lua-list-hyphen — Per-language listing of hyphenated words for LuaLaTeX*

Alan J. Cain†

Released 2026-04-11

Abstract

This LuaLaTeX 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

*This document describes v0.2.45, last revised 2026-04-11.

†a.j.cain (AT) gmail.com

1 Introduction

TEX's algorithm for finding points where a word can be hyphenated is good, but not perfect.[1] The present author writes in British English, where the valid division points can depend on the pronunciation of a word and on its internal structure (and hence its etymology), and where there are thus many situations where TEX's pattern-based approach cannot be expected to work; for example, *geometry* → *geom-etry*, but *geometric* → *geo-met-tric*.[2] For these words, babel's patterns for British English yield *geo-metry* and *geo-met-ric*, while polyglossia's produce *ge-om-e-try* and *ge-o-met-ric*. Even in languages which have more systematic rules for division points can refer to the semantics of compound words, not just to combinations of letters.

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).

[1] 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)

[2] See the *New Oxford Spelling Dictionary*, which is the standard reference 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`[3] 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.[4]]

Licence. lua-list-hyphen is released under the LaTeX Project Public Licence v1.3c or later.[5]

Feature requests and bug reports The development code and issue tracker are hosted at Codeberg.[6]

2 Requirements

lua-list-hyphen requires

(1) LuaLaTeX,

(2) a recent LaTeX 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 LuaLaTeX 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 LuaTeX 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 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

[3] URL: https://codeberg.org/ajcain/hyphenassist.

[4] In particular, *Form & Number: A History of Mathematical Beauty.* URL: https://archive.org/details/cain_formandnumber_ebook_large.

[5] URL: https://www.latex-project.org/lppl.txt

[6] URL: https://codeberg.org/ajcain/lua-list-hyphen

written out. It can be 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 be set to one of the following three values:

none: Hyphenations appear in the same order as the 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 LuaTEX'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 (LaTeX package)

```
⟨*package⟩
⟨@@=lualisthyphen⟩
```

7.1 Initial set-up

Package identification/version information.

```
\NeedsTeXFormat{LaTeX2e}[2020-02-02]
\ProvidesExplPackage{lua-list-hyphen}{2026-04-11}{0.2.45}
  {Listing hyphenated words for LuaLaTeX}
```

Check that LuaTeX is in use.

```
\sys_if_engine_luatex:F
  {
    \msg_new:nnn{ lua-list-hyphen }{ lualatex_required }
      { LuaLaTeX~required.~Package~loading~will~abort. }
    \msg_critical:nn{ lua-list-hyphen }{ lualatex_required }
  }
```

7.2 Options

\l__lualisthyphen_verbose_bool

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

```
\keys_define:nn { lua-list-hyphen }{
  verbose .bool_set:N = \l__lualisthyphen_verbose_bool,
}
```

(*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.

```
\int_new:N\l__lualisthyphen_unique_int
\keys_define:nn { lua-list-hyphen }{
  unique .choices:nn = { none, case, nocase }{
    \int_set:Nn\l__lualisthyphen_unique_int{ \l_keys_choice_int - 1 }
  },
}
```

(*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.

```
\int_new:N\l__lualisthyphen_sort_int
\keys_define:nn { lua-list-hyphen }{
  sort .choices:nn = { none, case, nocase }{
    \int_set:Nn\l__lualisthyphen_sort_int{ \l_keys_choice_int - 1 }
  },
}
```

(*End of definition for* `\l__lualisthyphen_sort_int`.)

\l__lualisthyphen_file_prefix_str

String option for the prefix of file files.

```
\keys_define:nn { lua-list-hyphen }{
  prefix .str_set:N = \l__lualisthyphen_file_prefix_str,
  prefix .initial:e = { \c_sys_jobname_str- },
}
```

(*End of definition for* `\l__lualisthyphen_file_prefix_str`.)

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

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

(*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.

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

(*End of definition for* `\l__lualisthyphen_debug_int`.)

Process package options.

```
\ProcessKeyOptions [ lua-list-hyphen ]
```

Convert boolean options to integers (which can be accessed from Lua).

```
\int_new:N\l__lualisthyphen_verbose_int
\bool_if:NT\l__lualisthyphen_verbose_bool
  { \int_set_eq:NN\l__lualisthyphen_verbose_int\c_one_int }
```

7.3 Lua backend and interface

Load the Lua backend.

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

7.4 Saving language names

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

```
\hook_gput_code:nnn{ enddocument/afterlastpage }{ lua-list-hyphen } {
  \__lualisthyphen_babel_save_language_names:
}
```

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

```
\cs_new:Npn \__lualisthyphen_babel_save_language_names:
  {
    \cs_if_exist:NT\bbl@languages
      {
```

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.

```
        \group_begin:
        \cs_set_eq:NN
          \bbl@elt
```

```
          \__lualisthyphen_babel_save_language_names_elt:nnnn
        \bbl@languages
        \group_end:
      }
  }
```

(*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.

```
\cs_new:Npn \__lualisthyphen_babel_save_language_names_elt:nnnn #1#2#3#4
  {
    \lua_now:n{
      lualisthyphen.babel_save_language_name(#2,'#1')
    }
  }
```

(*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.

```
\hook_gput_code:nnn{ enddocument/info }{ lua-list-hyphen } {
  \__lualisthyphen_process_write_hyphenation_lists:ee
    {\str_use:N\l__lualisthyphen_file_prefix_str}
    {\str_use:N\l__lualisthyphen_file_extension_str}
}
```

sthyphen_process_write_hyphenation_lists:nn Write hyphenations lists to files with prefix given in the first parameter and suffix in the second.

```
\cs_new:Npn \__lualisthyphen_process_write_hyphenation_lists:nn #1#2
  {
    \lua_now:e{
      lualisthyphen.process_write_hyphenation_lists(
        '\luaescapestring{#1}',
        '\luaescapestring{#2}'
      )
    }
  }
\cs_generate_variant:Nn
  \__lualisthyphen_process_write_hyphenation_lists:nn
  { ee }
```

(*End of definition for* `\__lualisthyphen_process_write_hyphenation_lists:nn`.)

⟨/package⟩

8 Implementation (Lua backend)

⟨*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.

```
local function debug(s)
end

if tex.count['l__lualisthyphen_debug_int'] ~= 0 then
  debug = function(s)
    print('lua-list-hyphen DEBUG: ' .. s)
  end
end
```

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

8.2 Table key constants

Keys for tables containing hyphenatable/hyphenated word data.

```
local KEY_WORD = 'word'
local KEY_LANG = 'lang'
local KEY_DIVISION = 'division'
local KEY_INDEX = 'index'
```

8.3 Node ID and subtype constants

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

```
local NODE_ID_HLIST = node.id('hlist')
local NODE_ID_DISC = node.id('disc')
local NODE_ID_GLUE = node.id('glue')
local NODE_ID_KERN = node.id('kern')
local NODE_ID_MARGIN_KERN = node.id('margin_kern')
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 fro the subtype other than searching the `node.subtype('kern')` table.

```
local NODE_KERN_SUBTYPE_FONTKERN
for k,v in pairs(node.subtypes('kern')) do
  if v == 'fontkern' then
    NODE_KERN_SUBTYPE_FONTKERN = k
    break
  end
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.)[7]

```
local function list_filter(t, f)
  local j = 1
  local n = #t

  for i=1,n do
    if (f(t[i])) then
      if (i ~= j) then
```

[7] Code adapted from https://stackoverflow.com/a/53038524.

```
        t[j] = t[i]
        t[i] = nil
      end
      j = j + 1
    else
      t[i] = nil
    end
  end

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.)

```
local function list_uniq(t, f)
  local j = 1
  local n = #t

  for i=2,n do
    if (f(t[i],t[j])) then
      t[i] = nil
    else
      j = i
    end
  end

  list_filter(
    t,
    function(a) return a end
  )
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.

```
local LIGATURE_COMPONENTS = {
  [0xfb00] = {'f','f'},
  [0xfb01] = {'f','i'},
  [0xfb02] = {'f','l'},
  [0xfb03] = {'f','f','i'},
  [0xfb04] = {'f','f','l'},
}
```

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

```
local get_node_text
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`.

```
local function is_possible_word_node(n)

  return (
    n.id == NODE_ID_GLYPH
    or
    n.id == NODE_ID_DISC
    or
    (n.id == NODE_ID_KERN and n.subtype == NODE_KERN_SUBTYPE_FONTKERN)
    or
    n.id == NODE_ID_MARGIN_KERN
  )

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.

```
local function trim_nonletters_both(s)

  return unicode.utf8.match(s,'^%A*(.-)%A*$')

end
```

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

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

```
local function trim_nonletters_start(s)

  return unicode.utf8.match(s,'^%A*(.-)$')

end
```

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

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

```
local function trim_nonletters_end(s)

  return unicode.utf8.match(s,'^(.-)%A*$')

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`.

```
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.

```
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`.)

```
-- local function get_first_glyph_lang(n)

--   item = n
--   while item and is_possible_word_node(item) do
--     if item.id == NODE_ID_GLYPH then
--       return item.lang
--     end
--     item = item.next
--   end

--   return nil

-- 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 LuaLaTeX.)

```
local function get_disc_lang(n)

  -- lang = get_first_glyph_lang(n.replace)
  -- if lang then
  --   print(lang)
  --   return lang
  -- end

  -- lang = get_first_glyph_lang(n.pre)
  -- if lang then
  --   print(lang)
  --   return lang
  -- end

  -- lang = get_first_glyph_lang(n.post)
```

```
  -- if lang then
  --   return lang
  -- end

  local item

  item = n
  while item and is_possible_word_node(item) do
    if item.id == NODE_ID_GLYPH then
      return item.lang
    end
    item = item.prev
  end

  item = n
  while item and is_possible_word_node(item) do
    if item.id == NODE_ID_GLYPH then
      return item.lang
    end
    item = item.next
  end

  return 0

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.

```
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.

```
  local word_start_node = nil
  local hyphenatable_index = nil
  local hyphenatable_count = 0
  local lang = nil

  debug('Pre-linebreak processing start')

  for item in node.traverse(hlist_head) do
```

If `item` is a glyph node, check for a new word start.

```
    if item.id == NODE_ID_GLYPH and not word_start_node then
      word_start_node = item
    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.

```
    if item.id == NODE_ID_DISC then
      if not hyphenatable_index then
        hyphenatable_count = hyphenatable_count + 1
        hyphenatable_index = hyphenatable_count
        lang = get_disc_lang(item)
      end
      node.set_attribute(item,hyphen_attr,hyphenatable_index)
    end
```

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

```
    if not is_possible_word_node(item) then
      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.

```
        local hyphenatable_word = trim_nonletters_both(
          get_nodelist_text(word_start_node,item.prev)
        )
        if hyphenatable_word ~= '' then
          debug(
            '  Hyphenatable word (index ' .. hyphenatable_index .. ') "'
            .. hyphenatable_word .. '"'
          )
          hlist_hyphenatable_word_list[hyphenatable_index] = {
            [KEY_WORD] = hyphenatable_word,
            [KEY_LANG] = lang,
          }
        end
      end
```

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

```
      word_start_node = nil
      hyphenatable_index = nil
    end

  end

  debug('Pre-linebreak processing finish')

  return true
end
```

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

8.8 Post-linebeak 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`.

```
local function get_used_disc(hlist_head)

  local item = node.tail(hlist_head)

  while item and item.id ~= NODE_ID_GLYPH do
    if item.id == NODE_ID_DISC then
      return item
    end
    item = item.prev
  end

  return nil

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.

```
local function get_disc_word_start(hlist_head,n)

  local item = n

  while item do
    local prev = item.prev

    if not (prev and is_possible_word_node(prev)) then
      return item
    end

    item = prev
  end

  return nil
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.

```
local function get_next_hlist(n)

  item = n.next

  while item do
    if item.id == NODE_ID_HLIST then
      return item
    end
    item = item.next
```

```
  end

  return nil

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.

```
local function get_line_first_word(hlist_head)
```

`word_start_node` is either `nil` or the (glyph) node that starts the word.

```
  local word_start_node = nil

  for item in node.traverse(hlist_head) do

    if item.id == NODE_ID_GLYPH then
      if not word_start_node then
        word_start_node = item
      end
    end

    if not is_possible_word_node(item) then
      if word_start_node then
        return get_nodelist_text(word_start_node,item.prev)
      end
    end

  end
```

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

```
  if word_start_node then
    return get_nodelist_text(word_start_node,node.tail(hlist_head))
  else
    return nil
  end
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).

```
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`.

```
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`.)

```
  local last_disc = get_used_disc(hlist.head)
  if not last_disc then
    debug('  No disc node found at end of line')
```

```
    return
  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`.)

```
  local hyphenation_index = node.has_attribute(last_disc,hyphen_attr)
  local t = hlist_hyphenatable_word_list[hyphenation_index]
  if not t then
    debug('  Disc node not associated to a stored word')
    return
  end
  local word = t[KEY_WORD]
  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.

```
  local next_line = get_next_hlist(hlist)

  if not next_line then
    debug('  No following line found (which should not happen)')
    return
  end
```

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

```
  lang_hyphenation_list = hyphenation_table[lang]
  if not lang_hyphenation_list then
    hyphenation_table[lang] = {}
    lang_hyphenation_list = hyphenation_table[lang]
  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.

```
  local pre = get_nodelist_text(get_disc_word_start(hlist.head,last_disc))
  local pre_temp = trim_nonletters_start(pre)
  if pre_temp ~= '' then
    pre = pre_temp
  end
```

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

```
  local post = trim_nonletters_end(get_line_first_word(next_line.head))

  debug(
    '  Hyphenated word found: "' .. word .. '" -> "' .. pre .. '<>' .. post .. '"'
  )
```

Store everything in the language hyphenation list.

```
  table.insert(
    lang_hyphenation_list,
    {
      [KEY_WORD] = word,
      [KEY_DIVISION] = pre .. post,
      [KEY_INDEX] = #lang_hyphenation_list,
```

```
    }
  )

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`.

```
local function post_linebreak(vlist_head,groupcode)

  debug('Post-linebreak processing start')

  local line_no = 0

  for item in node.traverse(vlist_head) do

    if item.id == NODE_ID_HLIST then
      line_no = line_no + 1
      debug('  Line no.' .. line_no)
      check_line_hyphenation(item)
    end

  end

  hlist_hyphenatable_word_list = {}

  debug('Post-linebreak processing end')

  return true

end
```

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

8.9 Callbacks

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

```
local LUA_LIST_HYPHEN_PRE_LINEBREAK = 'LUA_LIST_HYPHEN_PRE_LINEBREAK'
luatexbase.add_to_callback(
  'pre_linebreak_filter',
  pre_linebreak,
  LUA_LIST_HYPHEN_PRE_LINEBREAK
)
local LUA_LIST_HYPHEN_POST_LINEBREAK = 'LUA_LIST_HYPHEN_POST_LINEBREAK'
luatexbase.add_to_callback(
  'post_linebreak_filter',
  post_linebreak,
  LUA_LIST_HYPHEN_POST_LINEBREAK
)
```

8.10 Language settings

Table mapping language IDs to textual names.

```
local language_table = {}
```

Populating `language_table` is done differently for babel and polyglossia. If babel is in use, the 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 texual name, if no name has been assigned to that ID already.

```
local function babel_save_language_name(lang_id,name)

  if not language_table[lang_id] then
    language_table[lang_id] = name
  end

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.

```
local function polyglossia_get_language_names()

  if not polyglossia then
    return
  end

  for name,language in pairs(polyglossia.newloader_loaded_languages) do
    language_table[lang.id(language)] = name
  end

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.

```
local function equal_hyphenation_case_sensitive(a,b)
  return (
    a[KEY_WORD] == b[KEY_WORD]
    and
    a[KEY_DIVISION] == b[KEY_DIVISION]
  )
end
```

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

equal_hyphenation_case_insensitive Equality check for deduplicating the list of hyphenations case-insensitively.

```
local function equal_hyphenation_case_insensitive(a,b)
  return (
    unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
    and
    unicode.utf8.lower(a[KEY_DIVISION]) == unicode.utf8.lower(b[KEY_DIVISION])
  )
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.

```
local function lessthan_hyphenation_case_sensitive(a,b)
  return (
    a[KEY_WORD] < b[KEY_WORD]
    or
    (
      a[KEY_WORD] == b[KEY_WORD]
      and
      a[KEY_DIVISION] < b[KEY_DIVISION]
    )
    or
    (
      a[KEY_WORD] == b[KEY_WORD]
      and
      a[KEY_DIVISION] == b[KEY_DIVISION]
      and
      a[KEY_INDEX] == b[KEY_INDEX]
    )
  )
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.

```
local function lessthan_hyphenation_case_insensitive(a,b)
  return (
    unicode.utf8.lower(a[KEY_WORD]) < unicode.utf8.lower(b[KEY_WORD])
    or
    (
      unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
      and
      unicode.utf8.lower(a[KEY_DIVISION]) < unicode.utf8.lower(b[KEY_DIVISION])
    )
    or
    (
      unicode.utf8.lower(a[KEY_WORD]) == unicode.utf8.lower(b[KEY_WORD])
      and
      unicode.utf8.lower(a[KEY_DIVISION]) < unicode.utf8.lower(b[KEY_DIVISION])
      and
      a[KEY_INDEX] == b[KEY_INDEX]
    )
```

```
  )
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.

```
local function sort_lang_hyphenation_list_none(hyphenation_list)
  table.sort(
    hyphenation_list,
    function(a,b)
      return a[KEY_INDEX] < b[KEY_INDEX]
    end
  )
end
```

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

sort_lang_hyphenation_list_case_sensitive Sort `hyphenation_list` case-sensitively.

```
local function sort_lang_hyphenation_list_case_sensitive(hyphenation_list)
  table.sort(
    hyphenation_list,
    lessthan_hyphenation_case_sensitive
  )
end
```

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

sort_lang_hyphenation_list_case_insensitive Sort `hyphenation_list` case-insensitively.

```
local function sort_lang_hyphenation_list_case_insensitive(hyphenation_list)
  table.sort(
    hyphenation_list,
    lessthan_hyphenation_case_insensitive
  )
end
```

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

process_lang_hyphenation_list_sort Select the appropriate function for sorting.

```
local sort_lang_hyphenation_list
if tex.count['l__lualisthyphen_sort_int'] == 1 then
  sort_lang_hyphenation_list = sort_lang_hyphenation_list_case_sensitive
elseif tex.count['l__lualisthyphen_sort_int'] == 2 then
  sort_lang_hyphenation_list = sort_lang_hyphenation_list_case_insensitive
else
  sort_lang_hyphenation_list = sort_lang_hyphenation_list_none
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`.

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

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

licate_lang_hyphenation_list_case_sensitive Remove duplicates from `hyphenation_list` case-sensitively.

```
local function deduplicate_lang_hyphenation_list_case_sensitive(hyphenation_list)
  table.sort(
    hyphenation_list,
    lessthan_hyphenation_case_sensitive
  )
  list_uniq(
    hyphenation_list,
    equal_hyphenation_case_sensitive
  )
end
```

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

cate_lang_hyphenation_list_case_insensitive Remove duplicates from `hyphenation_list` case-insensitively.

```
local function deduplicate_lang_hyphenation_list_case_insensitive(hyphenation_list)
  table.sort(
    hyphenation_list,
    lessthan_hyphenation_case_insensitive
  )
  list_uniq(
    hyphenation_list,
    equal_hyphenation_case_insensitive
  )
end
```

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

deduplicate_lang_hyphenation_list Select the appropriate function for whether duplicates whould be removed.

```
local deduplicate_lang_hyphenation_list
if tex.count['l__lualisthyphen_unique_int'] == 1 then
  deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_case_sensitive
elseif tex.count['l__lualisthyphen_unique_int'] == 2 then
  deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_case_insensitive
else
  deduplicate_lang_hyphenation_list = deduplicate_lang_hyphenation_list_none
end
```

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

8.11.4 Combined processing

process_lang_hyphenation_list Remove duplicates and sort `hyphenation_list`.

```
local function process_lang_hyphenation_list(hyphenation_list)
  deduplicate_lang_hyphenation_list(hyphenation_list)
  sort_lang_hyphenation_list(hyphenation_list)
end
```

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

8.12 Writing

write_lang_hyphenation_list_standard

Write out just the hyphenated words.

```
local function write_lang_hyphenation_list_standard(f,hyphenation_list)

  for i,v in ipairs(hyphenation_list) do

    if v then
      f:write(v[KEY_DIVISION] .. '\n')
    end

  end

end
```

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

write_lang_hyphenation_list_verbose

Write out all hyphenation information.

```
local function write_lang_hyphenation_list_verbose(f,hyphenation_list)

  for i,v in ipairs(hyphenation_list) do

    if v then
      f:write(v[KEY_WORD] .. ' -> ' .. v[KEY_DIVISION] .. '\n')
    end

  end

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.

```
local write_lang_hyphenation_list
if tex.count['l__lualisthyphen_verbose_int'] == 0 then
  write_lang_hyphenation_list = write_lang_hyphenation_list_standard
else
  write_lang_hyphenation_list = write_lang_hyphenation_list_verbose
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.

```
local function get_hyphenation_file_path(prefix,extension,lang_id)

  local lang_name = language_table[lang_id]
  if not lang_name then
    lang_name = lang_id
  end

```

```
  local hyphenation_file_path = prefix .. tostring(lang_name) .. extension

  if not status.output_directory then
    return hyphenation_file_path
  end

  if string.sub(status.output_directory,-1,-1) == '/' then
    hyphenation_file_path = status.output_directory .. hyphenation_file_path
  else
    hyphenation_file_path = status.output_directory .. '/' .. hyphenation_file_path
  end

  return hyphenation_file_path

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`.

```
local function process_write_lang_hyphenation_list(
    prefix,extension,lang_id,hyphenation_list
)

  process_lang_hyphenation_list(hyphenation_list)

  local f = io.open(get_hyphenation_file_path(prefix,extension,lang_id),'w')
  write_lang_hyphenation_list(f,hyphenation_list)
  f:close()

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.

```
local function process_write_hyphenation_lists(prefix,extension)

  polyglossia_get_language_names()

  for k,v in pairs(hyphenation_table) do
    process_write_lang_hyphenation_list(prefix,extension,k,v)
  end

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 LaTeX frontend using `\lua_now:n`.

```
return {
  process_write_hyphenation_lists = process_write_hyphenation_lists,
```

```
  babel_save_language_name = babel_save_language_name,
}
```
⟨/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.

M

N

O

P

S

T

U

V

W