

# Highlighting Typographical Flaws with LuaLaTeX

Daniel Flipo

daniel.flipo@free.fr

## 1 What is it about?

The file `lua-typo.sty`<sup>1</sup>, 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 grey by default) and mentioned in the summary provided at the end of the `.log` file.

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!

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 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}
```

---

<sup>1</sup>The file described in this section has version number v.0.32 and was last revised on 2021/03/14.

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

Name	Glitch to highlight
All	Turns all options to <code>true</code>
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?

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`, 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.

Finally, please note that the footnotes' contents are not checked by `lua-typo`, I have currently no clue of how to do that, hints are welcome!

### 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` 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).

**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 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{mygrey}   % Overfull lines
% \luatypoSetColor8{mygrey}   % 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
%
```

`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

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

```
1 \ifdefined\directlua
2   \RequirePackage{luatexbase,luacode,luacolor}
3   \RequirePackage{kvoptions,atveryend}
4 \else
5   \PackageError{This package is meant for LuaTeX only! Aborting}
6               {No more information available, sorry!}
7 \fi
```

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

```
8 \newdimen\luatypoLLminWD
9 \newdimen\luatypoBackPI
10 \newdimen\luatypoBackFuzz
11 \newcount\luatypoStretchMax
12 \newcount\luatypoHyphMax
13 \newcount\luatypoPageMin
14 \newcount\luatypoMinFull
15 \newcount\luatypoMinPart
16 \newcount\luatypo@LANGno
17 \newcount\luatypo@options
18 \newtoks\luatypo@single
19 \newtoks\luatypo@double
```

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

```
20 \begin{luacode}
21   luatypo = { }
22 \end{luacode}
```

Set up `kvoptions` initializations.

```
23 \SetupKeyvalOptions{
24   family=luatypo,
25   prefix=LT@,
26 }
27 \DeclareBoolOption[false]{ShowOptions}
28 \DeclareBoolOption[false]{None}
29 \DeclareBoolOption[false]{All}
30 \DeclareBoolOption[false]{BackParindent}
31 \DeclareBoolOption[false]{ShortLines}
32 \DeclareBoolOption[false]{ShortPages}
33 \DeclareBoolOption[false]{OverfullLines}
34 \DeclareBoolOption[false]{UnderfullLines}
35 \DeclareBoolOption[false]{Widows}
36 \DeclareBoolOption[false]{Orphans}
37 \DeclareBoolOption[false]{EOPHyphens}
38 \DeclareBoolOption[false]{RepeatedHyphens}
39 \DeclareBoolOption[false]{ParLastHyphen}
40 \DeclareBoolOption[false]{EOLShortWords}
41 \DeclareBoolOption[false]{FirstWordMatch}
42 \DeclareBoolOption[false]{LastWordMatch}
```

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

```
43 \AddToKeyvalOption{luatypo}{All}{%
44   \LT@ShortLinestrue    \LT@ShortPagestrue
45   \LT@OverfullLinestrue \LT@UnderfullLinestrue
46   \LT@Widowstrue        \LT@Orphanstrue
47   \LT@EOPHyphenstrue    \LT@RepeatedHyphenstrue
48   \LT@ParLastHyphentrue  \LT@EOLShortWordstrue
49   \LT@FirstWordMatchtrue \LT@LastWordMatchtrue
50   \LT@BackParindenttrue
51 }
52 \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.

```
53 \AtEndOfPackage{%
54   \ifLT@None
55     \directlua{ luatypo.None = true }%
56   \else
57     \directlua{ luatypo.None = false }%
58   \fi
59   \ifLT@BackParindent
60     \advance\luatypo@options by 1
61     \directlua{ luatypo.BackParindent = true }%
62   \else
63     \directlua{ luatypo.BackParindent = false }%
64   \fi
65   \ifLT@ShortLines
66     \advance\luatypo@options by 1
67     \directlua{ luatypo.ShortLines = true }%
68   \else
69     \directlua{ luatypo.ShortLines = false }%
70   \fi
71   \ifLT@ShortPages
72     \advance\luatypo@options by 1
73     \directlua{ luatypo.ShortPages = true }%
74   \else
75     \directlua{ luatypo.ShortPages = false }%
76   \fi
77   \ifLT@OverfullLines
78     \advance\luatypo@options by 1
79     \directlua{ luatypo.OverfullLines = true }%
80   \else
81     \directlua{ luatypo.OverfullLines = false }%
82   \fi
83   \ifLT@UnderfullLines
84     \advance\luatypo@options by 1
85     \directlua{ luatypo.UnderfullLines = true }%
86   \else
87     \directlua{ luatypo.UnderfullLines = false }%
88   \fi
89   \ifLT@Widows
90     \advance\luatypo@options by 1
```

```

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

```

ShowOptions is specific:

```

138 \ifLT@ShowOptions
139     \GenericWarning{* }{%
140         *** List of possible options for lua-typo ***\MessageBreak
141         [Default values between brackets]%
142         \MessageBreak

```

```

143 ShowOptions      [false]\MessageBreak
144 None             [false]\MessageBreak
145 BackParindent    [false]\MessageBreak
146 ShortLines       [false]\MessageBreak
147 ShortPages       [false]\MessageBreak
148 OverfullLines    [false]\MessageBreak
149 UnderfullLines   [false]\MessageBreak
150 Widows           [false]\MessageBreak
151 Orphans          [false]\MessageBreak
152 EOPHyphens       [false]\MessageBreak
153 RepeatedHyphens  [false]\MessageBreak
154 ParLastHyphen    [false]\MessageBreak
155 EOLShortWords    [false]\MessageBreak
156 FirstWordMatch   [false]\MessageBreak
157 LastWordMatch    [false]\MessageBreak
158 \MessageBreak
159 *****%
160 \MessageBreak Lua-typo [ShowOptions]
161 }%
162 \fi

```

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

```

163 \AtBeginDocument{%
164   \directlua{
165     luatypo.HYPHmax = tex.count.luatypoHyphMax
166     luatypo.PAGEmin = tex.count.luatypoPageMin
167     luatypo.Stretch = tex.count.luatypoStretchMax
168     luatypo.MinFull = tex.count.luatypoMinFull
169     luatypo.MinPart = tex.count.luatypoMinPart
170     luatypo.LLminWD = tex.dimen.luatypoLLminWD
171     luatypo.BackPI  = tex.dimen.luatypoBackPI
172     luatypo.BackFuzz = tex.dimen.luatypoBackFuzz
173   }%
174 }

```

Print the summary of offending pages —if any— at the (very) end of document unless option **None** has been selected.

```

175 \AtVeryEndDocument{%
176 \ifnum\luatypo@options = 0 \LT@Nonetrue \fi
177 \ifLT@None
178   \directlua{
179     texio.write_nl(' ')
180     texio.write_nl('*****')
181     texio.write_nl('*** lua-typo loaded with NO option:')
182     texio.write_nl('*** NO CHECK PERFORMED! ***')
183     texio.write_nl('*****')
184     texio.write_nl(' ')
185   }%
186 \else
187   \directlua{
188     texio.write_nl(' ')
189     texio.write_nl('*****')

```



```

190 if luatypo.pagelist == "" then
191     texio.write_nl('*** lua-typo: No Typo Flaws found.')
192 else
193     texio.write_nl('*** lua-typo: WARNING *****')
194     texio.write_nl('The following pages need attention: ')
195     texio.write(luatypo.pagelist)
196 end
197 texio.write_nl('*****')
198 texio.write_nl(' ')
199 }%
200 \fi}

```

`\luatypoOneChar` 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.

```

201 \newcommand*{\luatypoOneChar}[2]{%
202   \def\luatypo@LANG{#1}\luatypo@single={#2}%
203   \ifcsname l@\luatypo@LANG\endcsname
204     \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
205   \directlua{
206     local langno = \the\luatypo@LANGno
207     local string = \the\luatypo@single
208     luatypo.single[langno] = " "
209     for p, c in utf8.codes(string) do
210       local s = string.char(c)
211       luatypo.single[langno] = luatypo.single[langno] .. s
212     end
213     <dbg> texio.write_nl("SINGLE=" .. luatypo.single[langno])
214     <dbg> texio.write_nl(' ')
215   }%
216   \else
217     \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
218       \MessageBreak \protect\luatypoOneChar\space command ignored}%
219   \fi}
220 \newcommand*{\luatypoTwoChars}[2]{%
221   \def\luatypo@LANG{#1}\luatypo@double={#2}%
222   \ifcsname l@\luatypo@LANG\endcsname
223     \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
224   \directlua{
225     local langno = \the\luatypo@LANGno
226     local string = \the\luatypo@double
227     luatypo.double[langno] = " "
228     for p, c in utf8.codes(string) do
229       local s = string.char(c)
230       luatypo.double[langno] = luatypo.double[langno] .. s
231     end
232     <dbg> texio.write_nl("DOUBLE=" .. luatypo.double[langno])
233     <dbg> texio.write_nl(' ')
234   }%
235   \else
236     \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",

```

```

237         \MessageBreak \protect\luatypoTwoChars\space command ignored}%
238     \fi}

```

**\luatypoSetColor** This is a user-level command to customise the colours highlighting the nine 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).

```

239 \newcommand*{\luatypoSetColor}[2]{%
240     \begingroup
241         \color{#2}%
242         \directlua{luatypo.colortbl[#1]=\the\LuaCol@Attribute}%
243     \endgroup
244 }

```

The Lua code now, initialisations.

```

245 \begin{luacode}
246 luatypo.single = { }
247 luatypo.double = { }
248 luatypo.colortbl = { }
249 luatypo.pagelist = ""
250
251 local hyphcount = 0
252 local parlines = 0
253 local pagelines = 0
254 local pageno = 0
255 local prevno = 0
256 local pageflag = false
257
258 local char_to_discard = { }
259 char_to_discard[string.byte(",")] = true
260 char_to_discard[string.byte(".")] = true
261 char_to_discard[string.byte("!")] = true
262 char_to_discard[string.byte("?")] = true
263 char_to_discard[string.byte(":")] = true
264 char_to_discard[string.byte(";")] = true
265 char_to_discard[string.byte("-")] = true
266
267 local split_lig = { }
268 split_lig[0xFB00] = "ff"
269 split_lig[0xFB01] = "fi"
270 split_lig[0xFB02] = "fl"
271 split_lig[0xFB03] = "ffi"
272 split_lig[0xFB04] = "ffl"
273 split_lig[0xFB05] = "st"
274 split_lig[0xFB06] = "st"
275
276 local DISC = node.id("disc")
277 local GLYPH = node.id("glyph")
278 local GLUE = node.id("glue")
279 local KERN = node.id("kern")

```

```

280 local HLIST = node.id("hlist")
281 local LPAR = node.id("local_par")
282 local MKERN = node.id("margin_kern")
283 local PENALTY = node.id("penalty")
284 % \end{macrocode}
285 %   Glue subtypes:
286 %   \begin{macrocode}
287 local USRSKIP = 0
288 local PARSKIP = 3
289 local LFTSKIP = 8
290 local RGTSKIP = 9
291 local TOPSKIP = 10
292 local PARFILL = 15
293 % \end{macrocode}
294 %   Hlist subtypes:
295 %   \begin{macrocode}
296 local LINE = 1
297 local BOX = 2

```

Penalty subtypes:

```

298 local USER = 0
299 local HYPH = 0x2D

```

Glyph subtypes:

```

300 local LIGA = 0x102
301
302 local effective_glue = node.effective_glue
303 local set_attribute = node.set_attribute
304 local slide = node.slide
305 local traverse = node.traverse
306 local traverse_id = node.traverse_id
307 local has_field = node.has_field
308 local uses_font = node.uses_font
309 local is_glyph = node.is_glyph
310

```

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

```

311 local color_node = function (node, color)
312   local attr = oberdiek.luacolor.getattribute()
313   if node and node.id == DISC then
314     local pre = node.pre
315     local post = node.post
316     local repl = node.replace
317     if pre then
318       set_attribute(pre,attr,color)
319 <dbg> texio.write_nl('PRE=' .. tostring(pre.char))
320     end
321     if post then
322       set_attribute(post,attr,color)
323 <dbg> texio.write_nl('POST=' .. tostring(post.char))
324     end
325     if repl then

```

```

326         set_attribute(repl,attr,color)
327 <dbg> texio.write_nl('REPL=' .. tostring(repl.char))
328     end
329 <dbg> texio.write_nl(' ')
330 elseif node then
331     set_attribute(node,attr,color)
332 end
333 end

```

This auxillary function colours the content of an `\hbox`. It requires two arguments: a node (the box) and a (named) colour.

```

334 local color_hbox = function (head, color)
335     local first = head.head
336     for n in traverse(first) do
337         color_node(n, color)
338     end
339 end

```

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

```

340 local color_line = function (head, color)
341     local first = head.head
342     for n in traverse(first) do
343         if n and n.id == HLIST and n.subtype == BOX then
344             color_hbox(n, color)
345         else
346             color_node(n, color)
347         end
348     end
349 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`).

```

350 local signature = function (node, string, swap)
351     local n = node
352     local str = string
353     if n and n.id == GLYPH then
354         local b, id = is_glyph(n)
355         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`.

```

356         if b == 0x2019 then b = 0x27 end

```

```

357     if b < 0x100 then
358         str = str .. string.char(b)
359     elseif split_lig[b] then
360         local c = split_lig[b]
361         if swap then
362             c = string.reverse(c)
363         end
364         str = str .. c

```

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

```

365         elseif n.subtype == LIGA and b > 0xE000 then
366             local c = string.sub(b,-2)
367             if swap then
368                 c = string.reverse(c)
369             end
370             str = str .. c
371         end
372     end
373 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...

```

374     local pre = n.pre
375     local post = n.post
376     local c1 = ""
377     local c2 = ""
378     if pre and pre.char and pre.char ~= HYPH and pre.char < 0x100 then
379         c1 = string.char(pre.char)
380     end
381     if post and post.char then
382         if post.char < 0x100 then
383             c2 = string.char(post.char)
384         elseif split_lig[post.char] then
385             c2 = split_lig[post.char]
386             if swap then
387                 c2 = string.reverse(c2)
388             end
389         end
390     end
391     if swap then
392         str = str .. c2 .. c1
393     else
394         str = str .. c1 .. c2
395     end
396 end

```

The returned length is the number of *letters*.

```

397     local len = string.len(str)
398     if string.find(str, "_") then
399         len = len - 1
400     end
401     return len, str
402 end

```

This auxillary function looks for 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 (named) colour and a boolean to cancel the coloration in some cases (end of paragraphs). It prints the matching part at end of linewith with the supplied colour and returns the current line's last word and a boolean (match).

```

403 local check_last_word = function (old, node, color, flag)
404   local match = false
405   local new = ""
406   local maxlen = 0
407   if flag and node then
408     local swap = true

```

Step back to the last glyph or discretionary.

```

409     local lastn = node
410     while lastn and lastn.id ~= GLYPH and lastn.id ~= DISC do
411       lastn = lastn.prev
412     end

```

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

```

413     local n = lastn
414     while n and n.id ~= GLUE do
415       maxlen, new = signature (n, new, swap)
416       n = n.prev
417     end
418     if n and n.id == GLUE then
419       new = new .. "_"
420       repeat
421         n = n.prev
422         maxlen, new = signature (n, new, swap)
423       until not n or n.id == GLUE
424     end
425     new = string.reverse(new)
426 <dbg> texio.write_nl('EOLsigold=' .. old)
427 <dbg> texio.write('   EOLsig=' .. new)
428     local MinFull = luatypo.MinFull
429     local MinPart = luatypo.MinPart
430     MinFull = math.min(MinPart, MinFull)
431     local k = MinPart
432     local oldlast = string.gsub (old, '.*_', '')
433     local newlast = string.gsub (new, '.*_', '')
434     local i = string.find(new, "_")
435     if i and i > maxlen - MinPart + 1 then
436       k = MinPart + 1
437     end
438     local oldsub = string.sub(old, -k)
439     local newsub = string.sub(new, -k)
440     local l = string.len(new)
441     if oldsub == newsub and l >= k then
442 <dbg> texio.write_nl('EOLnewsub=' .. newsub)
443       match = true
444     elseif oldlast == newlast and string.len(newlast) >= MinFull then
445 <dbg> texio.write_nl('EOLnewlast=' .. newlast)
446       match = true

```

```

447         oldsub = oldlast
448         newsub = newlast
449         k = string.len(newlast)
450     end
451     if match then

```

Minimal partial match; any more glyphs matching?

```

452         local osub = oldsub
453         local nsub = newsub
454         while osub == nsub and k <= maxlen do
455             k = k + 1
456             osub = string.sub(old, -k)
457             nsub = string.sub(new, -k)
458             if osub == nsub then
459                 newsub = nsub
460             end
461         end
462         newsub = string.gsub(newsub, '^_', '')
463 <dbg>     texio.write_nl('EOLfullmatch=' .. newsub)
464 <msg>     texio.write_nl('***EOLMATCH=' .. newsub ..
465 <msg>                                     " on page " .. pageno)
466 <msg>     texio.write_nl(' ')

```

Lest's colour the matching string.

```

467         oldsub = string.reverse(newsub)
468         local newsub = ""
469         local n = lastn
470         repeat
471             if n and n.id ~= GLUE then
472                 color_node(n, color)
473                 l, newsub = signature(n, newsub, swap)
474             elseif n then
475                 newsub = newsub .. "_"
476             end
477             n = n.prev
478         until not n or newsub == oldsub or l >= k
479     end
480 end
481 return new, match
482 end

```

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

```

483 local check_first_word = function (old, node, color, flag)
484     local match = false
485     local swap = false
486     local new = ""
487     local maxlen = 0
488     local start = node
489     local n = start
490     while n and n.id ~= GLYPH and n.id ~= DISC do
491         n = n.next
492     end

```

```

493 while n and n.id ~= GLUE do
494     maxlen, new = signature (n, new, swap)
495     n = n.next
496 end
497 if n and n.id == GLUE then
498     new = new .. "_"
499     repeat
500         n = n.next
501         maxlen, new = signature (n, new, swap)
502     until not n or n.id == GLUE
503 end
504 <dbg> texio.write_nl('B0Lsigold=' .. old)
505 <dbg> texio.write('    B0Lsig=' .. 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.

```

506 if flag then
507     local MinFull = luatypo.MinFull
508     local MinPart = luatypo.MinPart
509     MinFull = math.min(MinPart, MinFull)
510     local k = MinPart
511     local oldsub = ""
512     local newsub = ""
513     local oldfirst = string.gsub (old, '_.*', '')
514     local newfirst = string.gsub (new, '_.*', '')
515     local i = string.find(new, "_")
516     if i and i <= MinPart then
517         k = MinPart + 1
518     end
519     local oldsub = string.sub(old, 1, k)
520     local newsub = string.sub(new, 1, k)
521     local l = string.len(newsub)
522     if oldsub == newsub and l >= k then
523 <dbg> texio.write_nl('B0Lnewsub=' .. newsub)
524         match = true
525     elseif oldfirst == newfirst and string.len(newfirst) >= MinFull then
526 <dbg> texio.write_nl('B0Lnewfirst=' .. newfirst)
527         match = true
528         oldsub = oldfirst
529         newsub = newfirst
530         k = string.len(newfirst)
531     end
532     if match then

```

Minimal partial match; any more glyphs matching?

```

533         local osub = oldsub
534         local nsub = newsub
535         while osub == nsub and k <= maxlen do
536             k = k + 1
537             osub = string.sub(old, 1, k)
538             nsub = string.sub(new, 1, k)
539             if osub == nsub then
540                 newsub = nsub
541             end

```



```

542     end
543     newsub = string.gsub(newsub, '_$', '') --$
544 <dbg> texio.write_nl('BOLfullmatch=' .. newsub)
545 <msg> texio.write_nl('***BOLMATCH=' .. newsub ..
546 <msg>                " on page " .. pageno)
547 <msg> texio.write_nl(' ')

```

Lest's colour the matching string.

```

548     oldsub = newsub
549     local newsub = ""
550     local k = string.len(oldsub)
551     local n = start
552     repeat
553         if n and n.id ~= GLUE then
554             color_node(n, color)
555             l, newsub = signature(n, newsub, swap)
556         elseif n then
557             newsub = newsub .. "_"
558         end
559         n = n.next
560     until not n or newsub == oldsub or l >= k
561     end
562 end
563 return new, match
564 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 argument must be a node of type GLYPH, usually the last line's node.

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

```

565 local check_regexpr = function (glyph)
566     local COLOR = luatypo.colortbl[3]
567     local lang = glyph.lang
568     local match = false
569     local lchar, id = is_glyph(glyph)
570     local previous = glyph.prev

```

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

```

571 if lang and luatypo.single[lang] then

```

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

```

572     if lchar and lchar < 0x100 and previous and previous.id == GLUE then
573         match = string.find(luatypo.single[lang], string.char(lchar))
574         if match then
575             color_node(glyph, COLOR)
576 <msg> texio.write_nl('***RGXMATCH=' .. string.char(lchar) ..
577 <msg>                " on page " .. pageno)
578 <msg> texio.write_nl(' ')
579         end
580     end
581 end

```

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

```

582 if lang and luatypo.double[lang] then
583     if lchar and previous and previous.id == GLYPH then
584         local pchar, id = is_glyph(previous)
585         local pprev = previous.prev

```

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

```

586         if pchar and pchar < 0x100 and pprev and pprev.id == GLUE then
587             local pattern = string.char(pchar) .. string.char(lchar)
588             match = string.find(luatypo.double[lang], pattern)
589             if match then
590                 color_node(previous, COLOR)
591                 color_node(glyph, COLOR)
592 <msg>         texio.write_nl('***RGXMATCH=' .. pattern ..
593 <msg>                        " on page " .. pageno)
594 <msg>         texio.write_nl(' ')
595             end
596         end

```

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

```

597     elseif lchar and previous and previous.id == KERN then
598         local pprev = previous.prev
599         if pprev and pprev.id == GLYPH then
600             local pchar, id = is_glyph(pprev)
601             local ppprev = pprev.prev
602             if pchar and pchar < 0x100 and ppprev and ppprev.id == GLUE then
603                 local pattern = string.char(pchar) .. string.char(lchar)
604                 match = string.find(luatypo.double[lang], pattern)
605                 if match then
606                     color_node(pprev, COLOR)
607                     color_node(glyph, COLOR)
608 <msg>         texio.write_nl('***RGXMATCH=' .. pattern ..
609 <msg>                        " on page " .. pageno)
610 <msg>         texio.write_nl(' ')
611                 end
612             end
613         end
614     end
615 end
616 return match
617 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.

```

618 local show_pre_disc = function (disc, color)
619     local n = disc
620     while n and n.id ~= GLUE do
621         color_node(n, color)
622         n = n.prev
623     end
624     return n
625 end

```

This is the main function which will be added to the "pre\_output\_filter" callback unless option None is selected.

```

626 luatypo.check_page = function (head)
627   local PAGEmin    = luatypo.PAGEmin
628   local HYPHmax    = luatypo.HYPHmax
629   local LLminWD    = luatypo.LLminWD
630   local BackPI     = luatypo.BackPI
631   local BackFuzz   = luatypo.BackFuzz
632   local BackParindent = luatypo.BackParindent
633   local ShortLines = luatypo.ShortLines
634   local ShortPages = luatypo.ShortPages
635   local OverfullLines = luatypo.OverfullLines
636   local UnderfullLines = luatypo.UnderfullLines
637   local Widows      = luatypo.Widows
638   local Orphans     = luatypo.Orphans
639   local EOPHyphens  = luatypo.EOPHyphens
640   local RepeatedHyphens = luatypo.RepeatedHyphens
641   local FirstWordMatch = luatypo.FirstWordMatch
642   local ParLastHyphen = luatypo.ParLastHyphen
643   local EOLShortWords = luatypo.EOLShortWords
644   local LastWordMatch = luatypo.LastWordMatch
645   local Stretch = math.max(luatypo.Stretch/100,1)
646
647   local orphanflag = false
648   local widowflag  = false
649   local lwhyphflag = false
650   local match1     = false
651   local match2     = false
652   local firstwd    = ""
653   local lastwd     = ""
654   while head do
655     local nextnode = head.next
656     local prevnode = head.prev
657     local pprevnode = nil
658     if prevnode then
659       pprevnode = prevnode.prev
660     end

```

If the current node is a glue of type topskip, we are starting new page, let's reset some counters and flags : `pageflag` will be set to `true` if a possible typographic flaw is found on this page, and trigger the addition of this page number to the summary list. `hyphcount` hold the number the consecutive hyphenated lines.

```

661   if head.id == GLUE and head.subtype == TOPSKIP then
662     pageno = tex.getcount("c@page")
663     hyphcount = 0
664     if pageno > prevno then
665       pageflag = false
666       pagelines = 0
667       match1 = false
668       firstwd = ""
669       lastwd = ""
670       prevno = pageno
671     end

```

```
672     elseif head.id == HLIST and head.subtype == LINE then
```

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

```
673         local first = head.head
674         if first.id == MKERN or
675            (first.id == GLUE and first.subtype == LFTSKIP) then
676             first = first.next
677         end
678         pagelines = pagelines + 1
679         local ListItem = false
```

Is this line really a text line (one glyph at least)?

```
680         local textline = false
681         if first.id == GLYPH then
682             textline = true
683         else
684             local n = first
685             repeat
686                 n = n.next
687                 if n and n.id == GLYPH then
688                     textline = true
689                     break
690             end
691             until not n or (n.id == GLUE and n.subtype == RGTSKIP)
692         end
```

Is this line overfull or underfull?

```
693         if head.glue_set == 1 and head.glue_sign == 2 and
694            head.glue_order == 0 and OverfullLines then
695 <msg> texio.write_nl('***OVERFULL line on page ' .. pageno)
696 <msg> texio.write_nl(' ')
697         pageflag = true
698         local COLOR = luatypo.colortbl[7]
699         color_line (head, COLOR)
700     elseif head.glue_set >= Stretch and head.glue_sign == 1 and
701            head.glue_order == 0 and UnderfullLines then
702 <msg> texio.write_nl('***UNDERFULL line on page ' ..
703 <msg>                                tex.getcount("c@page"))
704 <msg> texio.write_nl(' ')
705         local COLOR = luatypo.colortbl[8]
706         pageflag = true
707         color_line (head, COLOR)
708     end
```

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

```
709         if first.id == LPAR then
```

It starts a paragraph...

```
710             hyphcount = 0
711             parlines = 1
712             if not nextnode then
```

No more nodes: we are at the page bottom, this line is an orphan (unless it is the unique line of the paragraph)... see below.

```
713         orphanflag = true
714     end
```

List items begin with LPAR followed by an hbox.

```
715         local nn = first.next
716         if nn and nn.id == HLIST and nn.subtype == BOX then
717             ListItem = true
718         end
719     else
720         parlines = parlines + 1
721     end
```

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

```
722     if FirstWordMatch then
723         local flag = not ListItem
724         local COLOR = luatypo.colortbl[10]
725         firstwd, match1 = check_first_word(firstwd, first, COLOR, flag)
726         if match1 then
727             pageflag = true
728         end
729     end
```

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

```
730     local ln = slide(first)
731     local pn = ln.prev
732     if pn and pn.id == GLUE and pn.subtype == PARFILL then
```

The paragraph ends with this line, it is not an orphan then...

```
733         hyphcount = 0
734         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.

```
735         if pagelines == 1 and parlines > 1 then
736             widowflag = true
737         end
```

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

```
738         local PFskip = effective_glue(pn,head)
739         if ShortLines then
740             local llwd = tex.hsize - PFskip
741 <dbg>         local PFskip_pt = PFskip/65536
742 <dbg>         local llwd_pt = llwd/65536
743 <dbg>         texio.write_nl('PFskip= ' .. PFskip_pt .. ' pt')
744 <dbg>         texio.write_nl('llwd= ' .. llwd_pt .. ' pt')
```

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

```
745         if llwd < LLminWD then
746 <msg>         texio.write_nl('***Last line too short, page ' .. pageno)
747 <msg>         texio.write_nl(' ')
```

```

748             pageflag = true
749             local COLOR = luatypo.colortbl[6]
750             local attr = oberdiek.luacolor.getattribute()

```

let's colour the whole line.

```

751             color_line (head, COLOR)
752         end
753     end

```

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

```

754         if BackParindent and PFskip < BackPI and PFskip > BackFuzz then
755 <msg>         texio.write_nl('***Last line nearly full, page ' .. pageno)
756 <msg>         texio.write_nl(' ')
757             pageflag = true
758             local COLOR = luatypo.colortbl[12]
759             local attr = oberdiek.luacolor.getattribute()
760             color_line (head, COLOR)
761         end

```

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

```

762         if LastWordMatch then
763             local COLOR = luatypo.colortbl[11]
764             local flag = textline
765             if PFskip > BackPI then
766                 flag = false
767             end
768             lastwd, match1 = check_last_word(lastwd, pn, COLOR, flag)
769             if match1 then
770                 pageflag = true
771             end
772         end
773         elseif pn and pn.id == DISC then

```

The current line ends with an hyphen.

```

774             hyphcount = hyphcount + 1
775             if LastWordMatch then
776                 local COLOR = luatypo.colortbl[11]
777                 lastwd, match1 = check_last_word(lastwd, ln, COLOR, true)
778                 if match1 then
779                     pageflag = true
780                 end
781             end
782             if hyphcount > HYPHmax and RepeatedHyphens then
783                 local COLOR = luatypo.colortbl[2]
784                 local pg = show_pre_disc (pn,COLOR)
785                 pageflag = true
786 <msg>         texio.write_nl('***HYPH issue page ' .. pageno)
787 <msg>         texio.write_nl(' ')
788             end
789             if not nextnode and EOPHyphens then

```

No more nodes: this hyphen occurs on the page last line.

```

790             lwhyphflag = true

```

```

791         end
792         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`.

```

793             local nnnode = nextnode.next
794             local nnnnode = nil
795             if nnnode and nnnode.next then
796                 nnnnode = nnnode.next
797                 if nnnnode and nnnnode.id == HLIST and
798                     nnnnode.subtype == 1 and nnnnode.glue_order == 2 then
799                     local COLOR = luatypo.colortbl[0]
800                     local pg = show_pre_disc (pn,COLOR)
801                     pageflag = true
802                 end
803             end
804         end
805     elseif pn and pn.id == GLYPH then

```

The current line ends with a character, reset `hyphcount` and compare the end of line's word with the one on previous line.

```

806         hyphcount = 0
807         if LastWordMatch then
808             local COLOR = luatypo.colortbl[11]
809             lastwd, match1 = check_last_word(lastwd, pn, COLOR, true)
810         end

```

Look for a short unwanted short word at the end of the current line.

```

811         if EOLShortWords then
812             match2 = check_regexpr(pn)
813         end
814         if match1 or match2 then
815             pageflag = true
816         end

```

Microtype sometimes adds a margin kern at the right margin, `check_regexpr` has to operate on the previous node then.

```

817     elseif pn and pn.id == MKERN then
818         hyphcount = 0
819         local ppn = pn.prev
820         if ppn and ppn.id == GLYPH then
821             if LastWordMatch then
822                 local COLOR = luatypo.colortbl[11]
823                 lastwd, match1 = check_last_word(lastwd, pn, COLOR, true)
824             end
825             if EOLShortWords then
826                 match2 = check_regexpr(ppn)
827             end
828             if match1 or match2 then
829                 pageflag = true
830             end
831         end

```

If the current line ends with anything else (no hyphen), reset hyphcount.

```
832     else
833         hyphcount = 0
834     end
```

Colour the whole line if it is a widow.

```
835     if widowflag and Widows then
836         pageflag = true
837         widowflag = false
838 <msg> texio.write_nl('***WIDOW on top of page ' .. pageno)
839 <msg> texio.write_nl(' ')
840         local COLOR = luatypo.colortbl[4]
841         color_line (head, COLOR)
842     end
```

Colour the whole line if it is an orphan.

```
843     if orphanflag and Orphans then
844 <msg> texio.write_nl('***ORPHAN at bottom of page ' .. pageno)
845 <msg> texio.write_nl(' ')
846         pageflag = true
847         local COLOR = luatypo.colortbl[5]
848         color_line (head, COLOR)
849     end
```

Colour (differently) the last word if hyphenated.

```
850     if lwhyphflag and EOPHyphens then
851 <msg> texio.write_nl('***LAST WORD SPLIT page ' .. pageno)
852 <msg> texio.write_nl(' ')
853         pageflag = true
854         local COLOR = luatypo.colortbl[1]
855         local pg = show_pre_disc (pn,COLOR)
856     end
```

Track empty pages: check the number of lines at end of page. Widows are already detected; get the last line and colour it. NOTE: there are usually two consecutive nodes of type 12-0 at end of pages...

```
857     elseif not nextnode and head.id == GLUE and
858             head.subtype == USRSKIP then
859         if pagelines > 1 and pagelines < PAGEmin and ShortPages then
860             pageflag = true
861             local COLOR = luatypo.colortbl[9]
862 <msg> texio.write_nl('***Only ' .. pagelines ..
863 <msg> ' lines on page ' .. pageno)
864 <msg> texio.write_nl(' ')
865             local n = head
866             while n and (n.id ~= HLIST or n.subtype ~= LINE) do
867                 n = n.prev
868             end
869             if n then
870                 color_line(n, COLOR)
871             end
872         end
```



```

873     end
874     head = nextnode
875 end

```

Add this page number to the summary if any flaw has been found on it. Skip duplicates.

```

876 if pageflag then
877     local pl = luatypo.pagelist
878     local p = tonumber(string.match(pl, "%s(%d+),%s$"))
879     if not p or pageno > p then
880         luatypo.pagelist = luatypo.pagelist .. tostring(pageno) .. ", "
881     end
882 end
883 return true
884 end
885 return luatypo.check_page
886 \end{luacode}

```

Add the `luatypo.check_page` function to the `pre_output_filter` callback, unless option `None` is selected; remember that the `None` boolean's value is forwarded to Lua `'AtEndOfPackage'`...

```

887 \AtEndOfPackage{%
888     \directlua{
889         if not luatypo.None then
890             luatexbase.add_to_callback
891                 ("pre_output_filter", luatypo.check_page, "check_page")
892         end
893     }
894 }

```

Load a local config file if present in LaTeX's search path. Otherwise, set reasonable defaults.

```

895
896 \InputIfFileExists{lua-typo.cfg}%
897 {\PackageInfo{lua-typo.sty}{'lua-typo.cfg' file loaded}}%
898 {\PackageInfo{lua-typo.sty}{'lua-typo.cfg' file not found.
899     \MessageBreak Providing default values.}}%
900 \definecolor{mygrey}{gray}{0.6}%
901 \definecolor{myred}{rgb}{1,0.55,0}
902 \luatypoSetColor0{red}%      Paragraph last full line hyphenated
903 \luatypoSetColor1{red}%      Page last word hyphenated
904 \luatypoSetColor2{red}%      Hyphens ending two many consecutive lines
905 \luatypoSetColor3{red}%      Short word at end of line
906 \luatypoSetColor4{cyan}%     Widow
907 \luatypoSetColor5{cyan}%     Orphan
908 \luatypoSetColor6{cyan}%     Paragraph ending on a short line
909 \luatypoSetColor7{blue}%     Overfull lines
910 \luatypoSetColor8{blue}%     Underfull lines
911 \luatypoSetColor9{red}%      Nearly empty page
912 \luatypoSetColor{10}{myred}% First word matches
913 \luatypoSetColor{11}{myred}% Last word matches
914 \luatypoSetColor{12}{mygrey}% Paragraph ending on a nearly full line
915 \luatypoBackPI=1em\relax
916 \luatypoBackFuzz=2pt\relax

```

```
917 \luatypoLLminWD=2\parindent\relax
918 \luatypoStretchMax=200\relax
919 \luatypoHyphMax=2\relax
920 \luatypoPageMin=5\relax
921 \luatypoMinFull=4\relax
922 \luatypoMinPART=4\relax
923 }%
```

## 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
\luatypoLLminWD=2\parindent\relax

%% 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 ending two 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)
\luatypoSetColor10{myred}   % First word matches
\luatypoSetColor11{myred}   % Last word matches
\luatypoSetColor12{mygrey}  % Paragraph ending on a nearly full line

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