

# lua-list-hyphen — Per-language listing of hyphenated words for Lua $\text{\LaTeX}$ \*

Alan J. Cain<sup>†</sup>

Released 2026-04-28

## Abstract

This Lua $\text{\LaTeX}$  package writes each word that has been hyphenated across lines to a file, using a different file for each language, for subsequent external checking.

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Introduction</b>  | <b>2</b> |
| <b>2</b> | <b>Requirements</b>  | <b>3</b> |
| <b>3</b> | <b>Installation</b>  | <b>3</b> |
| <b>4</b> | <b>Getting started</b>                                     | <b>3</b> |
| <b>5</b> | <b>Package options</b>                                     | <b>3</b> |
| <b>6</b> | <b>Usage notes</b>   | <b>4</b> |
| 6.1      | Languages . . . . .  | 4        |
| 6.2      | Limitations . . . . .                                      | 4        |
| <b>7</b> | <b>Implementation (<math>\text{\LaTeX}</math> package)</b> | <b>6</b> |
| 7.1      | Initial set-up . . . . .                                   | 6        |
| 7.2      | Options . . . . .  | 6        |
| 7.3      | Lua backend and interface . . . . .                        | 7        |
| 7.4      | Saving language names . . . . .                            | 7        |
| 7.5      | Processing and writing hyphenation lists . . . . .         | 8        |

---

\*This document describes v0.2.50, last revised 2026-04-28.

<sup>†</sup>a.j.cain (AT) gmail.com

|          |  |          |
|----------|--|----------|
| <b>8</b> | <b>Implementation (Lua backend)</b>              | <b>8</b> |
| 8.1      | Debugging function . . . . .                     | 8        |
| 8.2      | Table key constants . . . . .                    | 9        |
| 8.3      | Node ID and subtype constants . . . . .          | 9        |
| 8.4      | Utility functions . . . . .                      | 9        |
| 8.5      | Getting text from nodes . . . . .                | 10       |
| 8.6      | String manipulation . . . . .                    | 12       |
| 8.7      | Pre-linebreak processing . . . . .               | 13       |
| 8.8      | Post-linebreak processing . . . . .              | 15       |
| 8.9      | Callbacks . . . . .                              | 19       |
| 8.10     | Language settings . . . . .                      | 20       |
| 8.11     | Processing hyphenation lists . . . . .           | 20       |
|          | 8.11.1 Comparisons and equality checks . . . . . | 20       |
|          | 8.11.2 Sorting . . . . .                         | 22       |
|          | 8.11.3 Deduplication . . . . .                   | 23       |
|          | 8.11.4 Combined processing . . . . .             | 23       |
| 8.12     | Writing . . . . .                                | 24       |
| 8.13     | Export public functions . . . . .                | 25       |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>27</b> |
|--------------|-----------|

# 1 Introduction

TeX’s algorithm for finding points where a word can be hyphenated is good, but not perfect.<sup>1</sup> The present author writes in British English, where the valid division points can depend on both the pronunciation of a word and its internal structure (and hence its etymology). Currently, TeX’s pattern-based approach produces *bio-lo-gic*, *bio-logy*, *bio-lo-gist*, rather than the standard *bio-logic*, *biol-ogy*, *biolo-gist*.<sup>2</sup> To deal with such cases, at least a substantially larger number of patterns would be required than are available at present. There are also various words where the valid division points in British English cannot be deduced from their spelling alone: for instance, the verbs *at-trib-ute*, *pre-sent*, *pro-duce*, *re-cord* have different division points from the orthographically identical nouns *at-tri-bute*, *pres-ent*, *prod-uce*, *rec-ord*. For another example, compare *cur-ric-ulum vitae* and *school cur-ricu-lum*.

Easy checking of the chosen hyphenations is desirable. With LuaTeX, it is possible to extract the hyphenated words. The LuaLaTeX package `lua-check-hyphen` offers this facility. It checks hyphenated words against a whitelist, visually flags unknown hyphenations, and writes unknown hyphenations to a file. But it was first written in 2012, when LuaTeX was at an earlier stage of development, and so it has certain problems, such as with words containing ligatures. It also lacks multi-language support.

This LuaLaTeX package, `lua-list-hyphen`, uses some ideas from `lua-check-hyphen` but was written from scratch to work with a modern LuaTeX. It simply writes hyphenated words from each language to a separate file, so that they can be checked (manually or by an external program).

<sup>1</sup>For a description of the algorithm and its limitations, see Knuth’s account in Appendix H of *The TeXbook* (Addison-Wesley, 2021. ISBN: 978-0-201-13447-6)

<sup>2</sup>See the *New Oxford Spelling Dictionary*, which is the authority for word divisions in British English (Oxford University Press, 2005. ISBN: 978-0-19-860881-3).

[The author has written a simple Python application `hyphenassist`<sup>3</sup> that checks the listed hyphenations against a dictionary of valid divisions and allows the user to quickly choose to add entries to the division dictionary, add hyphenation exceptions, or ignore particular hyphenations. He has used this program in conjunction with code incorporated into this package to check hyphenations in his own books.<sup>4</sup>]

**Licence.** `lua-list-hyphen` is released under the L<sup>A</sup>T<sub>E</sub>X Project Public Licence v1.3c or later.<sup>5</sup>

**Acknowledgements.** The author thanks Keno Wehr for corrections and comments on the documentation.

**Feature requests and bug reports** The development code and issue tracker are hosted at Codeberg.<sup>6</sup>

## 2 Requirements

`lua-list-hyphen` requires

- (1) LuaL<sup>A</sup>T<sub>E</sub>X,
- (2) a recent L<sup>A</sup>T<sub>E</sub>X kernel with `expl3` support (any kernel version since 2020-02-02 should suffice).

It does not depend on any other packages, but will interface with `babel` or `polyglossia` (if one of them is loaded) to determine language names.

## 3 Installation

To install `lua-list-hyphen` manually, run `luatex lua-list-hyphen.ins` and copy `lua-list-hyphen.sty` and `lua-list-hyphen.lua` to somewhere LuaL<sup>A</sup>T<sub>E</sub>X can find them.

## 4 Getting started

Simply load the package; the hyphenated words are by default written to the file `\jobname-⟨lang-id⟩.hyph`, without being sorted or having duplicates removed. The `⟨lang-id⟩` is either a LuaT<sub>E</sub>X numerical language ID, or a `babel` or `polyglossia` name of the language, if one of these packages is in use. The prefix `\jobname-` and the extension `.hyph` can be customized; see [Section 5](#).

## 5 Package options

**verbose** The boolean option `verbose` controls how much information is written to the file about

---

<sup>3</sup>URL: <https://codeberg.org/ajcain/hyphenassist>.

<sup>4</sup>In particular, *Form & Number: A History of Mathematical Beauty*. URL: [https://archive.org/details/cain\\_formandnumber\\_ebook\\_large](https://archive.org/details/cain_formandnumber_ebook_large).

<sup>5</sup>URL: <https://www.latex-project.org/lppl.txt>

<sup>6</sup>URL: <https://codeberg.org/ajcain/lua-list-hyphen>

each hyphenated word. When **true**, for each hyphenated word, both the undivided original and the divided word are written out (on the same line). When **false**, only the hyphenated word is written. (*Default: false*)

**unique** The option **unique** controls removal of duplicates from the list of hyphenated words written out. It can be set to one of the following three values:

**none:** Duplicate hyphenations are not removed.

**case:** Hyphenations that are duplicate (case-sensitively) are removed. In this case, the hyphenations **geo-metry** and **Geo-metry** are considered to be distinct.

**nocase:** Hyphenations that are duplicate (case-insensitively) are removed. In this case, the hyphenations **geo-metry** and **Geo-metry** are considered to be duplicates. The case of each listed hyphenation will be that of the first appearance of that hyphenation.

(*Default: none*)

**sort** The option **sort** controls sorting of the list of hyphenated words. It can be set to one of the following three values:

**none:** Hyphenations appear in the same order as they occur in the document, or, if duplicates are removed, in the order of first appearance in the document.

**case:** Hyphenations are sorted case-sensitively. In this case, **Geo-metry** precedes **geo-meter**.

**nocase:** Hyphenations are sorted case-insensitively. In this case, **geo-meter** precedes **Geo-metry**.

(*Default: none*)

The two options **prefix** and **extension** specify the files to which hyphenations are written. Between the prefix and the extension is either a LuaTeX numerical language ID, or a **babel** or **polyglossia** name of the language, if one of these packages is in use.

**prefix** The **prefix** is the part of the file name to which the list of hyphenated words is written, before the language ID. (*Default: \jobname-* (note the hyphen).)

**extension** The extension of the file (including the **.**) to which the list of hyphenated words for each language is written. (*Default: .hyph*)

**debug** The boolean option **debug** controls whether debugging information is written to the terminal. (*Default: false*)

## 6 Usage notes

### 6.1 Languages

To determine the language of a word, **lua-list-hyphen** looks at what language is applied at the first possible hyphenation point, first considering the part of the word before it, then the part after it. In the (presumably rare) case of a ‘mixed-language’ word like ‘near-Zugzwang’ being specified (using, for example, **babel**) with **near-\foreignlanguage{german}{Zugzwang}**, it would be assigned to the language in which ‘near-’ is set.

Duplicates are removed within each language. If the same hyphenation occurs in two different languages, it will appear in both files, regardless of the value of **unique**.

## 6.2 Limitations

`lua-list-hyphen` uses Lua<sub>T</sub><sub>E</sub>X's built-in functions for pattern matching and converting between upper and lower case, which are based on the `slnunicode` library. This library has not been updated for some time and is based on an out-of-date version of the Unicode standard. Thus there may be problems with languages added to Unicode more recently. Hyphenated words from such languages should still be listed, but may contain extraneous characters and may not be sorted correctly. Users may prefer to leave sorting and removal of duplicates to an external program that adheres to the current Unicode standard.

## 7 Implementation (L<sup>A</sup>T<sub>E</sub>X package)

```

1 <*package>
2 <@@=lualisthyphen>

```

### 7.1 Initial set-up

Package identification/version information.

```

3 \NeedsTeXFormat{LaTeX2e}[2020-02-02]
4 \ProvidesExplPackage{lua-list-hyphen}{2026-04-28}{0.2.50}
5   {Listing hyphenated words for LuaLaTeX}

```

Check that LuaT<sub>E</sub>X is in use.

```

6 \sys_if_engine luatex:F
7   {
8     \msg_new:nnn{ lua-list-hyphen }{ luatex_required }
9     { LuaLaTeX~required.~Package~loading~will~abort. }
10    \msg_critical:nn{ lua-list-hyphen }{ luatex_required }
11  }

```

### 7.2 Options

`\l_lualisthyphen_verbose_bool` Boolean option to indicate whether lists of hyphenations should be written verbosely.

```

12 \keys_define:nn { lua-list-hyphen }{
13   verbose .bool_set:N = \l_lualisthyphen_verbose_bool,
14 }

```

*(End of definition for \l\_lualisthyphen\_verbose\_bool.)*

`\l_lualisthyphen_unique_int` Choice option to indicate whether lists of hyphenations should have duplicates removed, case-sensitively or case-insensitively.

```

15 \int_new:N\l_lualisthyphen_unique_int
16 \keys_define:nn { lua-list-hyphen }{
17   unique .choices:nn = { none, case, nocase }{
18     \int_set:Nn\l_lualisthyphen_unique_int{ \l_keys_choice_int - 1 }
19   },
20 }

```

*(End of definition for \l\_lualisthyphen\_unique\_int.)*

`\l_lualisthyphen_sort_int` Choice option to indicate whether lists of hyphenations should be sorted, case-sensitively or case-insensitively.

```

21 \int_new:N\l_lualisthyphen_sort_int
22 \keys_define:nn { lua-list-hyphen }{
23   sort .choices:nn = { none, case, nocase }{
24     \int_set:Nn\l_lualisthyphen_sort_int{ \l_keys_choice_int - 1 }
25   },
26 }

```

*(End of definition for \l\_lualisthyphen\_sort\_int.)*

`\l_lualisthyphen_file_prefix_str` String option for the prefix of file files.

```

27 \keys_define:nn { lua-list-hyphen }{
28   prefix .str_set:N = \l_lualisthyphen_file_prefix_str,
29   prefix .initial:e = { \c_sys_jobname_str- },
30 }

```

(End of definition for `\l__lualisthyphen_file_prefix_str`.)

`\l__lualisthyphen_file_extension_str` String option for the file extension of file files.

```

31 \keys_define:nn { lua-list-hyphen }{
32   extension .str_set:N = \l__lualisthyphen_file_extension_str,
33   extension .initial:n = { .hyph },
34 }

```

(End of definition for `\l__lualisthyphen_file_extension_str`.)

`\l__lualisthyphen_debug_int` Option to specify whether debug information is written to the terminal. Not intended for end users.

```

35 \int_new:N\l__lualisthyphen_debug_int
36 \keys_define:nn { lua-list-hyphen }{
37   debug .code:n = {\int_set_eq:NN\l__lualisthyphen_debug_int\c_one_int}
38 }

```

(End of definition for `\l__lualisthyphen_debug_int`.)

Process package options.

```

39 \ProcessKeyOptions [ lua-list-hyphen ]
    Convert boolean options to integers (which can be accessed from Lua).
40 \int_new:N\l__lualisthyphen_verbose_int
41 \bool_if:NT\l__lualisthyphen_verbose_bool
42   { \int_set_eq:NN\l__lualisthyphen_verbose_int\c_one_int }

```

### 7.3 Lua backend and interface

Load the Lua backend.

```

43 \lua_now:n{
44   lualisthyphen = require('lua-list-hyphen')
45 }

```

### 7.4 Saving language names

At `enddocument/afterlastpage`, if possible save `babel`'s language names. (`polyglossia`'s names can be found directly from Lua.)

```

46 \hook_gput_code:nnn{ enddocument/afterlastpage }{ lua-list-hyphen }{
47   \__lualisthyphen_babel_save_language_names:
48 }

```

`\__lualisthyphen_babel_save_language_names:` If `babel` is in use, get language names from `\bbl@languages`.

```

49 \cs_new:Npn \__lualisthyphen_babel_save_language_names:
50   {
51     \cs_if_exist:NT\bbl@languages
52     {

```

Iterate through `\bbl@languages` to get language names. Items stored in this macro are quadruples prefixed with `\bbl@elt`, so locally redefine this latter macro to an auxiliary function that passes language ID/name pairs to the Lua backend.

```

53       \group_begin:
54       \cs_set_eq:NN
55       \bbl@elt

```

```

56         \__lualisthyphen_babel_save_language_names_elt:nnnn
57         \bbl@languages
58         \group_end:
59     }
60 }

```

(End of definition for \\_\_lualisthyphen\_babel\_save\_language\_names:.)

sthyphen\_babel\_save\_language\_names\_elt:nnnn Auxiliary function that takes a quadruple stored in \bbl@languages and passes language ID/name pairs to the Lua backend.

```

61 \cs_new:Npn \__lualisthyphen_babel_save_language_names_elt:nnnn #1#2#3#4
62 {
63     \lua_now:n{
64         lualisthyphen.babel_save_language_name(#2,'#1')
65     }
66 }

```

(End of definition for \\_\_lualisthyphen\_babel\_save\_language\_names\_elt:nnnn.)

## 7.5 Processing and writing hyphenation lists

At `enddocument/info`, write the list of hyphenations for each language.

```

67 \hook_gput_code:nnn{ enddocument/info }{ lua-list-hyphen } {
68     \__lualisthyphen_process_write_hyphenation_lists:ee
69     {\str_use:N\l__lualisthyphen_file_prefix_str}
70     {\str_use:N\l__lualisthyphen_file_extension_str}
71 }

```

sthyphen\_process\_write\_hyphenation\_lists:nn Write hyphenations lists to files with prefix given in the first parameter and suffix in the second.

```

72 \cs_new:Npn \__lualisthyphen_process_write_hyphenation_lists:nn #1#2
73 {
74     \lua_now:e{
75         lualisthyphen.process_write_hyphenation_lists(
76             '\luaescapestring{#1}',
77             '\luaescapestring{#2}'
78         )
79     }
80 }
81 \cs_generate_variant:Nn
82 \__lualisthyphen_process_write_hyphenation_lists:nn
83 { ee }

```

(End of definition for \\_\_lualisthyphen\_process\_write\_hyphenation\_lists:nn.)

```

84 \</package>

```

## 8 Implementation (Lua backend)

```

85 \<lua>

```

### 8.1 Debugging function

debug Debugging function. Initially defined to do nothing, then overridden to become a function that actually writes debugging information if the package option was set.



```

86 local function debug(s)
87 end
88
89 if tex.count['l__lualisthyphen_debug_int'] ~= 0 then
90     debug = function(s)
91         print('lua-list-hyphen DEBUG: ' .. s)
92     end
93 end

```

(End of definition for debug.)

## 8.2 Table key constants

Keys for tables containing hyphenatable/hyphenated word data.

```

94 local KEY_WORD = 'word'
95 local KEY_LANG = 'lang'
96 local KEY_DIVISION = 'division'
97 local KEY_INDEX = 'index'

```

## 8.3 Node ID and subtype constants

Define constants for the node IDs that need to be recognized.

```

98 local NODE_ID_HLIST = node.id('hlist')
99 local NODE_ID_DISC = node.id('disc')
100 local NODE_ID_GLUE = node.id('glue')
101 local NODE_ID_KERN = node.id('kern')
102 local NODE_ID_MARGIN_KERN = node.id('margin_kern')
103 local NODE_ID_GLYPH = node.id('glyph')

```

Define a constant for the kern node subtype that needs to be recognized. There seems to be no automatic way to get the numerical value from the subtype other than searching the `node.subtype('kern')` table.

```

104 local NODE_KERN_SUBTYPE_FONTKERN
105 for k,v in pairs(node.subtypes('kern')) do
106     if v == 'fontkern' then
107         NODE_KERN_SUBTYPE_FONTKERN = k
108         break
109     end
110 end

```

## 8.4 Utility functions

`list_filter` Take a list `t` and remove from it any elements for which the function `f` does not return true. (The index `j` is always the destination index to which a ‘keep’ element is moved.)<sup>7</sup>

```

111 local function list_filter(t, f)
112     local j = 1
113     local n = #t
114
115     for i=1,n do
116         if (f(t[i])) then
117             if (i ~= j) then

```

<sup>7</sup>Code adapted from <https://stackoverflow.com/a/53038524>.

```

118         t[j] = t[i]
119         t[i] = nil
120     end
121     j = j + 1
122 else
123     t[i] = nil
124 end
125 end
126
127 end

```

*(End of definition for list\_filter.)*

`list_uniq` Take a list `t` and remove from it adjacent elements for which the function `f` returns true. (The index `j` is always the last ‘kept’ element.)

```

128 local function list_uniq(t, f)
129     local j = 1
130     local n = #t
131
132     for i=2,n do
133         if (f(t[i],t[j])) then
134             t[i] = nil
135         else
136             j = i
137         end
138     end
139
140     list_filter(
141         t,
142         function(a) return a end
143     )
144 end

```

*(End of definition for list\_uniq.)*

## 8.5 Getting text from nodes

Getting the components of the ligatures that have Unicode code points can be problematic, at least for some fonts, so define a lookup table for these cases.

```

145 local LIGATURE_COMPONENTS = {
146     [0xfb00] = {'f','f'},
147     [0xfb01] = {'f','i'},
148     [0xfb02] = {'f','l'},
149     [0xfb03] = {'f','f','i'},
150     [0xfb04] = {'f','f','l'},
151 }

```

Extracting text from nodes uses two functions that call each other, so the names have to be defined ahead of time.

```

152 local get_node_text
153 local get_nodelist_text

```

`get_node_text` Return the text content of a glyph node (which might be a normal glyph, a ligature, etc.).

```
154 get_node_text = function(n)
155
156   if n.id == NODE_ID_GLYPH then
157
158     if LIGATURE_COMPONENTS[n.char] ~= nil then
159       local text = ''
160
161       for _,c in ipairs(LIGATURE_COMPONENTS[n.char]) do
162         text = text .. c
163       end
164       return text
165
166     elseif n.components then
167       return get_nodelist_text(n.components)
168     else
169       -- See [https://tug.org/pipermail/luatex/2018-March/006786.html]
170       local u = fonts.hashes.identifiers[n.font].characters[n.char].tounicode
171       return utf8.char(tonumber(u,16))
172     end
173   elseif n.id == NODE_ID_DISC then
174     if n.replace then
175       return get_nodelist_text(n.replace)
176     else
177       return ''
178     end
179   else
180     return ''
181   end
182
183 end
```

*(End of definition for `get_node_text`.)*

`get_nodelist_text` Return the text content of the glyph nodes in the list starting at `head` up to and including the node `last`, or up to the end of the list if `last` is not specified.

```
184 get_nodelist_text = function (head,last)
185
186   local text = ''
187
188   for item in node.traverse(head) do
189
190     text = text .. get_node_text(item)
191
192     if item == last then
193       break
194     end
195   end
196
197   return text
198
199 end
```

*(End of definition for get\_nodelist\_text.)*

`is_possible_word_node` Return boolean indicating if node `n` could be part of a word. Assume that `glyph`, `disc`, and `margin_kern` nodes could be part of a word, as could a `kern` node with subtype `fontkern`.

```
200 local function is_possible_word_node(n)
201
202     return (
203         n.id == NODE_ID_GLYPH
204         or
205         n.id == NODE_ID_DISC
206         or
207         (n.id == NODE_ID_KERN and n.subtype == NODE_KERN_SUBTYPE_FONTKERN)
208         or
209         n.id == NODE_ID_MARGIN_KERN
210     )
211
212 end
```

*(End of definition for is\_possible\_word\_node.)*

## 8.6 String manipulation

`trim_nonletters_both` Remove non-letter characters from both the start and end of a string.

```
213 local function trim_nonletters_both(s)
214
215     return unicode.utf8.match(s, '^%A*(.*)%A*$')
216
217 end
```

*(End of definition for trim\_nonletters\_both.)*

`trim_nonletters_start` Remove non-letter characters from the start of a string.

```
218 local function trim_nonletters_start(s)
219
220     return unicode.utf8.match(s, '^%A*(.*)$')
221
222 end
```

*(End of definition for trim\_nonletters\_start.)*

`trim_nonletters_end` Remove non-letter characters from the end of a string.

```
223 local function trim_nonletters_end(s)
224
225     return unicode.utf8.match(s, '^(.*)%A*$')
226
227 end
```

*(End of definition for trim\_nonletters\_end.)*

## 8.7 Pre-linebreak processing

Before each line has been broken, find all potential division points and store the words in which they occur, linking each potential break point to the corresponding word.

Declare a new attribute, which will be used to store in each disc node the index of the corresponding word in the table `hlist_hyphenatable_word_list`.

```
228 local hyphen_attr = luatexbase.new_attribute('hyphen_attr')
```

Table to hold hyphenatable words found in the hlist that will be broken. This table will be cleared after the post-linebreak processing.

```
229 local hlist_hyphenatable_word_list = {}
```

`get_first_glyph_lang` Return the lang attribute of the first glyph in the the part of the list starting n that could be part of a word. (Currently unused; see the documentation of `get_disc_lang`.)

```
230 -- local function get_first_glyph_lang(n)
231
232 --   local item = n
233 --   while item and is_possible_word_node(item) do
234 --     if item.id == NODE_ID_GLYPH then
235 --       return item.lang
236 --     end
237 --     item = item.next
238 --   end
239
240 --   return nil
241
242 -- end
```

*(End of definition for get\_first\_glyph\_lang.)*

`get_disc_lang` Try to find the language ID in force at a given disc node by looking at (1) the last glyph in the word before the disc node; (2) the first glyph in the word after the disc node. Default to language ID 0.

(Looking at `replace`, `pre`, `post` is possible, but is unreliable and so disabled for the present. The author has encountered the situation where an explicit hyphen results in the hyphen characters in `replace` and `pre` having different language IDs. He has not had time to investigate how this arises from the interaction of `babel/polyglossia` and `LuaATEX`.)

```
243 local function get_disc_lang(n)
244
245 -- lang = get_first_glyph_lang(n.replace)
246 -- if lang then
247 --   print(lang)
248 --   return lang
249 -- end
250
251 -- lang = get_first_glyph_lang(n.pre)
252 -- if lang then
253 --   print(lang)
254 --   return lang
255 -- end
256
257 -- lang = get_first_glyph_lang(n.post)
```

```

258 -- if lang then
259 --   return lang
260 -- end
261
262 local item
263
264 item = n
265 while item and is_possible_word_node(item) do
266   if item.id == NODE_ID_GLYPH then
267     return item.lang
268   end
269   item = item.prev
270 end
271
272 item = n
273 while item and is_possible_word_node(item) do
274   if item.id == NODE_ID_GLYPH then
275     return item.lang
276   end
277   item = item.next
278 end
279
280 return 0
281
282 end

```

*(End of definition for get\_disc\_lang.)*

`pre_linebreak` For every word containing a disc node (a potential division point) in the hlist at `hlist_head`, add the word it which it appears, along with the language of that word to `hlist_hyphenatable_word_list`, and store its index in that table in the `hyphen_attr` attribute (declared above) of each disc node in the word.

```

283 local function pre_linebreak(hlist_head,groupcode)

```

When non-nil, `word_start_node` is the first node of the ‘current’ word. When non-nil, `hyphenatable_index` is the index in `hlist_hyphenatable_word_list` where the word will be added. `hyphenatable_count` is the number of potential hyphenatable words found so far, which is used to set `hyphenatable_index` when the first disc node is found in a new word.

```

284   local word_start_node = nil
285   local hyphenatable_index = nil
286   local hyphenatable_count = 0
287   local lang = nil
288
289   debug('Pre-linebreak processing start')
290
291   for item in node.traverse(hlist_head) do
292     if item.id == NODE_ID_GLYPH and not word_start_node then
293       word_start_node = item
294     end

```

If `item` is a disc node, check whether it is the first one found in the ‘current’ word (indicated by) `hyphenatable_index` being `nil`. If so, set `hyphenatable_index` and determine the language currently being used. Set the attribute of the disc node.

```

295     if item.id == NODE_ID_DISC then
296         if not hyphenatable_index then
297             hyphenatable_count = hyphenatable_count + 1
298             hyphenatable_index = hyphenatable_count
299             lang = get_disc_lang(item)
300         end
301         node.set_attribute(item,hyphen_attr,hyphenatable_index)
302     end

```

If `item` is not a node that can appear in a word assume that the word end has been reached.

```

303     if not is_possible_word_node(item) then
304         if word_start_node and hyphenatable_index then

```

Extract the text of the hyphenatable word. In fact, the ‘word’ might be something other than a genuine word, such as an ISBN (with hyphen separators). So only store the word in `hlist_hyphenatable_word_list` if something non-empty is left after trimming non-letters from both sides.

```

305         local hyphenatable_word = trim_nonletters_both(
306             get_nodelist_text(word_start_node,item.prev)
307         )
308         if hyphenatable_word ~= '' then
309             debug(
310                 ' Hyphenatable word (index ' .. hyphenatable_index .. ') "'
311                 .. hyphenatable_word .. '"'
312             )
313             hlist_hyphenatable_word_list[hyphenatable_index] = {
314                 [KEY_WORD] = hyphenatable_word,
315                 [KEY_LANG] = lang,
316             }
317         end
318     end

```

Reset `word_start_node` and `hyphenatable_index`, ready for the next word.

```

319         word_start_node = nil
320         hyphenatable_index = nil
321     end
322
323 end
324
325 debug('Pre-linebreak processing finish')
326
327 return true
328 end

```

*(End of definition for `pre_linebreak`.)*

## 8.8 Post-linebreak processing

After linebreaking, look for a discretionary node at the end of each line, which indicates that a word has been divided between the end of that line and the start of the next.

Extract the two word-pieces from the lines and store them in the appropriate language table.

`get_used_disc` If at the tail of the hlist at `hlist_head` (which will be a line) there is a disc node not followed by a glyph node, return that disc node. Otherwise return `nil`.

```

329 local function get_used_disc(hlist_head)
330
331   local item = node.tail(hlist_head)
332
333   while item and item.id ~= NODE_ID_GLYPH do
334     if item.id == NODE_ID_DISC then
335       return item
336     end
337     item = item.prev
338   end
339
340   return nil
341
342 end

```

*(End of definition for `get_used_disc`.)*

`get_disc_word_start` Return the node starting the word that includes a given disc node `n`, or `nil` if there is no such node.

```

343 local function get_disc_word_start(hlist_head,n)
344
345   local item = n
346
347   while item do
348     local prev = item.prev
349
350     if not (prev and is_possible_word_node(prev)) then
351       return item
352     end
353
354     item = prev
355   end
356
357   return nil
358 end

```

*(End of definition for `get_disc_word_start`.)*

`get_next_hlist` Return the next hlist in the list containing the given node `n`, or `nil` if there is no such hlist node.

```

359 local function get_next_hlist(n)
360
361   local item = n.next
362
363   while item do
364     if item.id == NODE_ID_HLIST then
365       return item
366     end
367     item = item.next

```



```

368     end
369
370     return nil
371
372 end

```

*(End of definition for get\_next\_hlist.)*

`get_line_first_word` Return the first word in the hlist at `hlist_head`, or nil if there is no such word.

```

373 local function get_line_first_word(hlist_head)
word_start_node is either nil or the (glyph) node that starts the word.
374     local word_start_node = nil
375
376     for item in node.traverse(hlist_head) do
377
378         if item.id == NODE_ID_GLYPH then
379             if not word_start_node then
380                 word_start_node = item
381             end
382         end
383
384         if not is_possible_word_node(item) then
385             if word_start_node then
386                 return get_nodelist_text(word_start_node,item.prev)
387             end
388         end
389
390     end

```

It is possible that the word ends at the end of the hlist, so check if a word has been started.

```

391     if word_start_node then
392         return get_nodelist_text(word_start_node,node.tail(hlist_head))
393     else
394         return nil
395     end
396 end

```

*(End of definition for get\_line\_first\_word.)*

Table for lists for hyphenated words in various languages. This table will be indexed by (numerical) language IDs. Each value will be a list, and each entry in the list will be a table containing the original word, the hyphenation, and the index of the table in the list (which is needed later for stable sorting and sorting into the original order).

```

397 local hyphenation_table = {}

```

`check_line_hyphenation` Check whether there is a hyphenated word at the end of the given hlist; if so, save the word to `hyphenation_list`.

```

398 local function check_line_hyphenation(hlist)
First, is there a disc node at the end of the list? ('End' modulo certain other node types;
see the documentation of get_used_disc.)
399     local last_disc = get_used_disc(hlist.head)
400     if not last_disc then
401         debug(' No disc node found at end of line')

```

```

402     return
403 end

```

The `hyphen_attr` may or may not contain the index of a word in `hlist_hyphenatable_word_list`. (See the documentation for `pre_linebreak`.)

```

404 local hyphenation_index = node.has_attribute(last_disc,hyphen_attr)
405 local t = hlist_hyphenatable_word_list[hyphenation_index]
406 if not t then
407     debug(' Disc node not associated to a stored word')
408     return
409 end
410 local word = t[KEY_WORD]
411 local lang = t[KEY_LANG]

```

There should always be a next line, since there is a disc node at the end of `hlist`, but check anyway.

```

412 local next_line = get_next_hlist(hlist)
413
414 if not next_line then
415     debug(' No following line found (which should not happen)')
416     return
417 end

```

Get the hyphenation list for the language of the word, ensuring that the list has been created.

```

418 lang_hyphenation_list = hyphenation_table[lang]
419 if not lang_hyphenation_list then
420     hyphenation_table[lang] = {}
421     lang_hyphenation_list = hyphenation_table[lang]
422 end

```

For the pre-linebreak part of the word, get the word that ends the line, and trim any leading non-letters. This could leave an empty word; for example, if *n-dimensional* is broken at the hyphen, the word ending the line is just the hyphen. If an empty word is left, just use the non-trimmed result.

```

423 local pre = get_nodelist_text(get_disc_word_start(hlist.head,last_disc))
424 local pre_temp = trim_nonletters_start(pre)
425 if pre_temp ~= '' then
426     pre = pre_temp
427 end

```

For the post-linebreak part, just get the word at the start of the next line, and trim and trailing non-letters.

```

428 local post = trim_nonletters_end(get_line_first_word(next_line.head))
429
430 debug(
431     ' Hyphenated word found: "' .. word .. '" -> "' .. pre .. '<>' .. post .. '"
432 )

```

Store everything in the language hyphenation list.

```

433 table.insert(
434     lang_hyphenation_list,
435     {
436         [KEY_WORD] = word,
437         [KEY_DIVISION] = pre .. post,
438         [KEY_INDEX] = #lang_hyphenation_list,

```

```

439     }
440   )
441
442 end

```

*(End of definition for check\_line\_hyphenation.)*

`post_linebreak` For every line in the vlist at `vlist_head`, check whether there is a hyphenated word at the end; if so, save the word to `hyphenation_list`.

```

443 local function post_linebreak(vlist_head,groupcode)
444
445   debug('Post-linebreak processing start')
446
447   local line_no = 0
448
449   for item in node.traverse(vlist_head) do
450
451     if item.id == NODE_ID_HLIST then
452       line_no = line_no + 1
453       debug(' Line no.' .. line_no)
454       check_line_hyphenation(item)
455     end
456
457   end
458
459   hlist_hyphenatable_word_list = {}
460
461   debug('Post-linebreak processing end')
462
463   return true
464
465 end

```

*(End of definition for post\_linebreak.)*

## 8.9 Callbacks

Add `pre_linebreak` and `post_linebreak` to the relevant callbacks.

```

466 local LUA_LIST_HYPHEN_PRE_LINEBREAK = 'LUA_LIST_HYPHEN_PRE_LINEBREAK'
467 luatexbase.add_to_callback(
468   'pre_linebreak_filter',
469   pre_linebreak,
470   LUA_LIST_HYPHEN_PRE_LINEBREAK
471 )
472
473 local LUA_LIST_HYPHEN_POST_LINEBREAK = 'LUA_LIST_HYPHEN_POST_LINEBREAK'
474 luatexbase.add_to_callback(
475   'post_linebreak_filter',
476   post_linebreak,
477   LUA_LIST_HYPHEN_POST_LINEBREAK
478 )

```

## 8.10 Language settings

Table mapping language IDs to textual names.

```
478 local language_table = {}
```

Populating `language_table` is done differently for `babel` and `polyglossia`. If `babel` is in use, the  $\text{\LaTeX}$  frontend iterates through `\bbl@languages` and calls `babel_save_language_name`. If `polyglossia` is in use, `language_table` is populated by `polyglossia_get_language_names`, which is called just before the hyphenation lists are written.

`babel_save_language_name` Store the association of a language ID to `babel`'s textual name, if no name has been assigned to that ID already.

```
479 local function babel_save_language_name(lang_id,name)
480
481   if not language_table[lang_id] then
482     language_table[lang_id] = name
483   end
484
485 end
```

*(End of definition for babel\_save\_language\_name.)*

`polyglossia_get_language_names` If `polyglossia` has been loaded, use it to build the table mapping language IDs to textual names.

```
486 local function polyglossia_get_language_names()
487
488   if not polyglossia then
489     return
490   end
491
492   for name,language in pairs(polyglossia.newloader_loaded_languages) do
493     language_table[lang.id(language)] = name
494   end
495
496 end
```

*(End of definition for polyglossia\_get\_language\_names.)*

## 8.11 Processing hyphenation lists

Before writing out hyphenation lists, remove duplicates and/or perform sorting, in accordance with the set options.

### 8.11.1 Comparisons and equality checks

`equal_hyphenation_case_sensitive` Equality check for deduplicating the list of hyphenations case-sensitively.

```
497 local function equal_hyphenation_case_sensitive(a,b)
498   return (
499     a[KEY_WORD] == b[KEY_WORD]
500     and
501     a[KEY_DIVISION] == b[KEY_DIVISION]
502   )
503 end
```

*(End of definition for equal\_hyphenation\_case\_sensitive.)*

equal\_hyphenation\_case\_insensitive Equality check for deduplicating the list of hyphenations case-insensitively.

```

504 local function equal_hyphenation_case_insensitive(a,b)
505   return (
506     unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
507     and
508     unicode.utf8.lower(a[KEY_DIVISION]) == unicode.utf8.lower(b[KEY_DIVISION])
509   )
510 end

```

*(End of definition for equal\_hyphenation\_case\_insensitive.)*

lessthan\_hyphenation\_case\_sensitive Comparison for sorting the list of hyphenations case-sensitively.  
The comparison of index keys ensures that the sorting is stable.

```

511 local function lessthan_hyphenation_case_sensitive(a,b)
512   return (
513     a[KEY_WORD] < b[KEY_WORD]
514     or
515     (
516       a[KEY_WORD] == b[KEY_WORD]
517       and
518       a[KEY_DIVISION] < b[KEY_DIVISION]
519     )
520     or
521     (
522       a[KEY_WORD] == b[KEY_WORD]
523       and
524       a[KEY_DIVISION] == b[KEY_DIVISION]
525       and
526       a[KEY_INDEX] == b[KEY_INDEX]
527     )
528   )
529 end

```

*(End of definition for lessthan\_hyphenation\_case\_sensitive.)*

lessthan\_hyphenation\_case\_insensitive Comparison for sorting the list of hyphenations case-insensitively.  
The comparison of index keys ensures that the sorting is stable.

```

530 local function lessthan_hyphenation_case_insensitive(a,b)
531   return (
532     unicode.utf8.lower(a[KEY_WORD]) < unicode.utf8.lower(b[KEY_WORD])
533     or
534     (
535       unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
536       and
537       unicode.utf8.lower(a[KEY_DIVISION]) < unicode.utf8.lower(b[KEY_DIVISION])
538     )
539     or
540     (
541       unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
542       and
543       unicode.utf8.lower(a[KEY_DIVISION]) < unicode.utf8.lower(b[KEY_DIVISION])
544       and
545       a[KEY_INDEX] == b[KEY_INDEX]
546     )

```

```

547     )
548 end

(End of definition for lessthan_hyphenation_case_insensitive.)

```

### 8.11.2 Sorting

sort\_lang\_hyphenation\_list\_none Sort hyphenation\_list into its original order of appearance.

```

549 local function sort_lang_hyphenation_list_none(hyphenation_list)
550     table.sort(
551         hyphenation_list,
552         function(a,b)
553             return a[KEY_INDEX] < b[KEY_INDEX]
554         end
555     )
556 end

(End of definition for sort_lang_hyphenation_list_none.)

```

sort\_lang\_hyphenation\_list\_case\_sensitive Sort hyphenation\_list case-sensitively.

```

557 local function sort_lang_hyphenation_list_case_sensitive(hyphenation_list)
558     table.sort(
559         hyphenation_list,
560         lessthan_hyphenation_case_sensitive
561     )
562 end

(End of definition for sort_lang_hyphenation_list_case_sensitive.)

```

sort\_lang\_hyphenation\_list\_case\_insensitive Sort hyphenation\_list case-insensitively.

```

563 local function sort_lang_hyphenation_list_case_insensitive(hyphenation_list)
564     table.sort(
565         hyphenation_list,
566         lessthan_hyphenation_case_insensitive
567     )
568 end

(End of definition for sort_lang_hyphenation_list_case_insensitive.)

```

process\_lang\_hyphenation\_list\_sort Select the appropriate function for sorting.

```

569 local sort_lang_hyphenation_list
570 if tex.count['l__lualisthyphen_sort_int'] == 1 then
571     sort_lang_hyphenation_list = sort_lang_hyphenation_list_case_sensitive
572 elseif tex.count['l__lualisthyphen_sort_int'] == 2 then
573     sort_lang_hyphenation_list = sort_lang_hyphenation_list_case_insensitive
574 else
575     sort_lang_hyphenation_list = sort_lang_hyphenation_list_none
576 end

(End of definition for process_lang_hyphenation_list_sort.)

```

### 8.11.3 Deduplication

deduplicate\_lang\_hyphenation\_list\_none

Dummy function; does not deduplicate hyphenation\_list.

```
577 local function deduplicate_lang_hyphenation_list_none(hyphenation_list)
578 end
```

*(End of definition for deduplicate\_lang\_hyphenation\_list\_none.)*

deduplicate\_lang\_hyphenation\_list\_case\_sensitive

Remove duplicates from hyphenation\_list case-sensitively.

```
579 local function deduplicate_lang_hyphenation_list_case_sensitive(hyphenation_list)
580   table.sort(
581     hyphenation_list,
582     lessthan_hyphenation_case_sensitive
583   )
584   list_uniq(
585     hyphenation_list,
586     equal_hyphenation_case_sensitive
587   )
588 end
```

*(End of definition for deduplicate\_lang\_hyphenation\_list\_case\_sensitive.)*

deduplicate\_lang\_hyphenation\_list\_case\_insensitive

Remove duplicates from hyphenation\_list case-insensitively.

```
589 local function deduplicate_lang_hyphenation_list_case_insensitive(hyphenation_list)
590   table.sort(
591     hyphenation_list,
592     lessthan_hyphenation_case_insensitive
593   )
594   list_uniq(
595     hyphenation_list,
596     equal_hyphenation_case_insensitive
597   )
598 end
```

*(End of definition for deduplicate\_lang\_hyphenation\_list\_case\_insensitive.)*

deduplicate\_lang\_hyphenation\_list

Select the appropriate function for whether duplicates should be removed.

```
599 local deduplicate_lang_hyphenation_list
600 if tex.count['l__lualisthyphen_unique_int'] == 1 then
601   deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_case_sensitive
602 elseif tex.count['l__lualisthyphen_unique_int'] == 2 then
603   deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_case_insensitive
604 else
605   deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_none
606 end
```

*(End of definition for deduplicate\_lang\_hyphenation\_list.)*

### 8.11.4 Combined processing

process\_lang\_hyphenation\_list

Remove duplicates and sort hyphenation\_list.

```
607 local function process_lang_hyphenation_list(hyphenation_list)
608   deduplicate_lang_hyphenation_list(hyphenation_list)
609   sort_lang_hyphenation_list(hyphenation_list)
610 end
```

*(End of definition for process\_lang\_hyphenation\_list.)*

## 8.12 Writing

`write_lang_hyphenation_list_standard`

Write out just the hyphenated words.

```
611 local function write_lang_hyphenation_list_standard(f,hyphenation_list)
612
613   for i,v in ipairs(hyphenation_list) do
614
615     if v then
616       f:write(v[KEY_DIVISION] .. '\n')
617     end
618
619   end
620
621 end
```

*(End of definition for write\_lang\_hyphenation\_list\_standard.)*

`write_lang_hyphenation_list_verbose`

Write out all hyphenation information.

```
622 local function write_lang_hyphenation_list_verbose(f,hyphenation_list)
623
624   for i,v in ipairs(hyphenation_list) do
625
626     if v then
627       f:write(v[KEY_WORD] .. ' -> ' .. v[KEY_DIVISION] .. '\n')
628     end
629
630   end
631
632 end
```

*(End of definition for write\_lang\_hyphenation\_list\_verbose.)*

`write_lang_hyphenation_list`

Set `write_lang_hyphenation_list` to be either `write_lang_hyphenation_list_standard` or `write_lang_hyphenation_list_verbose`, depending on the package options.

```
633 local write_lang_hyphenation_list
634 if tex.count['l_lualisthyphen_verbose_int'] == 0 then
635   write_lang_hyphenation_list = write_lang_hyphenation_list_standard
636 else
637   write_lang_hyphenation_list = write_lang_hyphenation_list_verbose
638 end
```

*(End of definition for write\_lang\_hyphenation\_list.)*

`get_hyphenation_file_path`

Get the file to which the list of hyphenated words will be written, based on the given `prefix`, `extension`, numerical `lang_id`, and taking into account any specified output directory for LuaTeX.

```
639 local function get_hyphenation_file_path(prefix,extension,lang_id)
640
641   local lang_name = language_table[lang_id]
642   if not lang_name then
643     lang_name = lang_id
644   end
645
```



```

646 local hyphenation_file_path = prefix .. tostring(lang_name) .. extension
647
648 if not status.output_directory then
649     return hyphenation_file_path
650 end
651
652 if string.sub(status.output_directory,-1,-1) == '/' then
653     hyphenation_file_path = status.output_directory .. hyphenation_file_path
654 else
655     hyphenation_file_path = status.output_directory .. '/' .. hyphenation_file_path
656 end
657
658 return hyphenation_file_path
659
660 end

```

*(End of definition for get\_hyphenation\_file\_path.)*

`process_write_lang_hyphenation_list` Process and write out the `hyphenation_list` (which will be for the language with the numerical `lang_id`) to a file with the given `prefix` and `extension`.

```

661 local function process_write_lang_hyphenation_list(
662     prefix,extension,lang_id,hyphenation_list
663 )
664
665     process_lang_hyphenation_list(hyphenation_list)
666
667     local f = io.open(get_hyphenation_file_path(prefix,extension,lang_id),'w')
668     write_lang_hyphenation_list(f,hyphenation_list)
669     f:close()
670
671 end

```

*(End of definition for process\_write\_lang\_hyphenation\_list.)*

`process_write_hyphenation_lists` If polyglossia is in use, populate `language_table`. Then, for each language, process and write out the hyphenation lists to a file.

```

672 local function process_write_hyphenation_lists(prefix,extension)
673
674     polyglossia_get_language_names()
675
676     for k,v in pairs(hyphenation_table) do
677         process_write_lang_hyphenation_list(prefix,extension,k,v)
678     end
679
680 end

```

*(End of definition for process\_write\_hyphenation\_lists.)*

## 8.13 Export public functions

Finally, make available the functions that will be called from the L<sup>A</sup>T<sub>E</sub>X frontend using `\lua_now:n`.

```

681 return {
682     process_write_hyphenation_lists = process_write_hyphenation_lists,

```

```
683     babel_save_language_name = babel_save_language_name,  
684 }  
685 </lua>
```

# Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

- B**
  - babel commands:
    - babel\_save\_language\_name ..... [479](#)
  - bool commands:
    - \bool\_if:NTF ..... [41](#)
- C**
  - check commands:
    - check\_line\_hyphenation ..... [398](#)
  - cs commands:
    - \cs\_generate\_variant:Nn ..... [81](#)
    - \cs\_if\_exist:NTF ..... [51](#)
    - \cs\_new:Npn ..... [49](#), [61](#), [72](#)
    - \cs\_set\_eq:NN ..... [54](#)
- D**
  - debug (option) ..... [4](#)
  - debug ..... [86](#)
  - deduplicate commands:
    - deduplicate\_lang\_hyphenation\_-list ..... [599](#)
    - deduplicate\_lang\_hyphenation\_-list\_case\_insensitive ..... [589](#)
    - deduplicate\_lang\_hyphenation\_-list\_case\_sensitive ..... [579](#)
    - deduplicate\_lang\_hyphenation\_-list\_none ..... [577](#)
- E**
  - equal commands:
    - equal\_hyphenation\_case\_insensitive ..... [504](#)
    - equal\_hyphenation\_case\_sensitive ..... [497](#)
  - extension (option) ..... [4](#)
- F**
  - \foreignlanguage ..... [4](#)
- G**
  - get commands:
    - get\_disc\_lang ..... [243](#)
    - get\_disc\_word\_start ..... [343](#)
    - get\_first\_glyph\_lang ..... [230](#)
    - get\_hyphenation\_file\_path ..... [639](#)
    - get\_line\_first\_word ..... [373](#)
    - get\_next\_hlist ..... [359](#)
    - get\_node\_text ..... [154](#)
    - get\_nodelist\_text ..... [184](#)
    - get\_used\_disc ..... [329](#)
- group commands:
  - \group\_begin: ..... [53](#)
  - \group\_end: ..... [58](#)
- H**
  - hook commands:
    - \hook\_gput\_code:nnn ..... [46](#), [67](#)
- I**
  - int commands:
    - \int\_new:N ..... [15](#), [21](#), [35](#), [40](#)
    - \int\_set:Nn ..... [18](#), [24](#)
    - \int\_set\_eq:NN ..... [37](#), [42](#)
    - \c\_one\_int ..... [37](#), [42](#)
  - is commands:
    - is\_possible\_word\_node ..... [200](#)
- J**
  - \jobname ..... [3](#), [4](#)
- K**
  - keys commands:
    - \l\_keys\_choice\_int ..... [18](#), [24](#)
    - \keys\_define:nn . [12](#), [16](#), [22](#), [27](#), [31](#), [36](#)
- L**
  - lessthan commands:
    - lessthan\_hyphenation\_case\_-insensitive ..... [530](#)
    - lessthan\_hyphenation\_case\_-sensitive ..... [511](#)
  - list commands:
    - list\_filter ..... [111](#)
    - list\_uniq ..... [128](#)
  - lua commands:
    - \lua\_now:n ..... [25](#), [43](#), [63](#), [74](#)
    - \luaescapestring ..... [76](#), [77](#)
  - lualisthyphen internal commands:
    - \\_\_lualisthyphen\_babel\_save\_-language\_names: ..... [47](#), [49](#), [49](#)
    - \\_\_lualisthyphen\_babel\_save\_-language\_names\_elt:nnnn [56](#), [61](#), [61](#)
    - \l\_\_lualisthyphen\_debug\_int ..... [35](#)
    - \l\_\_lualisthyphen\_file\_extension\_-str ..... [31](#), [70](#)
    - \l\_\_lualisthyphen\_file\_prefix\_-str ..... [27](#), [69](#)
    - \\_\_lualisthyphen\_process\_write\_-hyphenation\_lists:nn [68](#), [72](#), [72](#), [82](#)

|  |                          |  |                                  |
|--|--------------------------|--|----------------------------------|
| <code>\l_lualisthyphen_sort_int</code> . . . . .                   | <a href="#">21</a>       | <code>\ProcessKeyOptions</code> . . . . .                          | <a href="#">39</a>               |
| <code>\l_lualisthyphen_unique_int</code> . . . . .                 | <a href="#">15</a>       | <code>\ProvidesExplPackage</code> . . . . .                        | <a href="#">4</a>                |
| <code>\l_lualisthyphen_verbose_bool</code> . . . . .               | <a href="#">12, 41</a>   |  |                                  |
| <code>\l_lualisthyphen_verbose_int</code> <a href="#">40, 42</a>   |                          | <b>S</b>   |                                  |
| <b>M</b>   |                          | <code>sort (option)</code> . . . . .                               | <a href="#">4</a>                |
| msg commands:  |                          | sort commands:   |                                  |
| <code>\msg_critical:nn</code> . . . . .                            | <a href="#">10</a>       | <code>sort_lang_hyphenation_list_case-insensitive</code> . . . . . | <a href="#">563</a>              |
| <code>\msg_new:nnn</code> . . . . .                                | <a href="#">8</a>        | <code>sort_lang_hyphenation_list_case-sensitive</code> . . . . .   | <a href="#">557</a>              |
|  |                          | <code>sort_lang_hyphenation_list_none</code> <a href="#">549</a>   |                                  |
| <b>N</b>   |                          | str commands:  |                                  |
| <code>\n</code> . . . . .  | <a href="#">616, 627</a> | <code>\str_use:N</code> . . . . .                                  | <a href="#">69, 70</a>           |
| <code>\NeedsTeXFormat</code> . . . . .                             | <a href="#">3</a>        | sys commands:  |                                  |
| <b>O</b>   |                          | <code>\sys_if_engine luatex:TF</code> . . . . .                    | <a href="#">6</a>                |
| options:   |                          | <code>\c_sys_jobname_str</code> . . . . .                          | <a href="#">29</a>               |
| <code>debug</code> . . . . .                                       | <a href="#">4</a>        | <b>T</b>   |                                  |
| <code>extension</code> . . . . .                                   | <a href="#">4</a>        | TeX and L <sup>A</sup> T <sub>E</sub> X 2 <sub>ε</sub> commands:   |                                  |
| <code>prefix</code> . . . . .                                      | <a href="#">4</a>        | <code>\bbl@elt</code> . . . . .                                    | <a href="#">7, 55</a>            |
| <code>sort</code> . . . . .  | <a href="#">4</a>        | <code>\bbl@languages</code> . . . . .                              | <a href="#">7, 8, 20, 51, 57</a> |
| <code>unique</code> . . . . .                                      | <a href="#">3</a>        | trim commands:   |                                  |
| <code>verbose</code> . . . . .                                     | <a href="#">3</a>        | <code>trim_nonletters_both</code> . . . . .                        | <a href="#">213</a>              |
| <b>P</b>   |                          | <code>trim_nonletters_end</code> . . . . .                         | <a href="#">223</a>              |
| polyglossia commands:  |                          | <code>trim_nonletters_start</code> . . . . .                       | <a href="#">218</a>              |
| <code>polyglossia_get_language_names</code> . <a href="#">486</a>  |                          | <b>U</b>   |                                  |
| post commands:   |                          | <code>unique (option)</code> . . . . .                             | <a href="#">3</a>                |
| <code>post_linebreak</code> . . . . .                              | <a href="#">443</a>      | <b>V</b>   |                                  |
| pre commands:  |                          | <code>verbose (option)</code> . . . . .                            | <a href="#">3</a>                |
| <code>pre_linebreak</code> . . . . .                               | <a href="#">283</a>      | <b>W</b>   |                                  |
| <code>prefix (option)</code> . . . . .                             | <a href="#">4</a>        | write commands:  |                                  |
| process commands:  |                          | <code>write_lang_hyphenation_list</code> . . . . .                 | <a href="#">633</a>              |
| <code>process_lang_hyphenation_list</code> . . <a href="#">607</a> |                          | <code>write_lang_hyphenation_list-standard</code> . . . . .        | <a href="#">611</a>              |
| <code>process_lang_hyphenation_list-sort</code> . . . . .          | <a href="#">569</a>      | <code>write_lang_hyphenation_list-verbose</code> . . . . .         | <a href="#">622</a>              |
| <code>process_write_hyphenation_lists</code> <a href="#">672</a>   |                          |  |                                  |
| <code>process_write_lang_hyphenation-list</code> . . . . .         | <a href="#">661</a>      |  |                                  |