

Highlighting Typographical Flaws with LuaLaTeX

Daniel Flipo

daniel.flipo@free.fr

1 What is it about?

The file `lua-typo.sty`¹, is meant for careful writers and proofreaders who do not feel totally satisfied with LaTeX output, the most frequent issues being widows and orphans, hyphenated words split across two pages, consecutive lines ending with hyphens, paragraphs ending on too short or nearly full lines, homeoarchy, etc.

This package, which works with LuaLaTeX only, *does not try to correct anything* but just highlights potential issues (the offending lines or end of lines are printed in colour) and provides at the end of the `.log` file a summary of pages to be checked and manually corrected if possible. My understanding is that automatic correction often introduces new issues (underflow/overflow lines) when fixing one of the flaws mentioned above, human correction providing much better results. For completeness, overflow and underfull lines are also coloured (in blue by default) and mentioned in the summary provided at the end of the `.log` file.

The TeX commands `\spaceskip` and `\xspaceskip` which alter the inter-word spacing (locally when used in a group) should be considered: slightly enlarging the inter-word space may be sufficient to make a paragraph's last line acceptable when it was originally too short or add a line to a paragraph when its last line was nearly full, thus possibly removing an orphan. Conversely, slightly reducing it may remove a paragraph's last line (when it was short) and get rid of a widow on top of next page.

I suggest to add a call `\usepackage[All]{lua-typo}` to the preamble of a document which is “nearly finished” *and to remove it* once all possible corrections have been made: if some flaws remain, getting them printed in colour in the final document would be a shame!

Starting with version 0.50 a recent LaTeX kernel (dated 2021/06/01) is required. Users running an older kernel will get a warning and an error message “`Unable to register callback`”; for them, a “rollback” version of `lua-typo` is provided, it can be loaded this way: `\usepackage[All]{lua-typo}[v0.4]`.

See files `demo.tex` and `demo.pdf` for a short example (in French).

I am very grateful to Jacques André and Thomas Savary, who kindly tested my beta versions, providing much valuable feedback and suggesting many improvements for the first released version. Special thanks to both of them!

2 Usage

The easiest way to trigger all checks performed by `lua-typo` is:

```
\usepackage[All]{lua-typo}
```

It is possible to enable or disable some checks through boolean options passed to `lua-typo`; you may want to perform all checks except a few, then `lua-typo` should be

¹The file described in this section has version number v0.60 and was last revised on 2023-02-04.

loaded this way:

```
\usepackage[All, <OptX>=false, <OptY>=false]{lua-typo}
```

or to enable just a few checks, then do it this way:

```
\usepackage[<OptX>, <OptY>, <OptZ>]{lua-typo}
```

Here is the full list of possible checks (name and purpose):

Name	Glitch to highlight
All	Turns all options to true
BackParindent	paragraph's last line <i>nearly</i> full?
ShortLines	paragraph's last line too short?
ShortPages	nearly empty page (just a few lines)?
OverfullLines	overfull lines?
UnderfullLines	underfull lines?
Widows	widows (top of page)?
Orphans	orphans (bottom of page)?
EOPHyphens	hyphenated word split across two pages?
RepeatedHyphens	too many consecutive hyphens?
ParLastHyphen	paragraph's last full line hyphenated?
EOLShortWords	short words (1 or 2 chars) at end of line?
FirstWordMatch	same (part of) word starting two consecutive lines?
LastWordMatch	same (part of) word ending two consecutive lines?
FootnoteSplit	footnotes spread over two pages or more?

For example, if you want **lua-typo** to only warn about overfull and underfull lines, you can load **lua-typo** like this:

```
\usepackage[OverfullLines, UnderfullLines]{lua-typo}
```

If you want everything to be checked except paragraphs ending on a short line try:

```
\usepackage[All, ShortLines=false]{lua-typo}
```

please note that **All** has to be the first one, as options are taken into account as they are read *i.e.* from left to right.

The list of all available options is printed to the **.log** file when option **ShowOptions** is passed to **lua-typo**, this option provides an easy way to get their names without having to look into the documentation.

With option **None**, **lua-typo** *does absolutely nothing*, all checks are disabled as the main function is not added to any LuaTeX callback. It not quite equivalent to commenting out the `\usepackage{lua-typo}` line though, as user defined commands related to **lua-typo** are still defined and will not print any error message.

Please be aware of the following features:

FirstWordMatch: the first word of consecutive list items is not highlighted, as these repetitions result of the author's choice.

LastWordMatch: a paragraphs' last word ending "too far" from the right margin (*i.e.* more than `\luatypoBackPI` –default=1em– away) is never highlighted even if it matches the one on the previous line. Similarly, if it matches the one on the next line, the latter will not be highlighted either.

ShortPages: if a page is considered too short, its last line only is highlighted, not the whole page.

RepeatedHyphens: ditto, when the number of consecutives hyphenated lines is too high, only the hyphenated words in excess (the last ones) are highlighted.

Starting with version 0.50, the footnotes' contents are checked as well by **lua-typo** and footnotes too long to end on the current page are mentionned as a flaw (option **FootnoteSplit**). The list of all flaws found is written to a specific log-file whose name is suffixed by **.typo**.

3 Customisation

Some of the checks mentionned above require tuning, for instance, when is a last paragraph's length called too short? how many hyphens ending consecutive lines are acceptable? **lua-typo** provides user customisable parameters to set what is regarded as acceptable or not.

A default configuration file **lua-typo.cfg** is provided with all parameters set to their defaults; it is located under the **TEXMFDIST** directory. It is up to the users to copy this file into their working directory (or **TEXMFHOME** or **TEXMFLOCAL**) and tune the defaults according to their own taste.

It is also possible to provide defaults directly in the document's preamble (this overwrites the corresponding settings done in the configuration file found on TeX's search path: current directory, then **TEXMFHOME**, **TEXMFLOCAL** and finally **TEXMFDIST**).

Here are the parameters names (all prefixed by **luatypo** in order to avoid conflicts with other packages) and their default values:

BackParindent : paragraphs' last line should either touch the right margin (actually end at less than **\luatypoBackFuzz**, default **2pt**, from it) or leave at least **\luatypoBackPI**, default **1em**, between its end and the right margin.

ShortLines: **\luatypoLLminWD=2\parindent**² sets the minimum acceptable length for paragraphs' last lines.

ShortPages: **\luatypoPageMin=5** sets the minimum acceptable number of lines on a page (chapters' last page for instance). Actually, the last line's vertical position on the page is taken into account so that f.i. title pages or pages ending on a picture are not pointed out.

RepeatedHyphens: **\luatypoHyphMax=2** sets the maximum acceptable number of consecutive hyphenated lines.

UnderfullLines: **\luatypoStretchMax=200** sets the maximum acceptable percentage of stretch acceptable before a line is tagged by **lua-typo** as underfull; it must be an integer over 100, 100 means that the slightest stretch exceeding the font tolerance (**\fontdimen3**) will be warned about (be prepared for a lot of "underfull lines" with this setting), the default value 200 is just below what triggers TeX's "Underfull hbox" message (when **\tolerance=200** and **\hbadness=1000**).

First/LastWordMatch: **\luatypoMinFull=3** and **\luatypoMinPart=4** set the minimum number of characters required for a match to be pointed out. With this setting (3 and 4), two occurrences of the word 'out' at the beginning or end of two

²Or **20pt** if **\parindent=0pt**.

consecutive lines will be highlighted (three chars, ‘in’ wouldn’t match), whereas a line ending with “full” or “overfull” followed by one ending with “underfull” will match (four chars): the second occurrence of “full” or “erfull” will be highlighted.

EOLShortWords: this check deals with lines ending with very short words (one or two characters), not all of them but a user selected list depending on the current language.

```
\luatypoOneChar{<language>}{'<list of words>'}
\luatypoTwoChars{<language>}{'<list of words>'}
```

Currently, defaults (commented out) are suggested for the French language only:

```
\luatypoOneChar{french}{'À à Ô'}
\luatypoTwoChars{french}{'Je Tu Il On'}
```

Feel free to customise these lists for French or to add your own shorts words for other languages but remember that a) the first argument (language name) *must be known by babel*, so if you add `\luatypoOneChar` or `\luatypoTwoChars` commands, please make sure that `lua-typo` is loaded *after babel*; b) the second argument *must be a string* (i.e. surrounded by single or double ASCII quotes) made of your words separated by spaces.

It is possible to define a specific colour for each typographic flaws that `lua-typo` deals with. Currently, only five colours are used in `lua-typo.cfg`:

```
% \definecolor{mygrey}{gray}{0.6}
% \definecolor{myred}{rgb}{1,0.55,0}
% \luatypoSetColor0{red}      % Paragraph last full line hyphenated
% \luatypoSetColor1{red}      % Page last word hyphenated
% \luatypoSetColor2{red}      % Hyphens on consecutive lines
% \luatypoSetColor3{red}      % Short word at end of line
% \luatypoSetColor4{cyan}     % Widow
% \luatypoSetColor5{cyan}     % Orphan
% \luatypoSetColor6{cyan}     % Paragraph ending on a short line
% \luatypoSetColor7{blue}     % Overfull lines
% \luatypoSetColor8{blue}     % Underfull lines
% \luatypoSetColor9{red}      % Nearly empty page (a few lines)
% \luatypoSetColor{10}{myred} % First word matches
% \luatypoSetColor{11}{myred} % Last word matches
% \luatypoSetColor{12}{mygrey}% paragraph's last line nearly full
% \luatypoSetColor{13}{cyan}  % footnotes spread over two pages
%
```

`lua-typo` loads the `color` package from the LaTeX graphic bundle. Only named colours can be used by `lua-typo`, so you can either use the `\definecolor` from `color` package to define yours (as done in the config file for ‘mygrey’) or load the `xcolor` package which provides a bunch of named colours.

4 TeXnical details

Starting with version 0.50, this package uses the rollback mechanism to provide easier backward compatibility. Rollback version 0.40 is provided for users who would have a LaTeX kernel older than 2021/06/01.

```
1 \ifdefined\DeclareRelease
2   \DeclareRelease{v0.4}{2021-01-01}{lua-typo-2021-04-18.sty}
3   \DeclareCurrentRelease{}{2023-02-04}
4 \else
5   \PackageWarning{lua-typo}{Your LaTeX kernel is too old to provide
6     access\MessageBreak to former versions of the lua-typo package.%
7     \MessageBreak Anyway, lua-typo requires a LaTeX kernel dated%
8     \MessageBreak 2020-01-01 or newer; reported}
9 \fi
10 \NeedsTeXFormat{LaTeX2e}[2021/06/01]
```

This package only runs with LuaLaTeX and requires packages `luatexbase`, `luacode`, `luacolor` and `atveryend`.

```
11 \ifdefined\directlua
12   \RequirePackage{luatexbase,luacode,luacolor}
13   \RequirePackage{kvoptions,atveryend}
14 \else
15   \PackageError{This package is meant for LuaTeX only! Aborting}
16     {No more information available, sorry!}
17 \fi
```

Let's define the necessary internal counters, dimens, token registers and commands...

```
18 \newdimen\luatypoLLminWD
19 \newdimen\luatypoBackPI
20 \newdimen\luatypoBackFuzz
21 \newcount\luatypoStretchMax
22 \newcount\luatypoHyphMax
23 \newcount\luatypoPageMin
24 \newcount\luatypoMinFull
25 \newcount\luatypoMinPart
26 \newcount\luatypo@LANGno
27 \newcount\luatypo@options
28 \newtoks\luatypo@single
29 \newtoks\luatypo@double
```

... and define a global table for this package.

```
30 \begin{luacode}
31   luatypo = { }
32 \end{luacode}
```

Set up `kvoptions` initializations.

```
33 \SetupKeyvalOptions{
34   family=luatypo,
35   prefix=LT@,
36 }
37 \DeclareBoolOption[false]{ShowOptions}
```

```

38 \DeclareBoolOption[false]{None}
39 \DeclareBoolOption[false]{All}
40 \DeclareBoolOption[false]{BackParindent}
41 \DeclareBoolOption[false]{ShortLines}
42 \DeclareBoolOption[false]{ShortPages}
43 \DeclareBoolOption[false]{OverfullLines}
44 \DeclareBoolOption[false]{UnderfullLines}
45 \DeclareBoolOption[false]{Widows}
46 \DeclareBoolOption[false]{Orphans}
47 \DeclareBoolOption[false]{EOPHyphens}
48 \DeclareBoolOption[false]{RepeatedHyphens}
49 \DeclareBoolOption[false]{ParLastHyphen}
50 \DeclareBoolOption[false]{EOLShortWords}
51 \DeclareBoolOption[false]{FirstWordMatch}
52 \DeclareBoolOption[false]{LastWordMatch}
53 \DeclareBoolOption[false]{FootnoteSplit}

```

Option `All` resets all booleans relative to specific typographic checks to `true`.

```

54 \AddToKeyvalOption{luatypo}{All}{%
55   \LT@ShortLinestrue   \LT@ShortPagestrue
56   \LT@OverfullLinestrue \LT@UnderfullLinestrue
57   \LT@Widowstrue       \LT@Orphanstrue
58   \LT@EOPHyphenstrue   \LT@RepeatedHyphenstrue
59   \LT@ParLastHyphenstrue \LT@EOLShortWordstrue
60   \LT@FirstWordMatchtrue \LT@LastWordMatchtrue
61   \LT@BackParindenttrue  \LT@FootnoteSplittrue
62 }
63 \ProcessKeyvalOptions{luatypo}

```

Forward these options to the `luatypo` global table. Wait until the config file `lua-typo.cfg` has been read in order to give it a chance of overruling the boolean options. This enables the user to permanently change the defaults.

```

64 \AtEndOfPackage{%
65   \ifLT@None
66     \directlua{ luatypo.None = true }%
67   \else
68     \directlua{ luatypo.None = false }%
69   \fi
70   \ifLT@BackParindent
71     \advance\luatypo@options by 1
72     \directlua{ luatypo.BackParindent = true }%
73   \else
74     \directlua{ luatypo.BackParindent = false }%
75   \fi
76   \ifLT@ShortLines
77     \advance\luatypo@options by 1
78     \directlua{ luatypo.ShortLines = true }%
79   \else
80     \directlua{ luatypo.ShortLines = false }%
81   \fi
82   \ifLT@ShortPages
83     \advance\luatypo@options by 1
84     \directlua{ luatypo.ShortPages = true }%

```

```

85 \else
86   \directlua{ luatypo.ShortPages = false }%
87 \fi
88 \ifLT@OverfullLines
89   \advance\luatypo@options by 1
90   \directlua{ luatypo.OverfullLines = true }%
91 \else
92   \directlua{ luatypo.OverfullLines = false }%
93 \fi
94 \ifLT@UnderfullLines
95   \advance\luatypo@options by 1
96   \directlua{ luatypo.UnderfullLines = true }%
97 \else
98   \directlua{ luatypo.UnderfullLines = false }%
99 \fi
100 \ifLT@Widows
101   \advance\luatypo@options by 1
102   \directlua{ luatypo.Widows = true }%
103 \else
104   \directlua{ luatypo.Widows = false }%
105 \fi
106 \ifLT@Orphans
107   \advance\luatypo@options by 1
108   \directlua{ luatypo.Orphans = true }%
109 \else
110   \directlua{ luatypo.Orphans = false }%
111 \fi
112 \ifLT@EOPHyphens
113   \advance\luatypo@options by 1
114   \directlua{ luatypo.EOPHyphens = true }%
115 \else
116   \directlua{ luatypo.EOPHyphens = false }%
117 \fi
118 \ifLT@RepeatedHyphens
119   \advance\luatypo@options by 1
120   \directlua{ luatypo.RepeatedHyphens = true }%
121 \else
122   \directlua{ luatypo.RepeatedHyphens = false }%
123 \fi
124 \ifLT@ParLastHyphen
125   \advance\luatypo@options by 1
126   \directlua{ luatypo.ParLastHyphen = true }%
127 \else
128   \directlua{ luatypo.ParLastHyphen = false }%
129 \fi
130 \ifLT@EOLShortWords
131   \advance\luatypo@options by 1
132   \directlua{ luatypo.EOLShortWords = true }%
133 \else
134   \directlua{ luatypo.EOLShortWords = false }%
135 \fi
136 \ifLT@FirstWordMatch
137   \advance\luatypo@options by 1
138   \directlua{ luatypo.FirstWordMatch = true }%

```

```

139 \else
140   \directlua{ luatypo.FirstWordMatch = false }%
141 \fi
142 \ifLT@LastWordMatch
143   \advance\luatypo@options by 1
144   \directlua{ luatypo.LastWordMatch = true }%
145 \else
146   \directlua{ luatypo.LastWordMatch = false }%
147 \fi
148 \ifLT@FootnoteSplit
149   \advance\luatypo@options by 1
150   \directlua{ luatypo.FootnoteSplit = true }%
151 \else
152   \directlua{ luatypo.FootnoteSplit = false }%
153 \fi
154 }

```

ShowOptions is specific:

```

155 \ifLT@showOptions
156   \GenericWarning{* }{%
157     *** List of possible options for lua-typo ***\MessageBreak
158     [Default values between brackets]%
159     \MessageBreak
160     ShowOptions      [false]\MessageBreak
161     None              [false]\MessageBreak
162     BackParindent    [false]\MessageBreak
163     ShortLines        [false]\MessageBreak
164     ShortPages        [false]\MessageBreak
165     OverfullLines     [false]\MessageBreak
166     UnderfullLines    [false]\MessageBreak
167     Widows            [false]\MessageBreak
168     Orphans           [false]\MessageBreak
169     EOPHyphens        [false]\MessageBreak
170     RepeatedHyphens   [false]\MessageBreak
171     ParLastHyphen     [false]\MessageBreak
172     EOLShortWords     [false]\MessageBreak
173     FirstWordMatch    [false]\MessageBreak
174     LastWordMatch     [false]\MessageBreak
175     FootnoteSplit     [false]\MessageBreak
176     \MessageBreak
177     %*****%
178     \MessageBreak Lua-typo [ShowOptions]
179   }%
180 \fi

```

Some default values which can be customised in the preamble are forwarded to Lua AtBeginDocument.

```

181 \AtBeginDocument{%
182   \directlua{
183     luatypo.HYPHmax = tex.count.luatypoHyphMax
184     luatypo.PAGEmin = tex.count.luatypoPageMin
185     luatypo.Stretch = tex.count.luatypoStretchMax
186     luatypo.MinFull = tex.count.luatypoMinFull

```



```

187     luatypo.MinPart = tex.count.luatypepoMinPart
188     luatypo.LLminWD = tex.dimen.luatypepoLLminWD
189     luatypo.BackPI = tex.dimen.luatypepoBackPI
190     luatypo.BackFuzz = tex.dimen.luatypepoBackFuzz
191 }%
192 }

```

Print the summary of offending pages—if any— at the (very) end of document and write the report file on disc, unless option **None** has been selected.

```

193 \AtVeryEndDocument{%
194 \ifnum\luatypepo@options = 0 \LT@Nonetrue \fi
195 \ifLT@None
196   \directlua{
197     texio.write_nl(' ')
198     texio.write_nl('*****')
199     texio.write_nl('*** lua-typo loaded with NO option:')
200     texio.write_nl('*** NO CHECK PERFORMED! ***')
201     texio.write_nl('*****')
202     texio.write_nl(' ')
203   }%
204 \else
205   \directlua{
206     texio.write_nl(' ')
207     texio.write_nl('*****')
208     if luatypo.pagelist == " " then
209       texio.write_nl('*** lua-typo: No Typo Flaws found.')
210     else
211       texio.write_nl('*** lua-typo: WARNING *****')
212       texio.write_nl('The following pages need attention:')
213       texio.write(luatypo.pagelist)
214     end
215     texio.write_nl('*****')
216     texio.write_nl(' ')
217     local fileout= tex.jobname .. ".typo"
218     local out=io.open(fileout,"w+")
219     out:write(luatypo.buffer)
220     io.close(out)
221   }%
222 \fi}

```

\luatypepoOneChar These commands set which short words should be avoided at end of lines. The first argument is a language name, say **french**, which is turned into a command **\l@french** expanding to a number known by luatex, otherwise an error message occurs. The UTF8 string entered as second argument has to be converted into the font internal coding.

```

223 \newcommand*{\luatypepoOneChar}[2]{%
224   \def\luatypepo@LANG{#1}\luatypepo@single={#2}%
225   \ifcsname l@\luatypepo@LANG\endcsname
226     \luatypepo@LANGno=\the\csname l@\luatypepo@LANG\endcsname \relax
227   \directlua{
228     local langno = \the\luatypepo@LANGno
229     local string = \the\luatypepo@single

```

```

230     luatypo.single[langno] = " "
231     for p, c in utf8.codes(string) do
232         local s = string.char(c)
233         luatypo.single[langno] = luatypo.single[langno] .. s
234     end
235 <dbg>     texio.write_nl("SINGLE=" .. luatypo.single[langno])
236 <dbg>     texio.write_nl(' ')
237 }%
238 \else
239 \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
240 \MessageBreak \protect\luatypoOneChar\space command ignored}%
241 \fi}
242 \newcommand*{\luatypoTwoChars}[2]{%
243 \def\luatypo@LANG{#1}\luatypo@double={#2}%
244 \ifcsname l@\luatypo@LANG\endcsname
245 \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
246 \directlua{
247     local langno = \the\luatypo@LANGno
248     local string = \the\luatypo@double
249     luatypo.double[langno] = " "
250     for p, c in utf8.codes(string) do
251         local s = string.char(c)
252         luatypo.double[langno] = luatypo.double[langno] .. s
253     end
254 <dbg>     texio.write_nl("DOUBLE=" .. luatypo.double[langno])
255 <dbg>     texio.write_nl(' ')
256 }%
257 \else
258 \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
259 \MessageBreak \protect\luatypoTwoChars\space command ignored}%
260 \fi}

```

`\luatypoSetColor` This is a user-level command to customise the colours highlighting the fourteen types of possible typographic flaws. The first argument is a number (flaw type), the second the named colour associated to it. The colour support is based on the `luacolor` package (color attributes).

```

261 \newcommand*{\luatypoSetColor}[2]{%
262 \begingroup
263 \color{#2}%
264 \directlua{luatypo.colortbl[#1]=\the\LuaCol@Attribute}%
265 \endgroup
266 }

```

The Lua code now, initialisations.

```

267 \begin{luacode}
268 luatypo.single = { }
269 luatypo.double = { }
270 luatypo.colortbl = { }
271 luatypo.pagelist = " "
272 luatypo.buffer = "List of typographic flaws found for "

```

```

273             .. tex.jobname .. ".tex:\string\n\string\n"
274
275 local char_to_discard = { }
276 char_to_discard[string.byte(",")] = true
277 char_to_discard[string.byte(".")] = true
278 char_to_discard[string.byte("!")] = true
279 char_to_discard[string.byte("?")] = true
280 char_to_discard[string.byte(":")] = true
281 char_to_discard[string.byte(";")] = true
282 char_to_discard[string.byte("-")] = true
283
284 local split_lig = { }
285 split_lig[0xFB00] = "ff"
286 split_lig[0xFB01] = "fi"
287 split_lig[0xFB02] = "fl"
288 split_lig[0xFB03] = "ffi"
289 split_lig[0xFB04] = "ffl"
290 split_lig[0xFB05] = "st"
291 split_lig[0xFB06] = "st"
292
293 local DISC = node.id("disc")
294 local GLYPH = node.id("glyph")
295 local GLUE = node.id("glue")
296 local KERN = node.id("kern")
297 local RULE = node.id("rule")
298 local HLIST = node.id("hlist")
299 local VLIST = node.id("vlist")
300 local LPAR = node.id("local_par")
301 local MKERN = node.id("margin_kern")
302 local PENALTY = node.id("penalty")
303 local WHATSIT = node.id("whatsit")

```

Glue subtypes:

```

304 local USRSKIP = 0
305 local PARSKIP = 3
306 local LFTSKIP = 8
307 local RGTSKIP = 9
308 local TOPSKIP = 10
309 local PARFILL = 15

```

Hlist subtypes:

```

310 local LINE = 1
311 local BOX = 2
312 local INDENT = 3
313 local ALIGN = 4
314 local EQN = 6

```

Penalty subtypes:

```

315 local USER = 0
316 local HYPH = 0x2D

```

Glyph subtypes:

```

317 local LIGA = 0x102

```

`parline` (current paragraph) must not be reset on every new page!

```
318 local parline = 0
319
320 local dimensions = node.dimensions
321 local rangedimensions = node.rangedimensions
322 local effective_glue = node.effective_glue
323 local set_attribute = node.set_attribute
324 local slide = node.slide
325 local traverse = node.traverse
326 local traverse_id = node.traverse_id
327 local has_field = node.has_field
328 local uses_font = node.uses_font
329 local is_glyph = node.is_glyph
330
```

This auxillary function colours glyphs and discretionaries. It requires two arguments: a node and a (named) colour.

```
331 local color_node = function (node, color)
332   local attr = oberdiek.luacolor.getattribute()
333   if node and node.id == DISC then
334     local pre = node.pre
335     local post = node.post
336     local repl = node.replace
337     if pre then
338       set_attribute(pre,attr,color)
339 <dbg>      texio.write_nl('PRE=' .. tostring(pre.char))
340     end
341     if post then
342       set_attribute(post,attr,color)
343 <dbg>   if pre then
344 <dbg>     texio.write(' POST=' .. tostring(post.char))
345 <dbg>   else
346 <dbg>     texio.write_nl('POST=' .. tostring(post.char))
347 <dbg>   end
348     end
349     if repl then
350       set_attribute(repl,attr,color)
351 <dbg>   if pre or post then
352 <dbg>     texio.write(' REPL=' .. tostring(repl.char))
353 <dbg>   else
354 <dbg>     texio.write_nl('REPL=' .. tostring(repl.char))
355 <dbg>   end
356     end
357 <dbg>   if pre or post or repl then
358 <dbg>     texio.write_nl(' ')
359 <dbg>   end
360   elseif node then
361     set_attribute(node,attr,color)
362   end
363 end
```

This auxillary function colours a whole line. It requires two arguments: a line's node and a (named) colour.

Digging into nested hlists and vlists is needed f.i. to colour aligned equations.

```

364 local color_line = function (head, color)
365   local first = head.head
366   for n in traverse(first) do
367     if n.id == HLIST or n.id == VLIST then
368       local ff = n.head
369       for nn in traverse(ff) do
370         if nn.id == HLIST or nn.id == VLIST then
371           local f3 = nn.head
372           for n3 in traverse(f3) do
373             if n3.id == HLIST or n3.id == VLIST then
374               local f4 = n3.head
375               for n4 in traverse(f4) do
376                 if n4.id == HLIST or n4.id == VLIST then
377                   local f5 = n4.head
378                   for n5 in traverse(f5) do
379                     if n5.id == HLIST or n5.id == VLIST then
380                       local f6 = n5.head
381                       for n6 in traverse(f6) do
382                         color_node(n6, color)
383                       end
384                     else
385                       color_node(n5, color)
386                     end
387                   end
388                 else
389                   color_node(n4, color)
390                 end
391               end
392             else
393               color_node(n3, color)
394             end
395           end
396         else
397           color_node(nn, color)
398         end
399       end
400     else
401       color_node(n, color)
402     end
403   end
404 end

```

This function appends a line to a buffer which will be written to file ‘\jobname.typo’; it takes four arguments: a string, two numbers (which can be NIL) and a flag.

```

405 log_flaw= function (msg, line, colno, footnote)
406   local pageno = tex.getcount("c@page")
407   local prt = "p. " .. pageno
408   if colno then
409     prt = prt .. ", col." .. colno
410   end
411   if line then
412     local l = string.format("%2d, ", line)

```

```

413     if footnote then
414         prt = prt .. ", (ftn.) line " .. l
415     else
416         prt = prt .. ", line " .. l
417     end
418 end
419 prt = prt .. msg
420 luatypo.buffer = luatypo.buffer .. prt .. "\string\n"
421 end

```

The next three functions deal with “homeoarchy”, *i.e.* lines beginning or ending with the same (part of) word. While comparing two words, the only significant nodes are glyphs and ligatures, dicretionnaires other than ligatures, kerns (letterspacing) should be discarded. For each word to be compared we build a “signature” made of glyphs and split ligatures.

The first function adds a node to a signature of type string. It returns the augmented string and its length. The last argument is a boolean needed when building a signature backwards (see `check_last_word`).

```

422 local signature = function (node, string, swap)
423     local n = node
424     local str = string
425     if n and n.id == GLYPH then
426         local b, id = is_glyph(n)
427         if b and not char_to_discard[b] then

```

Punctuation has to be discarded; the French apostrophe (right quote U+2019) has a char code “out of range”, we replace it with U+0027; Other glyphs should have char codes less than 0x100 (or 0x180?) or be ligatures... standard ones (U+FB00 to U+FB06) are converted using table `split_lig`.

```

428         if b == 0x2019 then b = 0x0027 end
429         if b < 0x100 then
430             str = str .. string.char(b)
431         elseif split_lig[b] then
432             local c = split_lig[b]
433             if swap then
434                 c = string.reverse(c)
435             end
436             str = str .. c

```

Experimental: store other ligatures as the last two digits of their decimal code...

```

437         elseif n.subtype == LIGA and b > 0xE000 then
438             local c = string.sub(b,-2)
439             if swap then
440                 c = string.reverse(c)
441             end
442             str = str .. c
443         end
444     end
445     elseif n and n.id == DISC then

```

Ligatures are split into `pre` and `post` and both parts are stored. In case of *ffl*, *ffi*, the post part is also a ligature...

```

446     local pre = n.pre
447     local post = n.post
448     local c1 = ""
449     local c2 = ""
450     if pre and pre.char and pre.char ~= HYPH and pre.char < 0x100 then
451         c1 = string.char(pre.char)
452     end
453     if post and post.char then
454         if post.char < 0x100 then
455             c2 = string.char(post.char)
456         elseif split_lig[post.char] then
457             c2 = split_lig[post.char]
458             if swap then
459                 c2 = string.reverse(c2)
460             end
461         end
462     end
463     if swap then
464         str = str .. c2 .. c1
465     else
466         str = str .. c1 .. c2
467     end
468 end

```

The returned length is the number of *letters*.

```

469     local len = string.len(str)
470     if string.find(str, "_") then
471         len = len - 1
472     end
473     return len, str
474 end

```

This auxillary function looks for consecutive lines ending with the same letters. It requires four arguments: a string (previous line's signature), a node (the last one on the current line), a line number and a boolean to cancel checking in some cases (end of paragraphs). It prints the matching part at end of line with the supplied colour and returns the current line's last word and a boolean (match).

```

475 local check_last_word = function (old, node, line, flag)
476     local COLOR = luatypo.colortbl[11]
477     local match = false
478     local new = ""
479     local maxlen = 0
480     if flag and node then
481         local swap = true
482         local box, go

```

Step back to the last glyph or discretionary.

```

483     local lastn = node
484     while lastn and lastn.id ~= GLYPH and lastn.id ~= DISC and
485         lastn.id ~= HLIST do
486         lastn = lastn.prev
487     end

```

A signature is built from the last two words on the current line.

```
488     local n = lastn
489     if n and n.id == HLIST then
490         box = n
491         prev = n.prev
492         lastn = slide(n.head)
493         n = lastn
494     end
495     while n and n.id ~= GLUE do
496         maxlen, new = signature (n, new, swap)
497         n = n.prev
498     end
499     if n and n.id == GLUE then
500         new = new .. "_"
501         go = true
502     elseif box and not n then
503         local p = box.prev
504         if p.id == GLUE then
505             new = new .. "_"
506             n = p
507         else
508             n = box
509         end
510         go = true
511     end
512     if go then
513         repeat
514             n = n.prev
515             maxlen, new = signature (n, new, swap)
516         until not n or n.id == GLUE
517     end
518     new = string.reverse(new)
519 <dbg>     texio.write_nl('EOLsigold=' .. old)
520 <dbg>     texio.write('  EOLsig=' .. new)
521     local MinFull = luatypo.MinFull
522     local MinPart = luatypo.MinPart
523     MinFull = math.min(MinPart, MinFull)
524     local k = MinPart
525     local oldlast = string.gsub (old, '.*_', '')
526     local newlast = string.gsub (new, '.*_', '')
527     local i = string.find(new, "_")
528     if i and i > maxlen - MinPart + 1 then
529         k = MinPart + 1
530     end
531     local oldsub = string.sub(old, -k)
532     local newsub = string.sub(new, -k)
533     local l = string.len(new)
534     if oldsub == newsub and l >= k then
535 <dbg>         texio.write_nl('EOLnewsub=' .. newsub)
536         match = true
537     elseif oldlast == newlast and string.len(newlast) >= MinFull then
538 <dbg>         texio.write_nl('EOLnewlast=' .. newlast)
539         match = true
```



```

540         oldsub = oldlast
541         newsub = newlast
542         k = string.len(newlast)
543     end
544     if match then

```

Minimal partial match; any more glyphs matching?

```

545         local osub = oldsub
546         local nsub = newsub
547         while osub == nsub and k <= maxlen do
548             k = k + 1
549             osub = string.sub(old, -k)
550             nsub = string.sub(new, -k)
551             if osub == nsub then
552                 newsub = nsub
553             end
554         end
555         pageflag = true
556         newsub = string.gsub(newsub, '^_', '')
557 (dbg)         texio.write_nl('EOLfullmatch=' .. newsub)
558         local msg = "E.O.L. MATCH=" .. newsub
559         log_flaw(msg, line, colno, footnote)

```

Lest's colour the matching string.

```

560         oldsub = string.reverse(newsub)
561         local newsub = ""
562         local n = lastn
563         repeat
564             if n and n.id ~= GLUE then
565                 color_node(n, COLOR)
566                 l, newsub = signature(n, newsub, swap)
567             elseif n and n.id == GLUE then
568                 newsub = newsub .. "_"
569             elseif not n and box then
570                 n = box
571             else
572                 break
573             end
574             n = n.prev
575         until newsub == oldsub or l >= k
576     end
577 end
578 return new
579 end

```

Same thing for beginning of lines: check the first two words and compare their signature with the previous line's.

```

580 local check_first_word = function (old, node, line, flag)
581     local COLOR = luatypo.colortbl[10]
582     local match = false
583     local swap = false
584     local new = ""
585     local maxlen = 0

```

```

586 local n = node
587 local box, go
588 while n and n.id ~= GLYPH and n.id ~= DISC and
589     (n.id ~= HLIST or n.subtype == INDENT) do
590     n = n.next
591 end
592 local start = n
593 if n and n.id == HLIST then
594     box = n
595     start = n.head
596     n = n.head
597 end
598 while n and n.id ~= GLUE do
599     maxlen, new = signature (n, new, swap)
600     n = n.next
601 end
602 if n and n.id == GLUE then
603     new = new .. "_"
604     go = true
605 elseif box and not n then
606     local bn = box.next
607     if bn.id == GLUE then
608         new = new .. "_"
609         n = bn
610     else
611         n = box
612     end
613     go = true
614 end
615 if go then
616     repeat
617         n = n.next
618         maxlen, new = signature (n, new, swap)
619     until not n or n.id == GLUE
620 end
621 (dbg) texio.write_nl('BOLsigold=' .. old)
622 (dbg) texio.write('    BOLsig=' .. new)

```

When called with flag false, `check_first_word` returns the first word's signature, but doesn't compare it with the previous line's.

```

623 if flag then
624     local MinFull = luatypo.MinFull
625     local MinPart = luatypo.MinPart
626     MinFull = math.min(MinPart, MinFull)
627     local k = MinPart
628     local oldsub = ""
629     local newsub = ""
630     local oldfirst = string.gsub (old, '_.*', '')
631     local newfirst = string.gsub (new, '_.*', '')
632     local i = string.find(new, "_")
633     if i and i <= MinPart then
634         k = MinPart + 1
635     end
636     local oldsub = string.sub(old, 1, k)

```

```

637     local newsub = string.sub(new,1,k)
638     local l = string.len(newsub)
639     if oldsub == newsub and l >= k then
640 (dbg)         texio.write_nl('BOLnewsub=' .. newsub)
641         match = true
642     elseif oldfirst == newfirst and string.len(newfirst) >= MinFull then
643 (dbg)         texio.write_nl('BOLnewfirst=' .. newfirst)
644         match = true
645         oldsub = oldfirst
646         newsub = newfirst
647         k = string.len(newfirst)
648     end
649     if match then

```

Minimal partial match; any more glyphs matching?

```

650         local osub = oldsub
651         local nsub = newsub
652         while osub == nsub and k <= maxlen do
653             k = k + 1
654             osub = string.sub(old,1,k)
655             nsub = string.sub(new,1,k)
656             if osub == nsub then
657                 newsub = nsub
658             end
659         end
660         pageflag = true
661         newsub = string.gsub(newsub, '_$', '') --$
662 (dbg)         texio.write_nl('BOLfullmatch=' .. newsub)
663         local msg = "B.O.L. MATCH=" .. newsub
664         log_flaw(msg, line, colno, footnote)

```

Lest's colour the matching string.

```

665         oldsub = newsub
666         local newsub = ""
667         local k = string.len(oldsub)
668         local n = start
669         repeat
670             if n and n.id ~= GLUE then
671                 color_node(n, COLOR)
672                 l, newsub = signature(n, newsub, swap)
673             elseif n and n.id == GLUE then
674                 newsub = newsub .. "_"
675             elseif not n and box then
676                 n = box
677             else
678                 break
679             end
680             n = n.next
681         until newsub == oldsub or l >= k
682     end
683 end
684 return new
685 end

```

This auxillary function looks for a short word (one or two chars) at end of lines, compares it to a given list and colours it if matches. The first argument must be a node of type `GLYPH`, usually the last line's node, the second one is the line number.

TODO: where does “out of range” starts? U+0100? U+0180?

```
686 local check_regexpr = function (glyph, line)
687   local pageno = tex.getcount("c@page")
688   local COLOR = luatypo.colortbl[3]
689   local lang = glyph.lang
690   local match = false
691   local lchar, id = is_glyph(glyph)
692   local previous = glyph.prev
```

First look for single chars unless the list of words is empty.

```
693   if lang and luatypo.single[lang] then
```

For single char words, the previous node is a glue.

```
694     if lchar and lchar < 0x100 and previous and previous.id == GLUE then
695       match = string.find(luatypo.single[lang], string.char(lchar))
696       if match then
697         pageflag = true
698         local msg = "RGX MATCH=" .. string.char(lchar)
699         log_flaw(msg, line, colno, footnote)
700         color_node(glyph,COLOR)
701       end
702     end
703   end
```

Look for two chars words unless the list of words is empty.

```
704   if lang and luatypo.double[lang] then
705     if lchar and previous and previous.id == GLYPH then
706       local pchar, id = is_glyph(previous)
707       local pprev = previous.prev
```

For two chars words, the previous node is a glue...

```
708       if pchar and pchar < 0x100 and pprev and pprev.id == GLUE then
709         local pattern = string.char(pchar) .. string.char(lchar)
710         match = string.find(luatypo.double[lang], pattern)
711         if match then
712           pageflag = true
713           local msg = "RGX MATCH=" .. pattern
714           log_flaw(msg, line, colno, footnote)
715           color_node(previous,COLOR)
716           color_node(glyph,COLOR)
717         end
718       end
```

...unless a kern is found between the two chars.

```
719     elseif lchar and previous and previous.id == KERN then
720       local pprev = previous.prev
721       if pprev and pprev.id == GLYPH then
722         local pchar, id = is_glyph(pprev)
```

```

723         local ppprev = pprev.prev
724         if pchar and pchar < 0x100 and
725             ppprev and ppprev.id == GLUE then
726             local pattern = string.char(pchar) .. string.char(lchar)
727             match = string.find(luatypo.double[lang], pattern)
728             if match then
729                 pageflag = true
730                 local msg = "RGX MATCH=" .. pattern
731                 log_flaw(msg, line, colno, footnote)
732                 color_node(pprev, COLOR)
733                 color_node(glyph, COLOR)
734             end
735         end
736     end
737 end
738 end
739 end

```

This auxillary function prints the first part of an hyphenated word up to the discretionary, with a supplied colour. It requires two arguments: a `DISC` node and a (named) colour.

```

740 local show_pre_disc = function (disc, color)
741     local n = disc
742     while n and n.id ~= GLUE do
743         color_node(n, color)
744         n = n.prev
745     end
746     return n
747 end

```

footnoterule-ahead This auxillary function scans the current `VLIST` in search of a `\footnoterule`; it returns `true` if found, false otherwise. The `RULE` node above footnotes is normally surrounded by two (vertical) `KERN` nodes, the total height is either 0 (standard and koma classes) or equals the rule's height (memoir class).

```

748 local footnoterule_ahead = function (head)
749     local n = head
750     local flag = false
751     local totalht, ruleht, ht1, ht2, ht3
752     if n and n.id == KERN and n.subtype == 1 then
753         totalht = n.kern
754         n = n.next
755         <dbg> ht1 = string.format("%.2fpt", totalht/65536)

756         while n and n.id == GLUE do n = n.next end
757         if n and n.id == RULE and n.subtype == 0 then
758             ruleht = n.height
759             <dbg> ht2 = string.format("%.2fpt", ruleht/65536)
760             totalht = totalht + ruleht
761             n = n.next
762             if n and n.id == KERN and n.subtype == 1 then
763                 <dbg> ht3 = string.format("%.2fpt", n.kern/65536)
764                 totalht = totalht + n.kern

```

```

765         if totalht == 0 or totalht == ruleht then
766             flag = true
767         else
768 <dbg>             texio.write_nl(' ')
769 <dbg>             texio.write_nl('Not a footnoterule:')
770 <dbg>             texio.write(' KERN height=' .. ht1)
771 <dbg>             texio.write(' RULE height=' .. ht2)
772 <dbg>             texio.write(' KERN height=' .. ht3)
773         end
774     end
775 end
776 end
777 return flag
778 end

```

get-pagebody This auxillary function scans the VLISTS on the current page in search of the page body. It returns the corresponding node or nil in case of failure.

```

779 local get_pagebody = function (head)
780     local textht = tex.getdimen("textheight")
781     local fn = head.list
782     local body = nil
783     repeat
784         fn = fn.next
785     until fn.id == VLIST and fn.height > 0
786 <dbg>     texio.write_nl(' ')
787 <dbg>     local ht = string.format("%.1fpt", fn.height/65536)
788 <dbg>     local dp = string.format("%.1fpt", fn.depth/65536)
789 <dbg>     texio.write_nl('get_pagebody: TOP VLIST')
790 <dbg>     texio.write(' ht=' .. ht .. ' dp=' .. dp)
791     first = fn.list
792     for n in traverse_id(VLIST,first) do
793         if n.subtype == 0 and n.height == textht then
794 <dbg>             local ht = string.format("%.1fpt", n.height/65536)
795 <dbg>             texio.write_nl('BODY found: ht=' .. ht)
796 <dbg>             texio.write_nl(' ')
797             body = n
798             break
799         else
800 <dbg>             texio.write_nl('Skip short VLIST:')
801 <dbg>             local ht = string.format("%.1fpt", n.height/65536)
802 <dbg>             local dp = string.format("%.1fpt", n.depth/65536)
803 <dbg>             texio.write(' ht=' .. ht .. ' dp=' .. dp)
804             first = n.list
805             for n in traverse_id(VLIST,first) do
806                 if n.subtype == 0 and n.height == textht then
807 <dbg>                     local ht = string.format("%.1fpt", n.height/65536)
808 <dbg>                     texio.write_nl(' BODY: ht=' .. ht)
809                     body = n
810                     break
811                 end
812             end
813         end
814     end

```

```

815 if not body then
816     texio.write_nl('***lua-typo ERROR: PAGE BODY *NOT* FOUND!***')
817 end
818 return body
819 end

```

check-vtop This function is called repeatedly by `check_page` (see below); it scans the boxes found in the page body (f.i. columns) in search of typographical flaws and logs them if it finds any.

```

820 check_vtop = function (head, colno, vpos)
821     local PAGEmin    = luatypo.PAGEmin
822     local HYPHmax    = luatypo.HYPHmax
823     local LLminWD    = luatypo.LLminWD
824     local BackPI     = luatypo.BackPI
825     local BackFuzz   = luatypo.BackFuzz
826     local BackParindent = luatypo.BackParindent
827     local ShortLines = luatypo.ShortLines
828     local ShortPages = luatypo.ShortPages
829     local OverfullLines = luatypo.OverfullLines
830     local UnderfullLines = luatypo.UnderfullLines
831     local Widows      = luatypo.Widows
832     local Orphans     = luatypo.Orphans
833     local EOPHyphens  = luatypo.EOPHyphens
834     local RepeatedHyphens = luatypo.RepeatedHyphens
835     local FirstWordMatch = luatypo.FirstWordMatch
836     local ParLastHyphen = luatypo.ParLastHyphen
837     local EOLShortWords = luatypo.EOLShortWords
838     local LastWordMatch = luatypo.LastWordMatch
839     local FootnoteSplit = luatypo.FootnoteSplit
840     local Stretch      = math.max(luatypo.Stretch/100,1)
841     local blskip       = tex.getglue("baselineskip")
842     local vpos_min     = PAGEmin * blskip
843     vpos_min = vpos_min * 1.5
844     local linewd       = tex.getdimen("textwidth")
845     local first_bot    = true
846     local footnote     = false
847     local ftnsplit     = false
848     local orphanflag   = false
849     local widowflag    = false
850     local lwhyphflag   = false
851     local pageshort    = false
852     local firstwd      = ""
853     local lastwd       = ""
854     local hyphcount    = 0
855     local pageline     = 0
856     local ftnline     = 0
857     local line         = 0
858     local body_bottom  = false
859     local page_bottom  = false
860     local pageflag     = false
861     local plist        = luatypo.pagelist
862     local lastp        = tonumber(string.match(plist, "%s(%d+),%s$"))
863     local pageno       = tex.getcount("c@page")

```

The main loop scans the content of the `\vtop` holding the page (or column) body, footnotes included.

```
864 while head do
865     local nextnode = head.next
```

Let's scan the top nodes of this vbox: expected are HLIST (text lines or vboxes), RULE, KERN, GLUE...

```
866 if head.id == HLIST and head.subtype == LINE and
867     (head.height > 0 or head.depth > 0) then
```

This is a text line, store the line width, increment counters `pageline` or `ftnline` and `line` (for `log_flaw`). Let's update `vpos` (vertical position in 'sp' units) too.

```
868     vpos = vpos + head.height + head.depth
869     linewidth = head.width
870     if footnote then
871         ftnline = ftnline + 1
872         line = ftnline
873     else
874         pageline = pageline + 1
875         line = pageline
876     end
```

Is this line the last one on the page or before footnotes?

This has to be known early (orphanflag, lwhyphflag, ftnsplit).

```
877     local n = nextnode
878     while n and (n.id == GLUE or n.id == PENALTY or
879         n.id == WHATSIT ) do
880         n = n.next
881     end
882     if not n then
883         page_bottom = true
884         body_bottom = true
885     elseif footnoterule_ahead(n) then
886         body_bottom = true
887 <dbg> texio.write_nl('=> FOOTNOTE RULE ahead')
888 <dbg> texio.write_nl('check_vtop: last line before footnotes')
889 <dbg> texio.write_nl(' ')
890     end
```

Is the current line overfull or underfull?

```
891     local first = head.head
892     local hmax = linewidth + tex.hfuzz
893     local w,h,d = dimensions(1,2,0, first)
894     if w > hmax and OverfullLines then
895         pageflag = true
896         local wpt = string.format("%.2fpt", (w-head.width)/65536)
897         local msg = "OVERFULL line " .. wpt
898         log_flaw(msg, line, colno, footnote)
899         local COLOR = luatypo.colortbl[7]
900         color_line(head, COLOR)
901     elseif head.glue_set > Stretch and head.glue_sign == 1 and
902         head.glue_order == 0 and UnderfullLines then
```



```

903         pageflag = true
904         local s = string.format("%.0f%s", 100*head.glue_set, "%")
905         local msg = "UNDERFULL line stretch=" .. s
906         log_flaw(msg, line, colno, footnote)
907         local COLOR = luatypo.colortbl[8]
908         color_line (head, COLOR)
909     end

```

Set flag `ftnsplit` to `true` on every page's last line. This flag will be reset to false if the current line ends a paragraph.

```

910         if footnote and page_bottom then
911             ftnsplit = true
912         end

```

The current node is a line, `first` is the line's first node. Skip margin kern and/or leftskip if any.

```

913         while first.id == MKERN or
914             (first.id == GLUE and first.subtype == LFTSKIP) do
915             first = first.next
916         end
917         local ListItem = false

```

Now let's analyse the beginning of the current line.

```

918         if first.id == LPAR then

```

It starts a paragraph... Reset `parline` except in footnotes (`parline` and `pageline` counts are for “body” *only*, they are frozen in footnotes).

```

919             hyphcount = 0
920             if not footnote then
921                 parline = 1
922                 if body_bottom then

```

We are at the page bottom (footnotes excluded), this line is an orphan (unless it is the unique line of the paragraph, this will be checked later when scanning the end of line).

```

923                 orphanflag = true
924             end
925         end

```

List items begin with `LPAR` followed by an hbox.

```

926             local nn = first.next
927             if nn and nn.id == HLIST and nn.subtype == BOX then
928                 ListItem = true
929             end
930             elseif not footnote then
931                 parline = parline + 1
932             end

```

Let's track lines beginning with the same word (except lists).

```

933             if FirstWordMatch then
934                 local flag = not ListItem
935                 firstwd = check_first_word(firstwd, first, line, flag)
936             end

```

Let's check the end of line: `ln` (usually a `rightskip`) and `pn` are the last two nodes.

```
937         local ln = slide(first)
938         local pn = ln.prev
939         if pn and pn.id == GLUE and pn.subtype == PARFILL then
```

CASE 1: this line ends the paragraph, reset `ftnsplit` and `orphan` flags to false...

```
940         hyphcount = 0
941         ftnsplit = false
942         orphanflag = false
```

but it is a widow if it is the page's first line and it doesn't start a new paragraph.

Orphans and widows will be colored later.

```
943         if pageline == 1 and parline > 1 then
944             widowflag = true
945         end
```

`PFskip` is the rubber length (in sp) added to complete the line.

```
946         local PFskip = effective_glue(pn,head)
947         if ShortLines then
948             local llwd = linewidth - PFskip
949 <dbg>             local PFskip_pt = string.format("%.1fpt", PFskip/65536)
950 <dbg>             local llwd_pt = string.format("%.1fpt", llwd/65536)
951 <dbg>             texio.write_nl('PFskip= ' .. PFskip_pt)
952 <dbg>             texio.write(' llwd= ' .. llwd_pt)
```

`llwd` is the line's length. Is it too short?

```
953         if llwd < LLminWD then
954             pageflag = true
955             local msg = "SHORT LINE: " ..
956                 string.format("%.0fpt", llwd/65536)
957             log_flaw(msg, line, colno, footnote)
958             local COLOR = luatypo.colortbl[6]
959             local attr = oberdiek.luacolor.getattribute()
```

let's colour the whole line.

```
960             color_line (head, COLOR)
961         end
962     end
```

Is this line nearly full? (ending too close to the right margin)

```
963         if BackParindent and PFskip < BackPI and PFskip > BackFuzz then
964             pageflag = true
965             local msg = "LINE NEARLY FULL: missing " ..
966                 string.format("%.1fpt", PFskip/65536)
967             log_flaw(msg, line, colno, footnote)
968             local COLOR = luatypo.colortbl[12]
969             local attr = oberdiek.luacolor.getattribute()
970             color_line (head, COLOR)
971         end
```

Does the last word and the one on the previous line match?

```

972         if LastWordMatch then
973             local flag = textline
974             if PFskip > BackPI then
975                 flag = false
976             end
977             lastwd = check_last_word(lastwd, pn, line, flag)
978         end
979     elseif pn and pn.id == DISC then

```

CASE 2: the current line ends with an hyphen.

```

980         hyphcount = hyphcount + 1
981         if LastWordMatch then
982             lastwd = check_last_word(lastwd, ln, line, true)
983         end
984         if hyphcount > HYPHmax and RepeatedHyphens then
985             local COLOR = luatypo.colortbl[2]
986             local pg = show_pre_disc (pn,COLOR)
987             pageflag = true
988             local msg = "REPEATED HYPHENS: more than " .. HYPHmax
989             log_flaw(msg, line, colno, footnote)
990         end
991         if (page_bottom or body_bottom) and EOPHyphens then

```

This hyphen occurs on the page's last line (body or footnote).

```

992             lwhyphflag = true
993         end
994         if nextnode and ParLastHyphen then

```

Does the next line end the current paragraph? If so, `nextnode` is a 'linebreak penalty', the next one is a 'baseline skip' and the node after a HLIST of subtype LINE with `glue_order=2`.

```

995             local nn = nextnode.next
996             local nnn = nil
997             if nn and nn.next then
998                 nnn = nn.next
999                 if nnn.id == HLIST and nnn.subtype == LINE and
1000                    nnn.glue_order == 2 then
1001                     pageflag = true
1002                     local msg = "HYPHEN on next to last line"
1003                     log_flaw(msg, line, colno, footnote)
1004                     local COLOR = luatypo.colortbl[0]
1005                     local pg = show_pre_disc (pn,COLOR)
1006                 end
1007             end
1008         end

```

CASE 3: the current line ends with anything else (MKERN, GLYPH, HLIST, etc.), reset `hyphcount`, perform checks for 'LastWordMatch' and for 'EOLShortWords'.

```

1009     else
1010         hyphcount = 0
1011         if LastWordMatch and pn then
1012             lastwd = check_last_word(lastwd, pn, line, true)
1013         end

```

```

1014         if EOLShortWords then
1015             while pn and pn.id ~= GLYPH and pn.id ~= HLIST do
1016                 pn = pn.prev
1017             end
1018             if pn and pn.id == GLYPH then
1019                 check_regexpr(pn,line)
1020             end
1021         end
1022     end

```

Colour the whole line if it is a widow.

```

1023     if widowflag and Widows then
1024         pageflag = true
1025         widowflag = false
1026         local msg = "WIDOW"
1027         log_flaw(msg, line, colno, footnote)
1028         local COLOR = luatypo.colortbl[4]
1029         color_line (head, COLOR)
1030     end

```

Colour the whole line if it is an orphan or footnote continuing on the next page.

```

1031     if orphanflag and Orphans then
1032         pageflag = true
1033         local msg = "ORPHAN"
1034         log_flaw(msg, line, colno, footnote)
1035         local COLOR = luatypo.colortbl[5]
1036         color_line (head, COLOR)
1037     end
1038     if ftnsplit and FootnoteSplit then
1039         pageflag = true
1040         local msg = "FOOTNOTE SPLIT"
1041         log_flaw(msg, line, colno, footnote)
1042         local COLOR = luatypo.colortbl[13]
1043         color_line (head, COLOR)
1044     end

```

Colour (differently) the last word if hyphenated.

```

1045     if lwhyphflag and EOPHyphens then
1046         pageflag = true
1047         local msg = "LAST WORD SPLIT"
1048         log_flaw(msg, line, colno, footnote)
1049         local COLOR = luatypo.colortbl[1]
1050         local pg = show_pre_disc (pn,COLOR)
1051     end

```

End of scanning for the main type of node (text lines).

```

1052     elseif head.id == HLIST and
1053         (head.subtype == EQN or head.subtype == ALIGN) and
1054         (head.height > 0 or head.depth > 0) then

```

This line is a displayed or aligned equation. Let's update vpos and the line number.

```

1055         vpos = vpos + head.height + head.depth
1056         if footnote then

```

```

1057         ftnline = ftnline + 1
1058         line = ftnline
1059     else
1060         pageline = pageline + 1
1061         line = pageline
1062     end

```

Let's check for an "Overfull box". For a displayed equation it is straightforward. A set of aligned equations all have the same (maximal) width; in order to avoid highlighting the whole set, we have to look for glues at the end of embedded HLISTs.

```

1063     local fl = true
1064     local wd = 0
1065     local hmax = 0
1066     if head.subtype == EQN then
1067         local f = head.list
1068         wd = rangedimensions(head,f)
1069         hmax = head.width + tex.hfuzz
1070     else
1071         wd = head.width
1072         hmax = tex.getdimen("linewidth") + tex.hfuzz
1073     end
1074     if wd > hmax and OverfullLines then
1075         if head.subtype == ALIGN then
1076             local first = head.list
1077             for n in traverse_id(HLIST, first) do
1078                 local last = slide(n.list)
1079                 if last.id == GLUE and last.subtype == USER then
1080                     wd = wd - effective_glue(last,n)
1081                     if wd <= hmax then fl = false end
1082                 end
1083             end
1084         end
1085         if fl then
1086             pageflag = true
1087             local w = wd - hmax + tex.hfuzz
1088             local wpt = string.format("%.2fpt", w/65536)
1089             local msg = "OVERFULL equation " .. wpt
1090             log_flaw(msg, line, colno, footnote)
1091             local COLOR = luatypo.colortbl[7]
1092             color_line (head, COLOR)
1093         end
1094     end
1095     elseif head and head.id == RULE and head.subtype == 0 then

```

This is a RULE, possibly a footnote rule.

```

1096         vpos = vpos + head.height + head.depth
1097         if body_bottom then

```

If a `\footnoterule` has been detected on the previous run, set the `footnote` flag and reset some counters and flags for the coming footnote lines.

```

1098 <dbg>         texio.write_nl('check_vtop: footnotes start')
1099 <dbg>         texio.write_nl(' ')
1100         footnote = true

```

```

1101         ftnline = 0
1102         body_bottom = false
1103         ftnrule_ahead = false
1104         orphanflag = false
1105         lwhyphflag = false
1106         hyphcount = 0
1107         firstwd = ""
1108         lastwd = ""
1109     end

```

Track short pages: check the number of lines at end of page, in case this number is low, *and* `vpos` is less than `vpos_min`, fetch the last line and colour it.

NOTE1: `effective_glue` requires a ‘parent’ node, as pointed out by Marcel Krüger on S.E., this implies using `pre_shipout_filter` instead of `pre_output_filter`.

NOTE2: Widows are already detected, skip them here; there are usually two consecutive nodes of type GLUE-0 at end of pages...

```

1110     elseif body_bottom and head.id == GLUE and head.subtype == 0 then
1111         if first_bot then
1112             <dbg> local vpos_pt = string.format("%.1fpt", vpos/65536)
1113             <dbg> local vmin_pt = string.format("%.1fpt", vpos_min/65536)
1114             <dbg> texio.write_nl('pageline=' .. pageline)
1115             <dbg> texio.write_nl('vpos=' .. vpos_pt)
1116             <dbg> texio.write(' vpos_min=' .. vmin_pt)
1117             <dbg> if page_bottom then
1118                 <dbg> local tht = tex.getdimen("textheight")
1119                 <dbg> local tht_pt = string.format("%.1fpt", tht/65536)
1120                 <dbg> texio.write(' textheight=' .. tht_pt)
1121             <dbg> end
1122             <dbg> texio.write_nl(' ')
1123             if pageline > 1 and pageline < PAGEmin and ShortPages then
1124                 pageshort = true
1125             end
1126             if pageshort and vpos < vpos_min then
1127                 pageflag = true
1128                 local msg = "SHORT PAGE: only " .. pageline .. " lines"
1129                 log_flaw(msg, line, colno, footnote)
1130                 local COLOR = luatypo.colortbl[9]
1131                 local n = head
1132                 repeat
1133                     n = n.prev
1134                 until n.id == HLIST
1135                 color_line (n, COLOR)
1136             end
1137             first_bot = false
1138         end
1139     elseif head.id == GLUE then

```

Increment `vpos` on other vertical glues.

```

1140         vpos = vpos + effective_glue(head,body)
1141     elseif head.id == KERN and head.subtype == 1 then

```

This is a vertical kern, let's update `vpos`.

```

1142     vpos = vpos + head.kern
1143     elseif head.id == VLIST then

```

This is a vertical a \vbox, let's update vpos.

```

1144     vpos = vpos + head.height + head.depth

```

Leave `check_vtop` if a two columns box starts.

```

1145     elseif head.id == HLIST and head.subtype == BOX then
1146         local hf = head.list
1147         if hf and hf.id == VLIST and hf.subtype == 0 then
1148 <dbg>             texio.write_nl('check_vtop: BREAK => multicol')
1149 <dbg>             texio.write_nl(' ')
1150                 break
1151         end
1152     end
1153     head = nextnode
1154 end
1155 <dbg> if nextnode then
1156 <dbg>     texio.write('Exit check_vtop, next=')
1157 <dbg>     texio.write(tostring(node.type(nextnode.id)))
1158 <dbg>     texio.write('-'.. nextnode.subtype)
1159 <dbg> else
1160 <dbg>     texio.write_nl('Exit check_vtop, next=nil')
1161 <dbg> end
1162 <dbg> texio.write_nl('')

```

Record flaws:

```

1163 if pageflag then
1164     if not lastp or pageno > lastp then
1165         luatypo.pagelist = luatypo.pagelist .. tostring(pageno) .. ", "
1166     end
1167 end
1168 return head

```

head is nil unless `check_vtop` exited on a two column start.

```

1169 end

```

`check-page` This is the main function which will be added to the `pre_shipout_filter` callback unless option `None` is selected. It executes `get_pagebody` which returns a node of type `VLIST-0`, then scans this `VLIST`: expected are `VLIST-0` (full width block) or `HLIST-2` (multi column block). The vertical position of the current node is stored in the `vpos` dimension (integer in 'sp' units, 1 pt = 65536 sp). It is used to detect short pages.

```

1170 luatypo.check_page = function (head)
1171     local textwd = tex.getdimen("textwidth")
1172     local vpos = 0
1173     local n2, n3, col, colno
1174     local body = get_pagebody(head)
1175     local footnote = false
1176     local top = body
1177     local first = body.list
1178     if (first and first.id == HLIST and first.subtype == BOX) or
1179         (first and first.id == VLIST and first.subtype == 0) then

```

Some classes (memoir, tugboat ...) use one more level of bowing, let's step down one level.

```

1180 <dbg>      local boxwd = string.format("%.1fpt", first.width/65536)
1181 <dbg>      texio.write_nl('One step down: boxwd=' .. boxwd)
1182 <dbg>      texio.write_nl(' ')
1183      top = body.list
1184      first = top.list
1185  end
1186  while top do
1187      first = top.list
1188 <dbg>      texio.write_nl('Page loop: top=' .. tostring(node.type(top.id)))
1189 <dbg>      texio.write('-' .. top.subtype)
1190 <dbg>      texio.write_nl(' ')
1191      if top and top.id == VLIST and top.subtype == 0 and
1192          top.width > textwd/2                                then

```

Single column, run `check_vtop` on the top vlist.

```

1193 <dbg>      local boxht = string.format("%.1fpt", top.height/65536)
1194 <dbg>      local boxwd = string.format("%.1fpt", top.width/65536)
1195 <dbg>      texio.write_nl('**VLIST: ')
1196 <dbg>      texio.write(tostring(node.type(top.id)))
1197 <dbg>      texio.write('-' .. top.subtype)
1198 <dbg>      texio.write(' wd=' .. boxwd .. ' ht=' .. boxht)
1199 <dbg>      texio.write_nl(' ')
1200      local next = check_vtop(first,colno,vpos)
1201      if next then
1202          top = next
1203      elseif top then
1204          top = top.next
1205      end
1206      elseif (top and top.id == HLIST and top.subtype == BOX) and
1207          (first and first.id == VLIST and first.subtype == 0) and
1208          (first.height > 0 and first.width > 0) then

```

Two or more columns, each one is boxed in a vlist.

Run `check_vtop` on every column.

```

1209 <dbg>      texio.write_nl('**MULTICOL type1:')
1210 <dbg>      texio.write_nl(' ')
1211      colno = 0
1212      for n in traverse_id(VLIST, first) do
1213          colno = colno + 1
1214          col = n.list
1215 <dbg>          texio.write_nl('Start of col.' .. colno)
1216 <dbg>          texio.write_nl(' ')
1217          check_vtop(col,colno,vpos)
1218 <dbg>          texio.write_nl('End of col.' .. colno)
1219 <dbg>          texio.write_nl(' ')
1220      end
1221      colno = nil
1222      top = top.next
1223 <dbg>      texio.write_nl('MULTICOL type1 END: next=')
1224 <dbg>      texio.write(tostring(node.type(top.id)))
1225 <dbg>      texio.write('-' .. top.subtype)

```



```

1226 <dbg>      texio.write_nl(' ')
1227      elseif (top and top.id == HLIST and top.subtype == BOX) and
1228      (first and first.id == HLIST and first.subtype == BOX) and
1229      (first.height > 0 and first.width > 0) then

```

Two or more columns, each one is boxed in an hlist which holds a vlist.

Run `check_vtop` on every column.

```

1230 <dbg>      texio.write_nl('**MULTICOL type2:')
1231 <dbg>      texio.write_nl(' ')
1232      colno = 0
1233      for n in traverse_id(HLIST, first) do
1234          colno = colno + 1
1235          local nn = n.list
1236          if nn and nn.list then
1237              col = nn.list
1238 <dbg>          texio.write_nl('Start of col.' .. colno)
1239 <dbg>          texio.write_nl(' ')
1240              check_vtop(col,colno,vpos)
1241 <dbg>          texio.write_nl('End of col.' .. colno)
1242 <dbg>          texio.write_nl(' ')
1243          end
1244      end
1245      colno = nil
1246      top = top.next
1247  else
1248      top = top.next
1249  end
1250 end
1251 return true
1252 end
1253 return luatypo.check_page
1254 \end{luacode}

```

Add the `luatypo.check_page` function to the `pre_shipout_filter` callback (with priority 1 for color attributes to be effective), unless option `None` is selected; remember that the `None` boolean's value is forwarded to Lua 'AtEndOfPackage'...

```

1255 \AtEndOfPackage{%
1256   \directlua{
1257     if not luatypo.None then
1258       luatexbase.add_to_callback
1259         ("pre_shipout_filter",luatypo.check_page,"check_page",1)
1260     end
1261   }%
1262 }

```

Load a local config file if present in LaTeX's search path.

Otherwise, set reasonable defaults.

```

1263
1264 \InputIfFileExists{lua-typo.cfg}%
1265   {\PackageInfo{lua-typo.sty}{lua-typo.cfg file loaded}}%
1266   {\PackageInfo{lua-typo.sty}{lua-typo.cfg file not found.
1267     \MessageBreak Providing default values.}%
1268   \definecolor{mygrey}{gray}{0.6}%

```

```

1269 \definecolor{myred}{rgb}{1,0.55,0}
1270 \luatypSetColor0{red}% Paragraph last full line hyphenated
1271 \luatypSetColor1{red}% Page last word hyphenated
1272 \luatypSetColor2{red}% Hyphens on to many consecutive lines
1273 \luatypSetColor3{red}% Short word at end of line
1274 \luatypSetColor4{cyan}% Widow
1275 \luatypSetColor5{cyan}% Orphan
1276 \luatypSetColor6{cyan}% Paragraph ending on a short line
1277 \luatypSetColor7{blue}% Overfull lines
1278 \luatypSetColor8{blue}% Underfull lines
1279 \luatypSetColor9{red}% Nearly empty page
1280 \luatypSetColor{10}{myred}% First word matches
1281 \luatypSetColor{11}{myred}% Last word matches
1282 \luatypSetColor{12}{mygrey}% Paragraph ending on a nearly full line
1283 \luatypSetColor{13}{cyan}% Footnote split
1284 \luatypoBackPI=1em\relax
1285 \luatypoBackFuzz=2pt\relax
1286 \ifdim\parindent=0pt \luatypoLLminWD=20pt\relax
1287 \else\luatypoLLminWD=2\parindent\relax\fi
1288 \luatypoStretchMax=200\relax
1289 \luatypoHyphMax=2\relax
1290 \luatypoPageMin=5\relax
1291 \luatypoMinFull=4\relax
1292 \luatypoMinPART=4\relax
1293 }%

```

5 Configuration file

```
%%% Configuration file for lua-typo.sty
%%% These settings can also be overruled in the preamble.

%% Minimum gap between end of paragraphs' last lines and the right margin
\luatypoBackPI=1em\relax
\luatypoBackFuzz=2pt\relax

%% Minimum length of paragraphs' last lines
\ifdim\parindent=0pt \luatypoLLminWD=20pt\relax
\else \luatypoLLminWD=2\parindent\relax
\fi

%% Maximum number of consecutive hyphenated lines
\luatypoHyphMax=2\relax

%% Nearly empty pages: minimum number of lines
\luatypoPageMin=5\relax

%% Maximum acceptable stretch before a line is tagged as Underfull
\luatypoStretchMax=200\relax

%% Minimum number of matching characters for words at begin/end of line
\luatypoMinFull=3\relax
\luatypoMinPart=4\relax

%% Default colours = red, cyan, mygrey
\definecolor{mygrey}{gray}{0.6}
\definecolor{myred}{rgb}{1,0.55,0}
\luatypoSetColor0{red}      % Paragraph last full line hyphenated
\luatypoSetColor1{red}      % Page last word hyphenated
\luatypoSetColor2{red}      % Hyphens on to many consecutive lines
\luatypoSetColor3{red}      % Short word at end of line
\luatypoSetColor4{cyan}     % Widow
\luatypoSetColor5{cyan}     % Orphan
\luatypoSetColor6{cyan}     % Paragraph ending on a short line
\luatypoSetColor7{blue}     % Overfull lines
\luatypoSetColor8{blue}     % Underfull lines
\luatypoSetColor9{red}      % Nearly empty page (just a few lines)
\luatypoSetColor{10}{myred} % First word matches
\luatypoSetColor{11}{myred} % Last word matches
\luatypoSetColor{12}{mygrey}% Paragraph ending on a nearly full line
\luatypoSetColor{13}{cyan}  % Footnote split

%% Language specific settings (example for French):
%% short words (two letters max) to be avoided at end of lines.
%%\luatypoOneChar{french}{'À à Ô'}
%%\luatypoTwoChars{french}{'Je Tu Il On'}
```

6 Debugging lua-typo

Personal stuff useful *only* for maintaining the `lua-typo` package has been added at the end of `lua-typo.dtx` in version 0.60. It is not extracted unless a) both ‘`\iffalse`’ and ‘`\fi`’ on lines 41 and 46 at the beginning of `lua-typo.dtx` are commented out and b) all files are generated again by a `luatex lua-typo.dtx` command; then a (very) verbose version of `lua-typo.sty` is generated together with a `scan-page.sty` file which can be used instead of `lua-typo.sty` to show the structured list of nodes found in a document.

7 Change History

Changes are listed in reverse order (latest first) from version 0.30.

v0.60	get-pagebody: New function
General: Debugging stuff added. . . 36	‘get_pagebody’ required for
check-page: Loop redesigned to	callback ‘pre_shipout_filter’. . . . 22
properly handle two columns. . . . 31	check-vtop: Consider displayed and
check-vtop: Break ‘check_vtop’ loop	aligned equations too for overfull
if a two columns box starts. . . . 23	boxes. 28
Loop redesigned. 23	Detection of overfull boxes fixed:
Typographical flaws are recorded	the former code didn’t work for
here (formerly in check_page). . . 23	typewriter fonts. 24
v0.51	footnoterule-ahead: New function
footnoterule-ahead: In some cases	‘footnoterule_ahead’. 21
glue nodes might precede the	v0.40
footnote rule; next line added . . 21	check-vtop: All hlists of subtype
v0.50	LINE now count as a pageline. . . 25
General: Callback ‘pre_output_filter’	Both MKERN and LFTSKIP may
replaced by ‘pre_shipout_filter’, in	occur on the same line. 25
the former the material is not	Title pages, pages with figures
boxed yet and footnotes are not	and/or tables may not be empty
visible. 33	pages: check ‘vpos’ last line’s
Go down deeper into hlists and	position. 23
vlists to colour nodes. 13	v0.32
Homeoarchy detection added for	General: Better protection against
lines starting or ending on \mbox. 15	unexpected nil nodes. 12
Rollback mechanism used for	Experimental code to deal with
recovering older versions. 5	non standard ligatures. 14
Summary of flaws written to file	Functions ‘check_last_word’ and
‘\jobname.typo’. 13	‘check_last_word’ rewritten. . . . 15