers.more
1778                             + parsers.blankline) * parsers.linechar^1
1779                          * parsers.newline)^0
1780
1781 if not options.breakableBlockquotes then
1782     larsers.blockquote_body = larsers.blockquote_body
1783                             * (parsers.blankline^0 / "")
1784 end

```

3.1.5.5 Parsers Used for Citations (local)

```

1785 larsers.citations = function(text_cites, raw_cites)
1786     local function normalize(str)
1787         if str == "" then
1788             str = nil
1789         else
1790             str = (options.citationNbsps and parse_inlines_nbsp or
1791                 parse_inlines)(str)
1792         end
1793         return str
1794     end
1795
1796     local cites = {}
1797     for i = 1,#raw_cites,4 do
1798         cites[#cites+1] = {
1799             prenote = normalize(raw_cites[i]),
1800             suppress_author = raw_cites[i+1] == "-",
1801             name = writer.string(raw_cites[i+2]),
1802             postnote = normalize(raw_cites[i+3]),
1803         }
1804     end
1805     return writer.citations(text_cites, cites)
1806 end

```

3.1.5.6 Parsers Used for Footnotes (local)

```

1807 local rawnotes = {}
1808
1809 -- like indirect_link
1810 local function lookup_note(ref)
1811     return function()
1812         local found = rawnotes[normalize_tag(ref)]
1813         if found then
1814             return writer.note(parse_blocks_toplevel(found))
1815         else
1816             return {"[", parse_inlines("^" .. ref), "]" }
1817         end
1818     end
1819 end
1820
1821 local function register_note(ref,rawnote)
1822     rawnotes[normalize_tag(ref)] = rawnote
1823     return ""
1824 end
1825
1826 larsers.NoteRef      = parsers.RawNoteRef / lookup_note
1827
1828

```

```

1829 larsers.NoteBlock = parsers.leader * parsers.RawNoteRef * parsers.colon
1830                   * parsers.spnl * parsers.indented_blocks(parsers.chunk)
1831                   / register_note
1832
1833 larsers.InlineNote = parsers.circumflex
1834                   * (parsers.tag / parse_inlines_no_inline_note) -- no notes inside
1835                   / writer.note

```

3.1.5.7 Helpers for Links and References (local)

```

1836 -- List of references defined in the document
1837 local references
1838
1839 -- add a reference to the list
1840 local function register_link(tag,url,title)
1841     references[normalize_tag(tag)] = { url = url, title = title }
1842     return ""
1843 end
1844
1845 -- lookup link reference and return either
1846 -- the link or nil and fallback text.
1847 local function lookup_reference(label,sps,tag)
1848     local tagpart
1849     if not tag then
1850         tag = label
1851         tagpart = ""
1852     elseif tag == "" then
1853         tag = label
1854         tagpart = "[]"
1855     else
1856         tagpart = {"[", parse_inlines(tag), "]" }
1857     end
1858     if sps then
1859         tagpart = {sps, tagpart}
1860     end
1861     local r = references[normalize_tag(tag)]
1862     if r then
1863         return r
1864     else
1865         return nil, {"[", parse_inlines(label), "]", tagpart}
1866     end
1867 end
1868
1869 -- lookup link reference and return a link, if the reference is found,
1870 -- or a bracketed label otherwise.
1871 local function indirect_link(label,sps,tag)
1872     return function()

```

```

1873     local r,fallback = lookup_reference(label,sps,tag)
1874     if r then
1875         return writer.link(parse_inlines_no_link(label), r.url, r.title)
1876     else
1877         return fallback
1878     end
1879 end
1880 end
1881
1882 -- lookup image reference and return an image, if the reference is found,
1883 -- or a bracketed label otherwise.
1884 local function indirect_image(label,sps,tag)
1885     return function()
1886         local r,fallback = lookup_reference(label,sps,tag)
1887         if r then
1888             return writer.image(writer.string(label), r.url, r.title)
1889         else
1890             return {"!", fallback}
1891         end
1892     end
1893 end

```

3.1.5.8 Inline Elements (local)

```

1894 larsers.Str      = parsers.normalchar^1 / writer.string
1895
1896 larsers.Symbol   = (parsers.specialchar - parsers.tightblocksep)
1897                  / writer.string
1898
1899 larsers.Ellipsis = P("...") / writer.ellipsis
1900
1901 larsers.Smart    = larsers.Ellipsis
1902
1903 larsers.Code     = parsers.inticks / writer.code
1904
1905 if options.blankBeforeBlockquote then
1906     larsers.bqstart = parsers.fail
1907 else
1908     larsers.bqstart = parsers.more
1909 end
1910
1911 if options.blankBeforeHeading then
1912     larsers.headerstart = parsers.fail
1913 else
1914     larsers.headerstart = parsers.hash
1915                        + (parsers.line * (parsers.equal^1 + parsers.dash^1)
1916                        * parsers.optionalspace * parsers.newline)

```

```

1917 end
1918
1919 if not options.fencedCode or options.blankBeforeCodeFence then
1920     larsers.fencestart = parsers.fail
1921 else
1922     larsers.fencestart = parsers.fencehead(parsers.backtick)
1923                       + parsers.fencehead(parsers.tilde)
1924 end
1925
1926 larsers.Endline    = parsers.newline * -( -- newline, but not before...
1927                       parsers.blankline -- paragraph break
1928                       + parsers.tightblocksep -- nested list
1929                       + parsers.eof         -- end of document
1930                       + larsers.bqstart
1931                       + larsers.headerstart
1932                       + larsers.fencestart
1933                       ) * parsers.spacechar^0 / writer.space
1934
1935 larsers.Space      = parsers.spacechar^2 * larsers.Endline / writer.linebreak
1936                       + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
1937                       + parsers.spacechar^1 * larsers.Endline^-1
1938                               * parsers.optionalspace / writer.space
1939
1940 larsers.NonbreakingEndline
1941     = parsers.newline * -( -- newline, but not before...
1942                       parsers.blankline -- paragraph break
1943                       + parsers.tightblocksep -- nested list
1944                       + parsers.eof         -- end of document
1945                       + larsers.bqstart
1946                       + larsers.headerstart
1947                       + larsers.fencestart
1948                       ) * parsers.spacechar^0 / writer.nbsp
1949
1950 larsers.NonbreakingSpace
1951     = parsers.spacechar^2 * larsers.Endline / writer.linebreak
1952     + parsers.spacechar^1 * larsers.Endline^-1 * parsers.eof / ""
1953     + parsers.spacechar^1 * larsers.Endline^-1
1954       * parsers.optionalspace / writer.nbsp
1955
1956 if options.underscores then
1957     larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
1958                                     parsers.doubleasterisks)
1959                     + parsers.between(parsers.Inline, parsers.doubleunderscores,
1960                                     parsers.doubleunderscores)
1961                     ) / writer.strong
1962
1963     larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,

```



```

1964                                     parsers.asterisk)
1965         + parsers.between(parsers.Inline, parsers.underscore,
1966                             parsers.underscore)
1967         ) / writer.emphasis
1968     else
1969         larsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
1970                                             parsers.doubleasterisks)
1971         ) / writer.strong
1972
1973         larsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
1974                                             parsers.asterisk)
1975         ) / writer.emphasis
1976     end
1977
1978     larsers.AutoLinkUrl    = parsers.less
1979                             * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
1980                             * parsers.more
1981                             / function(url)
1982                                 return writer.link(writer.string(url), url)
1983                             end
1984
1985     larsers.AutoLinkEmail = parsers.less
1986                             * C((parsers.alphanumeric + S("-._+"))^1
1987                             * P("@") * parsers.urlchar^1)
1988                             * parsers.more
1989                             / function(email)
1990                                 return writer.link(writer.string(email),
1991                                                         "mailto: "..email)
1992                             end
1993
1994     larsers.DirectLink     = (parsers.tag / parse_inlines_no_link) -- no links inside link
1995                             * parsers.spnl
1996                             * parsers.lparent
1997                             * (parsers.url + Cc("")) -- link can be empty [foo]()
1998                             * parsers.optionaltitle
1999                             * parsers.rparent
2000                             / writer.link
2001
2002     larsers.IndirectLink   = parsers.tag * (C(parsers.spnl) * parsers.tag)^-1
2003                             / indirect_link
2004
2005     -- parse a link or image (direct or indirect)
2006     larsers.Link           = larsers.DirectLink + larsers.IndirectLink
2007
2008     larsers.DirectImage    = parsers.exclamation
2009                             * (parsers.tag / parse_inlines)
2010                             * parsers.spnl

```

```

2011             * parsers.lparent
2012             * (parsers.url + Cc("")) -- link can be empty [foo]()
2013             * parsers.optionaltitle
2014             * parsers.rparent
2015             / writer.image
2016
2017 larsers.IndirectImage = parsers.exclamation * parsers.tag
2018                       * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
2019
2020 larsers.Image         = larsers.DirectImage + larsers.IndirectImage
2021
2022 larsers.TextCitations = Ct(Cc(""))
2023                       * parsers.citation_name
2024                       * ((parsers.spnl
2025                           * parsers.lbracket
2026                           * parsers.citation_headless_body
2027                           * parsers.rbracket) + Cc("")))
2028                       / function(raw_cites)
2029                           return larsers.citations(true, raw_cites)
2030                       end
2031
2032 larsers.ParenthesizedCitations
2033                       = Ct(parsers.lbracket
2034                           * parsers.citation_body
2035                           * parsers.rbracket)
2036                       / function(raw_cites)
2037                           return larsers.citations(false, raw_cites)
2038                       end
2039
2040 larsers.Citations     = larsers.TextCitations + larsers.ParenthesizedCitations
2041
2042 -- avoid parsing long strings of * or _ as emph/strong
2043 larsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
2044                       / writer.string
2045
2046 larsers.EscapedChar   = S("\\") * C(parsers.escapable) / writer.string
2047
2048 larsers.InlineHtml    = C(parsers.inlinehtml) / writer.inline_html
2049
2050 larsers.HtmlEntity    = parsers.hexentity / entities.hex_entity / writer.string
2051                       + parsers.decentity / entities.dec_entity / writer.string
2052                       + parsers.tagentity / entities.char_entity / writer.string

```

3.1.5.9 Block Elements (local)

```

2053 larsers.ContentBlock = parsers.leader
2054                       * (parsers.localfilepath + parsers.onlineimageurl)

```

```

2055             * parsers.contentblock_tail
2056             / writer.contentblock
2057
2058     larsers.DisplayHtml = C(parsers.displayhtml)
2059             / expandtabs / writer.display_html
2060
2061     larsers.Verbatim    = Cs( (parsers.blanklines
2062             * ((parsers.indentedline - parsers.blankline))^1)^1
2063             ) / expandtabs / writer.verbatim
2064
2065     larsers.FencedCode  = (parsers.TildeFencedCode
2066             + parsers.BacktickFencedCode)
2067             / function(infostring, code)
2068             return writer.fencedCode(writer.string(infostring),
2069                                     expandtabs(code))
2070             end
2071
2072     larsers.Blockquote   = Cs(larsers.blockquote_body^1)
2073             / parse_blocks_toplevel / writer.blockquote
2074
2075     larsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
2076             + parsers.lineof(parsers.dash)
2077             + parsers.lineof(parsers.underscore)
2078             ) / writer.hrule
2079
2080     larsers.Reference    = parsers.define_reference_parser / register_link
2081
2082     larsers.Paragraph    = parsers.nonindentSPACE * Ct(parsers.Inline^1)
2083             * parsers.newline
2084             * ( parsers.blankline^1
2085             + #parsers.hash
2086             + #(parsers.leader * parsers.more * parsers.space^-1)
2087             )
2088             / writer.paragraph
2089
2090     larsers.ToplevelParagraph
2091             = parsers.nonindentSPACE * Ct(parsers.Inline^1)
2092             * ( parsers.newline
2093             * ( parsers.blankline^1
2094             + #parsers.hash
2095             + #(parsers.leader * parsers.more * parsers.space^-1)
2096             + parsers.eof
2097             )
2098             + parsers.eof )
2099             / writer.paragraph
2100
2101     larsers.Plain        = parsers.nonindentSPACE * Ct(parsers.Inline^1)

```

```
2102                                     / writer.plain
```

3.1.5.10 Lists (local)

```
2103 larsers.starter = parsers.bullet + larsers.enumerator
2104
2105 -- we use \001 as a separator between a tight list item and a
2106 -- nested list under it.
2107 larsers.NestedList          = Cs((parsers.optionallyindentedline
2108                                - larsers.starter)^1)
2109                                / function(a) return "\001"..a end
2110
2111 larsers.ListBlockLine       = parsers.optionallyindentedline
2112                                - parsers.blankline - (parsers.indent^-1
2113                                                         * larsers.starter)
2114
2115 larsers.ListBlock           = parsers.line * larsers.ListBlockLine^0
2116
2117 larsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
2118                                * larsers.ListBlock
2119
2120 larsers.TightListItem = function(starter)
2121     return -larsers.HorizontalRule
2122            * (Cs(starter / "" * larsers.ListBlock * larsers.NestedList^-1)
2123               / parse_blocks)
2124            * -(parsers.blanklines * parsers.indent)
2125 end
2126
2127 larsers.LooseListItem = function(starter)
2128     return -larsers.HorizontalRule
2129            * Cs( starter / "" * larsers.ListBlock * Cc("\n")
2130                 * (larsers.NestedList + larsers.ListContinuationBlock^0)
2131                 * (parsers.blanklines / "\n\n")
2132                 ) / parse_blocks
2133 end
2134
2135 larsers.BulletList = ( Ct(larsers.TightListItem(parsers.bullet)^1) * Cc(true)
2136                        * parsers.skipblanklines * -parsers.bullet
2137                        + Ct(larsers.LooseListItem(parsers.bullet)^1) * Cc(false)
2138                        * parsers.skipblanklines )
2139                        / writer.bulletlist
2140
2141 local function ordered_list(items,tight,startNumber)
2142     if options.startNumber then
2143         startNumber = tonumber(startNumber) or 1 -- fallback for '#'
2144     else
2145         startNumber = nil
```

```

2146     end
2147     return writer.orderedlist(items,tight,startNumber)
2148 end
2149
2150 larsers.OrderedList = Cg(larsers.enumerator, "listtype") *
2151     ( Ct(larsers.TightListItem(Cb("listtype")))
2152       * larsers.TightListItem(larsers.enumerator)^0)
2153     * Cc(true) * parsers.skipblanklines * -larsers.enumerator
2154     + Ct(larsers.LooseListItem(Cb("listtype")))
2155       * larsers.LooseListItem(larsers.enumerator)^0)
2156     * Cc(false) * parsers.skipblanklines
2157     ) * Cb("listtype") / ordered_list
2158
2159 local function definition_list_item(term, defs, tight)
2160     return { term = parse_inlines(term), definitions = defs }
2161 end
2162
2163 larsers.DefinitionListItemLoose = C(parsers.line) * parsers.skipblanklines
2164     * Ct((parsers.defstart
2165         * parsers.indented_blocks(parsers.dlchunk)
2166         / parse_blocks_toplevel)^1)
2167     * Cc(false) / definition_list_item
2168
2169 larsers.DefinitionListItemTight = C(parsers.line)
2170     * Ct((parsers.defstart * parsers.dlchunk
2171         / parse_blocks)^1)
2172     * Cc(true) / definition_list_item
2173
2174 larsers.DefinitionList = ( Ct(larsers.DefinitionListItemLoose^1) * Cc(false)
2175     + Ct(larsers.DefinitionListItemTight^1)
2176     * (parsers.skipblanklines
2177       * -larsers.DefinitionListItemLoose * Cc(true))
2178     ) / writer.definitionlist

```

3.1.5.11 Blank (local)

```

2179 larsers.Blank      = parsers.blankline / ""
2180                   + larsers.NoteBlock
2181                   + larsers.Reference
2182                   + (parsers.tightblocksep / "\n")

```

3.1.5.12 Headings (local)

```

2183 -- parse atx header
2184 larsers.AtxHeading = Cg(parsers.HeadingStart,"level")
2185     * parsers.optionalspace
2186     * (C(parsers.line) / strip_atx_end / parse_inlines)
2187     * Cb("level")

```

```

2188             / writer.heading
2189
2190 -- parse setext header
2191 larsers.SettextHeading = #(parsers.line * S("--"))
2192                       * Ct(parsers.line / parse_inlines)
2193                       * parsers.HeadingLevel
2194                       * parsers.optionalspace * parsers.newline
2195                       / writer.heading
2196
2197 larsers.Heading = larsers.AtXHeading + larsers.SettextHeading

```

3.1.5.13 Syntax Specification

```

2198 local syntax =
2199   { "Blocks",
2200
2201     Blocks          = larsers.Blank^0 * parsers.Block^-1
2202                     * (larsers.Blank^0 / function()
2203                       return writer.interblocksep
2204                       end
2205                       * parsers.Block)^0
2206                     * larsers.Blank^0 * parsers.eof,
2207
2208     Blank           = larsers.Blank,
2209
2210     Block           = V("ContentBlock")
2211                     + V("Blockquote")
2212                     + V("Verbatim")
2213                     + V("FencedCode")
2214                     + V("HorizontalRule")
2215                     + V("BulletList")
2216                     + V("OrderedList")
2217                     + V("Heading")
2218                     + V("DefinitionList")
2219                     + V("DisplayHtml")
2220                     + V("Paragraph")
2221                     + V("Plain"),
2222
2223     ContentBlock    = larsers.ContentBlock,
2224     Blockquote      = larsers.Blockquote,
2225     Verbatim        = larsers.Verbatim,
2226     FencedCode      = larsers.FencedCode,
2227     HorizontalRule  = larsers.HorizontalRule,
2228     BulletList      = larsers.BulletList,
2229     OrderedList     = larsers.OrderedList,
2230     Heading         = larsers.Heading,
2231     DefinitionList  = larsers.DefinitionList,

```

```

2232     DisplayHtml      = larsers.DisplayHtml,
2233     Paragraph        = larsers.Paragraph,
2234     Plain            = larsers.Plain,
2235
2236     Inline           = V("Str")
2237                     + V("Space")
2238                     + V("Endline")
2239                     + V("U1OrStarLine")
2240                     + V("Strong")
2241                     + V("Emph")
2242                     + V("InlineNote")
2243                     + V("NoteRef")
2244                     + V("Citations")
2245                     + V("Link")
2246                     + V("Image")
2247                     + V("Code")
2248                     + V("AutoLinkUrl")
2249                     + V("AutoLinkEmail")
2250                     + V("InlineHtml")
2251                     + V("HtmlEntity")
2252                     + V("EscapedChar")
2253                     + V("Smart")
2254                     + V("Symbol"),
2255
2256     Str              = larsers.Str,
2257     Space            = larsers.Space,
2258     Endline          = larsers.Endline,
2259     U1OrStarLine     = larsers.U1OrStarLine,
2260     Strong           = larsers.Strong,
2261     Emph             = larsers.Emph,
2262     InlineNote       = larsers.InlineNote,
2263     NoteRef          = larsers.NoteRef,
2264     Citations        = larsers.Citations,
2265     Link             = larsers.Link,
2266     Image            = larsers.Image,
2267     Code             = larsers.Code,
2268     AutoLinkUrl      = larsers.AutoLinkUrl,
2269     AutoLinkEmail    = larsers.AutoLinkEmail,
2270     InlineHtml       = larsers.InlineHtml,
2271     HtmlEntity       = larsers.HtmlEntity,
2272     EscapedChar      = larsers.EscapedChar,
2273     Smart            = larsers.Smart,
2274     Symbol           = larsers.Symbol,
2275 }
2276
2277 if not options.citations then
2278     syntax.Citations = parsers.fail

```

```

2279 end
2280
2281 if not options.contentBlocks then
2282     syntax.ContentBlock = parsers.fail
2283 end
2284
2285 if not options.codeSpans then
2286     syntax.Code = parsers.fail
2287 end
2288
2289 if not options.definitionLists then
2290     syntax.DefinitionList = parsers.fail
2291 end
2292
2293 if not options.fencedCode then
2294     syntax.FencedCode = parsers.fail
2295 end
2296
2297 if not options.footnotes then
2298     syntax.NoteRef = parsers.fail
2299 end
2300
2301 if not options.html then
2302     syntax.DisplayHtml = parsers.fail
2303     syntax.InlineHtml = parsers.fail
2304     syntax.HtmlEntity = parsers.fail
2305 end
2306
2307 if not options.inlineFootnotes then
2308     syntax.InlineNote = parsers.fail
2309 end
2310
2311 if not options.smartEllipses then
2312     syntax.Smart = parsers.fail
2313 end
2314
2315 local blocks_toplevel_t = util.table_copy(syntax)
2316 blocks_toplevel_t.Paragraph = larsers.ToplevelParagraph
2317 larsers.blocks_toplevel = Ct(blocks_toplevel_t)
2318
2319 larsers.blocks = Ct(syntax)
2320
2321 local inlines_t = util.table_copy(syntax)
2322 inlines_t[1] = "Inlines"
2323 inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
2324 larsers.inlines = Ct(inlines_t)
2325

```



```

2326 local inlines_no_link_t = util.table_copy(inlines_t)
2327 inlines_no_link_t.Link = parsers.fail
2328 larsers.inlines_no_link = Ct(inlines_no_link_t)
2329
2330 local inlines_no_inline_note_t = util.table_copy(inlines_t)
2331 inlines_no_inline_note_t.InlineNote = parsers.fail
2332 larsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
2333
2334 local inlines_nbsp_t = util.table_copy(inlines_t)
2335 inlines_nbsp_t.Endline = larsers.NonbreakingEndline
2336 inlines_nbsp_t.Space = larsers.NonbreakingSpace
2337 larsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

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

```

2338 function self.convert(input)
2339     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.

```

2340     local opt_string = {}
2341     for k,_ in pairs(defaultOptions) do
2342         local v = options[k]
2343         if k ~= "cacheDir" then
2344             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
2345         end
2346     end
2347     table.sort(opt_string)
2348     local salt = table.concat(opt_string, ",") .. "," .. metadata.version

```

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

```

2349     local name = util.cache(options.cacheDir, input, salt, function(input)
2350         return util.ropetostring(parse_blocks_toplevel(input)) .. writer.eof
2351     end, ".md" .. writer.suffix)
2352     return writer.pack(name)
2353 end
2354 return self
2355 end

```

3.1.6 Conversion from Markdown to Plain T_EX

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`.

```
2356 function M.new(options)
2357   local writer = M.writer.new(options)
2358   local reader = M.reader.new(writer, options)
2359   return reader.convert
2360 end
2361
2362 return M
```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX 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 T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```
2363 \def\markdownInfo#1{%
2364   \message{(1.\the\inputlineno) markdown.tex info: #1.}}%
2365 \def\markdownWarning#1{%
2366   \message{(1.\the\inputlineno) markdown.tex warning: #1}}%
2367 \def\markdownError#1#2{%
2368   \errhelp{#2.}}%
2369   \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
2370 \def\markdownRendererInterblockSeparatorPrototype{\par}%
2371 \def\markdownRendererLineBreakPrototype{\hfil\break}%
2372 \let\markdownRendererEllipsisPrototype\dots
2373 \def\markdownRendererNbspPrototype{~}%
2374 \def\markdownRendererLeftBracePrototype{\char'{}%
2375 \def\markdownRendererRightBracePrototype{\char'}%
2376 \def\markdownRendererDollarSignPrototype{\char'$}%
2377 \def\markdownRendererPercentSignPrototype{\char'\}%
2378 \def\markdownRendererAmpersandPrototype{\char'&}%
2379 \def\markdownRendererUnderscorePrototype{\char'_}%
2380 \def\markdownRendererHashPrototype{\char'#}%
2381 \def\markdownRendererCircumflexPrototype{\char'^}%
2382 \def\markdownRendererBackslashPrototype{\char'\}%
```

```

2383 \def\markdownRendererTildePrototype{\char'~}%
2384 \def\markdownRendererPipePrototype{|}%
2385 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
2386 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
2387 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
2388   \markdownInput{#3}}%
2389 \def\markdownRendererContentBlockOnlineImagePrototype{%
2390   \markdownRendererImage}%
2391 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
2392   \markdownRendererInputFencedCode{#3}{#2}}%
2393 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
2394 \def\markdownRendererUlBeginPrototype{}%
2395 \def\markdownRendererUlBeginTightPrototype{}%
2396 \def\markdownRendererUlItemPrototype{}%
2397 \def\markdownRendererUlItemEndPrototype{}%
2398 \def\markdownRendererUlEndPrototype{}%
2399 \def\markdownRendererUlEndTightPrototype{}%
2400 \def\markdownRendererOlBeginPrototype{}%
2401 \def\markdownRendererOlBeginTightPrototype{}%
2402 \def\markdownRendererOlItemPrototype{}%
2403 \def\markdownRendererOlItemWithNumberPrototype#1{}%
2404 \def\markdownRendererOlItemEndPrototype{}%
2405 \def\markdownRendererOlEndPrototype{}%
2406 \def\markdownRendererOlEndTightPrototype{}%
2407 \def\markdownRendererDlBeginPrototype{}%
2408 \def\markdownRendererDlBeginTightPrototype{}%
2409 \def\markdownRendererDlItemPrototype#1{#1}%
2410 \def\markdownRendererDlItemEndPrototype{}%
2411 \def\markdownRendererDlDefinitionBeginPrototype{}%
2412 \def\markdownRendererDlDefinitionEndPrototype{\par}%
2413 \def\markdownRendererDlEndPrototype{}%
2414 \def\markdownRendererDlEndTightPrototype{}%
2415 \def\markdownRendererEmphasisPrototype#1{\it#1}%
2416 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
2417 \def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
2418 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
2419 \def\markdownRendererInputVerbatimPrototype#1{%
2420   \par{\tt\input"#1"\relax}\par}%
2421 \def\markdownRendererInputFencedCodePrototype#1#2{%
2422   \markdownRendererInputVerbatimPrototype{#1}}%
2423 \def\markdownRendererHeadingOnePrototype#1{#1}%
2424 \def\markdownRendererHeadingTwoPrototype#1{#1}%
2425 \def\markdownRendererHeadingThreePrototype#1{#1}%
2426 \def\markdownRendererHeadingFourPrototype#1{#1}%
2427 \def\markdownRendererHeadingFivePrototype#1{#1}%
2428 \def\markdownRendererHeadingSixPrototype#1{#1}%
2429 \def\markdownRendererHorizontalRulePrototype{}%

```

```

2430 \def\markdownRendererFootnotePrototype#1{#1}%
2431 \def\markdownRendererCitePrototype#1{}%
2432 \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).

```

2433 \def\markdownLuaOptions{%
2434 \ifx\markdownOptionBlankBeforeBlockquote\undefined\else
2435   blankBeforeBlockquote = \markdownOptionBlankBeforeBlockquote,
2436 \fi
2437 \ifx\markdownOptionBlankBeforeCodeFence\undefined\else
2438   blankBeforeCodeFence = \markdownOptionBlankBeforeCodeFence,
2439 \fi
2440 \ifx\markdownOptionBlankBeforeHeading\undefined\else
2441   blankBeforeHeading = \markdownOptionBlankBeforeHeading,
2442 \fi
2443 \ifx\markdownOptionBreakableBlockquotes\undefined\else
2444   breakableBlockquotes = \markdownOptionBreakableBlockquotes,
2445 \fi
2446 \ifx\markdownOptionCacheDir\undefined\else
2447   cacheDir = "\markdownOptionCacheDir",
2448 \fi
2449 \ifx\markdownOptionCitations\undefined\else
2450   citations = \markdownOptionCitations,
2451 \fi
2452 \ifx\markdownOptionCitationNbsps\undefined\else
2453   citationNbsps = \markdownOptionCitationNbsps,
2454 \fi
2455 \ifx\markdownOptionCodeSpans\undefined\else
2456   codeSpans = \markdownOptionCodeSpans,
2457 \fi
2458 \ifx\markdownOptionContentBlocks\undefined\else
2459   contentBlocks = \markdownOptionContentBlocks,
2460 \fi
2461 \ifx\markdownOptionContentBlocksLanguageMap\undefined\else
2462   contentBlocksLanguageMap =
2463     "\markdownOptionContentBlocksLanguageMap",
2464 \fi
2465 \ifx\markdownOptionDefinitionLists\undefined\else
2466   definitionLists = \markdownOptionDefinitionLists,
2467 \fi
2468 \ifx\markdownOptionFootnotes\undefined\else
2469   footnotes = \markdownOptionFootnotes,
2470 \fi

```

```

2471 \ifx\markdownOptionFencedCode\undefined\else
2472   fencedCode = \markdownOptionFencedCode,
2473 \fi
2474 \ifx\markdownOptionHashEnumerators\undefined\else
2475   hashEnumerators = \markdownOptionHashEnumerators,
2476 \fi
2477 \ifx\markdownOptionHtml\undefined\else
2478   html = \markdownOptionHtml,
2479 \fi
2480 \ifx\markdownOptionHybrid\undefined\else
2481   hybrid = \markdownOptionHybrid,
2482 \fi
2483 \ifx\markdownOptionInlineFootnotes\undefined\else
2484   inlineFootnotes = \markdownOptionInlineFootnotes,
2485 \fi
2486 \ifx\markdownOptionPreserveTabs\undefined\else
2487   preserveTabs = \markdownOptionPreserveTabs,
2488 \fi
2489 \ifx\markdownOptionSmartEllipses\undefined\else
2490   smartEllipses = \markdownOptionSmartEllipses,
2491 \fi
2492 \ifx\markdownOptionStartNumber\undefined\else
2493   startNumber = \markdownOptionStartNumber,
2494 \fi
2495 \ifx\markdownOptionTightLists\undefined\else
2496   tightLists = \markdownOptionTightLists,
2497 \fi
2498 \ifx\markdownOptionUnderscores\undefined\else
2499   underscores = \markdownOptionUnderscores,
2500 \fi}
2501 }%

```

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.

```

2502 \def\markdownPrepare{%

```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```

2503 local lfs = require("lfs")
2504 local cacheDir = "\markdownOptionCacheDir"
2505 if lfs.isdir(cacheDir) == true then else
2506   assert(lfs.mkdir(cacheDir))
2507 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.

```

2508 local md = require("markdown")
2509 local convert = md.new(\markdownLuaOptions)

```

2510 }%

3.2.4 Buffering Markdown Input

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` when the Lua shell escape bridge is in use, and to store the markdown input in the file named `\markdownOptionInputTempFileName` during the expansion of the macro `\markdownReadAndConvert`.

2511 \csname newwrite\endcsname\markdownLuaExecuteFileStream

The `\markdownReadAndConvertTab` macro contains the tab character literal.

2512 \begingroup

2513 \catcode'\^^I=12%

2514 \gdef\markdownReadAndConvertTab{^^I}%

2515 \endgroup

The `\markdownReadAndConvert` macro is largely a rewrite of the \TeX `\filecontents` macro to plain \TeX .

2516 \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.

2517 \catcode'\^^M=13%

2518 \catcode'\^^I=13%

2519 \catcode'|=0%

2520 \catcode'\=12%

2521 |gdef|markdownReadAndConvert#1#2{%

2522 |begingroup%

Open the `\markdownOptionInputTempFileName` file for writing.

2523 |immediate|openout|markdownLuaExecuteFileStream%

2524 |markdownOptionInputTempFileName%

2525 |markdownInfo{Buffering markdown input into the temporary %

2526 input file "|markdownOptionInputTempFileName" and scanning %

2527 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.

2528 |def|do##1{|catcode'##1=12}|dospecials%

2529 |catcode'|=12%

2530 |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.

```
2531 |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.
2532 |ifx|relax##3|relax%
2533 |immediate|write|markdownLuaExecuteFileStream{##1}%
2534 |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.

```
2535 |def^^M{%
2536 |markdownInfo{The ending token sequence was found}%
2537 |immediate|closeout|markdownLuaExecuteFileStream%
2538 |endgroup%
2539 |markdownInput|markdownOptionInputTempFileName%
2540 #2}%
2541 |fi%
```

Repeat with the next line.

```
2542 ^^M}%
```

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

```
2543 |catcode'|^^I=13%
2544 |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.

```
2545 |catcode'|^^M=13%
2546 |def^^M##1^^M{%
2547 |def^^M####1^^M{%
2548 |markdownReadAndConvertProcessLine####1#1#1|relax}%
2549 ^^M}%
2550 ^^M}%
```

Reset the character categories back to the former state.

```
2551 |endgroup
```

3.2.5 Lua Shell Escape Bridge

The following T_EX code is intended for T_EX 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` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

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

```
2552 \ifnum\markdownMode<2\relax
2553 \ifnum\markdownMode=0\relax
2554   \markdownInfo{Using mode 0: Shell escape via write18}%
2555 \else
2556   \markdownInfo{Using mode 1: Shell escape via os.execute}%
2557 \fi
```

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` (LuaT_EX, PdfT_EX) or the `\shellescape` (X_TT_EX) 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.

```
2558 \ifx\pdfshellescape\undefined
2559   \ifx\shellescape\undefined
2560     \ifnum\markdownMode=0\relax
2561       \def\markdownExecuteShellEscape{1}%
2562     \else
2563       \def\markdownExecuteShellEscape{%
2564         \directlua{tex.sprint(status.shell_escape or "1")}}%
2565     \fi
2566   \else
2567     \let\markdownExecuteShellEscape\shellescape
2568   \fi
2569 \else
2570   \let\markdownExecuteShellEscape\pdfshellescape
2571 \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 `os.execute` method otherwise.

```
2572 \ifnum\markdownMode=0\relax
2573   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
2574 \else
2575   \def\markdownExecuteDirect#1{%
2576     \directlua{os.execute("\luaescapestring{#1}")}}%
```


2577 \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.

```
2578 \def\markdownExecute#1{%
2579   \ifnum\markdownExecuteShellEscape=1\relax
2580     \markdownExecuteDirect{#1}%
2581   \else
2582     \markdownError{I can not access the shell}{Either run the TeX
2583       compiler with the --shell-escape or the --enable-write18 flag,
2584       or set shell_escape=t in the texmf.cnf file}%
2585   \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.

```
2586 \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.

```
2587   \immediate\openout\markdownLuaExecuteFileStream=%
2588     \markdownOptionHelperScriptFileName
2589   \markdownInfo{Writing a helper Lua script to the file
2590     "\markdownOptionHelperScriptFileName"}%
2591   \immediate\write\markdownLuaExecuteFileStream{%
2592     local kpse = require('kpse')
2593     kpse.set_program_name('luatex') #1}%
2594   \immediate\closeout\markdownLuaExecuteFileStream
```

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

```
2595   \markdownInfo{Executing a helper Lua script from the file
2596     "\markdownOptionHelperScriptFileName" and storing the result in the
2597     file "\markdownOptionOutputTempFileName"}%
2598   \markdownExecute{texlua "\markdownOptionHelperScriptFileName" >
2599     "\markdownOptionOutputTempFileName"}%
```

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

```
2600   \input\markdownOptionOutputTempFileName\relax}%
```

3.2.6 Direct Lua Access

The following TeX code is intended for TeX engines that provide direct access to Lua (LuaTeX). The macro `\markdownLuaExecute` defined here and in Section 3.2.5

are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```
2601 \else
2602 \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 3.2.5,

```
2603 \def\markdownLuaExecute#1{\directlua{local print = tex.print #1}}%
2604 \fi
```

3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain \TeX .

```
2605 \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.

```
2606 \catcode'\=0%
2607 \catcode'\=12%
2608 \gdef\markdownInput#1{%
2609     \markdownInfo{Including markdown document "#1"}%
2610     \markdownLuaExecute{%
2611         \markdownPrepare
2612         local input = assert(io.open("#1","r")):read("*a")
```

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

```
2613         print(convert(input:gsub("\r\n?", "\n")))}%
2614 \endgroup
```

3.3 \LaTeX Implementation

The \LaTeX implementation makes use of the fact that, apart from some subtle differences, \TeX implements the majority of the plain \TeX format (see [5, Section 9]). As a consequence, we can directly reuse the existing plain \TeX implementation.

```
2615 \input markdown
2616 \def\markdownVersionSpace{ }%
2617 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
2618 \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Logging Facilities

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

```
2619 \renewcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2620 \renewcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2621 \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 \TeX interface (see Section 2.3.2).

```
2622 \let\markdownInputPlainTeX\markdownInput
2623 \renewcommand\markdownInput[2][{}]{%
2624   \begingroup
2625     \markdownSetup{#1}%
2626     \markdownInputPlainTeX{#2}%
2627   \endgroup}%

```

The `markdown`, and `markdown*` \TeX environments are implemented using the `\markdownReadAndConvert` macro.

```
2628 \renewenvironment{markdown}{%
2629   \markdownReadAndConvert@markdown{}}\relax
2630 \renewenvironment{markdown*}[1]{%
2631   \markdownSetup{#1}%
2632   \markdownReadAndConvert@markdown*}\relax
2633 \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*.

```
2634 \catcode'\|=0\catcode'\<=1\catcode'\>=2%
2635 \catcode'\|=12\catcode'\{=12\catcode'\}=12%
2636 \gdef\markdownReadAndConvert@markdown#1<%
2637   \markdownReadAndConvert<\end{markdown#1}>%
2638   <\end<markdown#1>>>%
2639 \endgroup

```

3.3.3 Options

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

```
2640 \DeclareOption*{%
2641   \expandafter\markdownSetup\expandafter{\CurrentOption}}%

```

```
2642 \ProcessOptions\relax
```

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

```
2643 \define@key{markdownOptions}{renderers}{%
2644   \setkeys{markdownRenderers}{#1}%
2645   \def\KV@prefix{KV@markdownOptions@}}%
2646 \define@key{markdownOptions}{rendererPrototypes}{%
2647   \setkeys{markdownRendererPrototypes}{#1}%
2648   \def\KV@prefix{KV@markdownOptions@}}%
```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
2649 \RequirePackage{url}
2650 \RequirePackage{graphicx}
```

If the `\markdownOptionTightLists` macro expands to `false`, do not load the `paralist` package. This is necessary for $\text{\LaTeX 2}_{\epsilon}$ document classes that do not play nice with `paralist`, such as `beamer`. If the `\markdownOptionTightLists` is undefined and the `beamer` document class is in use, then do not load the `paralist` package either.

```
2651 \RequirePackage{ifthen}
2652 \ifx\markdownOptionTightLists\undefined
2653   \@ifclassloaded{beamer}{}{
2654     \RequirePackage{paralist}}
2655 \else
2656   \ifthenelse{\equal{\markdownOptionTightLists}{false}}{}{
2657     \RequirePackage{paralist}}
2658 \fi
```

If we loaded the `paralist` package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
2659 \@ifpackageloaded{paralist}{
2660   \markdownSetup{rendererPrototypes={
2661     ulBeginTight = {\begin{compactitem}},
2662     ulEndTight = {\end{compactitem}},
2663     olBeginTight = {\begin{compactenum}},
2664     olEndTight = {\end{compactenum}},
2665     dlBeginTight = {\begin{compactdesc}},
2666     dlEndTight = {\end{compactdesc}}}}
2667 }{
2668   \markdownSetup{rendererPrototypes={
2669     ulBeginTight = {\markdownRendererUlBegin},
2670     ulEndTight = {\markdownRendererUlEnd},
2671     olBeginTight = {\markdownRendererOlBegin},
2672     olEndTight = {\markdownRendererOlEnd},
2673     dlBeginTight = {\markdownRendererDlBegin},
```

```

2674     dlEndTight = {\markdownRendererDlEnd}}}}
2675 \RequirePackage{fancyvrb}
2676 \RequirePackage{csvsimple}
2677 \markdownSetup{rendererPrototypes={
2678     lineBreak = {\},
2679     leftBrace = {\textbraceleft},
2680     rightBrace = {\textbraceright},
2681     dollarSign = {\textdollar},
2682     underscore = {\textunderscore},
2683     circumflex = {\textasciicircum},
2684     backslash = {\textbackslash},
2685     tilde = {\textasciitilde},
2686     pipe = {\textbar},
2687     codeSpan = {\texttt{#1}},
2688     link = {#1\footnote{\ifx\empty#4\empty\else#4:
2689         \fi\texttt<\url{#3}\texttt>}},
2690     contentBlock = {%
2691         \ifthenelse{\equal{#1}{csv}}{%
2692             \begin{table}%
2693             \begin{center}%
2694                 \csvautotabular{#3}%
2695             \end{center}
2696             \ifx\empty#4\empty\else
2697                 \caption{#4}%
2698             \fi
2699             \label{tab:#1}%
2700         \end{table}}{%
2701         \markdownInput{#3}}},
2702     image = {%
2703         \begin{figure}%
2704         \begin{center}%
2705             \includegraphics{#3}%
2706         \end{center}%
2707         \ifx\empty#4\empty\else
2708             \caption{#4}%
2709         \fi
2710         \label{fig:#1}%
2711         \end{figure}},
2712     ulBegin = {\begin{itemize}},
2713     ulItem = {\item},
2714     ulEnd = {\end{itemize}},
2715     olBegin = {\begin{enumerate}},
2716     olItem = {\item},
2717     olItemWithNumber = {\item[#1.]},
2718     olEnd = {\end{enumerate}},
2719     dlBegin = {\begin{description}},
2720     dlItem = {\item[#1]},

```

```

2721 dlEnd = {\end{description}},
2722 emphasis = {\emph{#1}},
2723 blockQuoteBegin = {\begin{quotation}},
2724 blockQuoteEnd = {\end{quotation}},
2725 inputVerbatim = {\VerbatimInput{#1}},
2726 inputFencedCode = {%
2727     \ifx\relax#2\relax
2728         \VerbatimInput{#1}%
2729     \else
2730         \ifx\minted@jobname\undefined
2731             \ifx\lst@version\undefined
2732                 \markdownRendererInputFencedCode{#1}{}%

```

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

```

2733     \else
2734         \lstinputlisting[language=#2]{#1}%
2735     \fi

```

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

```

2736     \else
2737         \inputminted{#2}{#1}%
2738     \fi
2739 \fi},
2740 horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
2741 footnote = {\footnote{#1}}}

```

Support the nesting of strong emphasis.

```

2742 \newif\ifmarkdownLATEXStrongEmphasisNested
2743 \markdownLATEXStrongEmphasisNestedfalse
2744 \markdownSetup{rendererPrototypes={
2745     strongEmphasis = {%
2746         \ifmarkdownLATEXStrongEmphasisNested
2747             \markdownLATEXStrongEmphasisNestedfalse
2748             \textmd{#1}%
2749         \markdownLATEXStrongEmphasisNestedtrue
2750     \else
2751         \markdownLATEXStrongEmphasisNestedtrue
2752         \textbf{#1}%
2753     \markdownLATEXStrongEmphasisNestedfalse
2754     \fi}}}

```

Support \LaTeX document classes that do not provide chapters.

```

2755 \ifx\chapter\undefined
2756     \markdownSetup{rendererPrototypes = {
2757         headingOne = {\section{#1}},
2758         headingTwo = {\subsection{#1}},
2759         headingThree = {\subsubsection{#1}},
2760         headingFour = {\paragraph{#1}},

```

```

2761     headingFive = {\subparagraph{#1}}}}
2762 \else
2763   \markdownSetup{rendererPrototypes = {
2764     headingOne = {\chapter{#1}},
2765     headingTwo = {\section{#1}},
2766     headingThree = {\subsection{#1}},
2767     headingFour = {\subsubsection{#1}},
2768     headingFive = {\paragraph{#1}},
2769     headingSix = {\subparagraph{#1}}}}
2770 \fi

```

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

```

2771 \newcount\markdownLaTeXCitationsCounter
2772
2773 % Basic implementation
2774 \def\markdownLaTeXBasicCitations#1#2#3#4{%
2775   \advance\markdownLaTeXCitationsCounter by 1\relax
2776   \ifx\relax#2\relax\else#2~\fi\cite[#3]{#4}%
2777   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2778     \expandafter\@gobble
2779   \fi\markdownLaTeXBasicCitations}
2780 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
2781
2782 % BibLaTeX implementation
2783 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
2784   \advance\markdownLaTeXCitationsCounter by 1\relax
2785   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2786     \autocites#1[#3][#4]{#5}%
2787     \expandafter\@gobbletwo
2788   \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
2789 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
2790   \advance\markdownLaTeXCitationsCounter by 1\relax
2791   \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
2792     \textcites#1[#3][#4]{#5}%
2793     \expandafter\@gobbletwo
2794   \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
2795
2796 \markdownSetup{rendererPrototypes = {
2797   cite = {%
2798     \markdownLaTeXCitationsCounter=1%
2799     \def\markdownLaTeXCitationsTotal{#1}%
2800     \ifx\autocites\undefined
2801       \expandafter
2802       \markdownLaTeXBasicCitations
2803     \else

```

```

2804     \expandafter\expandafter\expandafter
2805     \markdownLaTeXBibLaTeXCitations
2806     \expandafter{\expandafter}%
2807     \fi},
2808     textCite = {%
2809         \markdownLaTeXCitationsCounter=1%
2810         \def\markdownLaTeXCitationsTotal{#1}%
2811         \ifx\textcites\undefined
2812             \expandafter
2813             \markdownLaTeXBasicTextCitations
2814         \else
2815             \expandafter\expandafter\expandafter
2816             \markdownLaTeXBibLaTeXTextCitations
2817             \expandafter{\expandafter}%
2818         \fi}}

```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```

2819 \newcommand\markdownMakeOther{%
2820     \count0=128\relax
2821     \loop
2822         \catcode\count0=11\relax
2823         \advance\count0 by 1\relax
2824     \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain T_EX implementation. As a consequence, we can directly reuse the existing plain T_EX implementation after supplying the missing plain T_EX macros.

```

2825 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2826     \do\#\do\^\do\_ \do\%\do\~}%
2827 \input markdown

```

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents LaTeX package.

```

2828 \def\markdownMakeOther{%

```



```

2829 \count0=128\relax
2830 \loop
2831   \catcode\count0=11\relax
2832   \advance\count0 by 1\relax
2833 \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in `ConTEXt`.

```

2834 \catcode'\=12}%

```

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.

```

2835 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2836 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%

```

3.4.2 Typesetting Markdown

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

```

2837 \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*.

```

2838 \catcode'\|=0%
2839 \catcode'\=12%
2840 |gdef|startmarkdown{%
2841   |markdownReadAndConvert{\stopmarkdown}%
2842   {|stopmarkdown}}%
2843 |endgroup

```

3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

2844 \def\markdownRendererLineBreakPrototype{\blank}%
2845 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
2846 \def\markdownRendererRightBracePrototype{\textbraceright}%
2847 \def\markdownRendererDollarSignPrototype{\textdollar}%
2848 \def\markdownRendererPercentSignPrototype{\percent}%
2849 \def\markdownRendererUnderscorePrototype{\textunderscore}%
2850 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
2851 \def\markdownRendererBackslashPrototype{\textbackslash}%
2852 \def\markdownRendererTildePrototype{\textasciitilde}%

```

```

2853 \def\markdownRendererPipePrototype{\char' |}%
2854 \def\markdownRendererLinkPrototype#1#2#3#4{%
2855   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
2856   \fi\tt<\hyphenatedurl{#3}>}}%
2857 \usemodule[database]
2858 \defineseparatedlist
2859   [MarkdownConTeXtCSV]
2860   [separator={,},
2861   before=\bTABLE,after=\eTABLE,
2862   first=\bTR,last=\eTR,
2863   left=\bTD,right=\eTD]
2864 \def\markdownConTeXtCSV{csv}
2865 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
2866   \def\markdownConTeXtCSV@arg{#1}%
2867   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
2868     \placetable[] [tab:#1]{#4}{%
2869     \processseparatedfile[MarkdownConTeXtCSV][#3]}%
2870   \else
2871     \markdownInput{#3}%
2872   \fi}%
2873 \def\markdownRendererImagePrototype#1#2#3#4{%
2874   \placefigure[] [fig:#1]{#4}{\externalfigure[#3]}%
2875 \def\markdownRendererU1BeginPrototype{\startitemize}%
2876 \def\markdownRendererU1BeginTightPrototype{\startitemize[packed]}%
2877 \def\markdownRendererU1ItemPrototype{\item}%
2878 \def\markdownRendererU1EndPrototype{\stopitemize}%
2879 \def\markdownRendererU1EndTightPrototype{\stopitemize}%
2880 \def\markdownRendererO1BeginPrototype{\startitemize[n]}%
2881 \def\markdownRendererO1BeginTightPrototype{\startitemize[packed,n]}%
2882 \def\markdownRendererO1ItemPrototype{\item}%
2883 \def\markdownRendererO1ItemWithNumberPrototype#1{\sym{#1.}}%
2884 \def\markdownRendererO1EndPrototype{\stopitemize}%
2885 \def\markdownRendererO1EndTightPrototype{\stopitemize}%
2886 \definedescription
2887   [MarkdownConTeXtDlItemPrototype]
2888   [location=hanging,
2889   margin=standard,
2890   headstyle=bold]%
2891 \definestartstop
2892   [MarkdownConTeXtDlPrototype]
2893   [before=\blank,
2894   after=\blank]%
2895 \definestartstop
2896   [MarkdownConTeXtDlTightPrototype]
2897   [before=\blank\startpacked,
2898   after=\stoppacked\blank]%
2899 \def\markdownRendererDlBeginPrototype{%

```

```

2900 \startMarkdownConTeXtDlPrototype}%
2901 \def\markdownRendererDlBeginTightPrototype{%
2902 \startMarkdownConTeXtDlTightPrototype}%
2903 \def\markdownRendererDlItemPrototype#1{%
2904 \startMarkdownConTeXtDlItemPrototype{#1}}%
2905 \def\markdownRendererDlItemEndPrototype{%
2906 \stopMarkdownConTeXtDlItemPrototype}%
2907 \def\markdownRendererDlEndPrototype{%
2908 \stopMarkdownConTeXtDlPrototype}%
2909 \def\markdownRendererDlEndTightPrototype{%
2910 \stopMarkdownConTeXtDlTightPrototype}%
2911 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
2912 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
2913 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
2914 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
2915 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
2916 \def\markdownRendererInputFencedCodePrototype#1#2{%
2917 \ifx\relax#2\relax
2918 \typefile{#1}%
2919 \else

```

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

```

% Map the 'TEX' syntax highlighter to the 'latex' infostring.
\definetying [latex]
\setuptyping [latex] [option=TEX]

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

```

```

2920 \typefile[#2] []{#1}%
2921 \fi}%
2922 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
2923 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
2924 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
2925 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
2926 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%

```

```

2927 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
2928 \def\markdownRendererHorizontalRulePrototype{%
2929   \blackrule[height=1pt, width=\hsize]}%
2930 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
2931 \stopmodule\protect

```

References

1. L^AT_EX DEVELOPMENT TEAM. *LuaT_EX reference manual* [online]. 2016 [visited on 2016-11-27]. Available from: <http://www.luatex.org/svn/trunk/manual/luatex.pdf>.
2. SOTKOV, Anton. *File transclusion syntax for Markdown* [online]. 2017 [visited on 2017-03-18]. Available from: <https://github.com/iainc/Markdown-Content-Blocks>.
3. KNUTH, Donald Ervin. *The T_EXbook*. 3rd ed. Addison-Westley, 1986. ISBN 0-201-13447-0.
4. IERUSALIMSKY, Roberto. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. ISBN 978-85-903798-5-0.
5. BRAAMS, Johannes; CARLISLE, David; JEFFREY, Alan; LAMPORT, Leslie; MITTELBACH, Frank; ROWLEY, Chris; SCHÖPF, Rainer. *The L^AT_EX 2_ε Sources* [online]. 2016 [visited on 2016-09-27]. Available from: <http://mirrors.ctan.org/macros/latex/base/source2e.pdf>.