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 overfull or underfull lines, widows and orphans, hyphenated words split across two pages, two many 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 improved if possible. `lua-typo` also creates a `<jobname>.typo` file which summarises the informations (type, page, line number) about the detected issues.

Important notice: a) the highlighted lines are only meant to *draw the proofreader's attention* on possible issues, it is up to him/her to decide whether an improvement is desirable or not; they should *not* be regarded as blamable! some issues may be acceptable in some conditions (multi-columns, technical papers) and unbearable in others (literary works f.i.). Moreover, correcting a potential issue somewhere may result in other much more serious flaws somewhere else …
b) Conversely, possible bugs in `lua-typo` might hide issues that should normally be highlighted. Starting with version 0.85, the `<jobname>.typo` file lists, if any, the pages on which no text line could be found. The warning may be irrelevant (page only composed of figures) or point out a possible bug.

`lua-typo` is highly configurable in order to meet the variable expectations of authors and correctors: see the options' list and the `lua-typo.cfg` configuration file below.

When `lua-typo` shows possible flaws in the page layout, how can we fix them? The simpliest way is to rephrase some bits of text… this is an option for an author, not for a proofreader. When the text can not be altered, it is possible to *slightly* adjust the inter-word spacing (via the TeX commands `\spaceskip` and `\xspaceskip`) and/or the letter spacing (via `microtype`'s `\textls` command): slightly enlarging either of them or both 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 them 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]`.

[1]The file described in this section has version number v.0.86 and was last revised on 2024-01-12.

Version 0.85 requires a LaTeX kernel dated 2022/06/01 or later. Another “rollback” version `[=v0.65]` has been added for those who run an older kernel.

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 and to Michel Bovani whose contributions led to version 0.61!

2 Usage

The easiest way to trigger all checks perfomed 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}
```
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 *nearly* 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?
`ShortFinalWord`	Short word ending a sentence on the next page
`MarginparPos`	Margin note ending too low on the page

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.

`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 hightlighted.

`ShortFinalWord :` the first word on a page is highlighted if it ends a sentence and is short (up to `\luatypoMinLen=4` letters).

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 end at at sufficient distance (`\luatypoBackPI`, default `1em`) of the right margin, or (approximately) touch the right margin —the tolerance is `\luatypoBackFuzz` (default `2pt`) [2].

`ShortLines: \luatypoLLminWD=2\parindent` [3] 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

[2]Some authors do not accept full lines at end of paragraphs, they can just set `\luatypoBackFuzz=0pt` to make them pointed out as faulty.

[3]Or `20pt` if `\parindent=0pt`.

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 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 occurence 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 Ô Y'}`
`\luatypoTwoChars{french}{'Je Tu Il On Au De'}`

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.

`\luatypoMarginparTol` is a *dimension* which defaults to `\baselineskip`; marginal notes trigger a flaw if they end lower than `\luatypoMarginparTol` under the page's last line.

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

```
% \definecolor{LTgrey}{gray}{0.6}
% \definecolor{LTred}{rgb}{1,0.55,0}
% \definecolor{LTline}{rgb}{0.7,0,0.3}
% \luatypoSetColor1{red}      % Paragraph last full line hyphenated
% \luatypoSetColor2{red}      % Page last word hyphenated
% \luatypoSetColor3{red}      % Hyphens on consecutive lines
% \luatypoSetColor4{red}      % Short word at end of line
% \luatypoSetColor5{cyan}     % Widow
% \luatypoSetColor6{cyan}     % Orphan
% \luatypoSetColor7{cyan}     % Paragraph ending on a short line
% \luatypoSetColor8{blue}     % Overfull lines
% \luatypoSetColor9{blue}     % Underfull lines
% \luatypoSetColor{10}{red}   % Nearly empty page (a few lines)
% \luatypoSetColor{11}{LTred} % First word matches
% \luatypoSetColor{12}{LTred} % Last word matches
% \luatypoSetColor{13}{LTgrey}% Paragraph’s last line nearly full
% \luatypoSetColor{14}{cyan}  % Footnotes spread over two pages
% \luatypoSetColor{15}{red}   % Short final word on top of the page
% \luatypoSetColor{16}{LTline}% Line color for multiple flaws
% \luatypoSetColor{17}{red}   % Margin note ending too low
```

`lua-typo` loads the `luacolor` package which loads the `color` package from the LaTeX graphic bundle. `\luatypoSetColor` requires named colours, so you can either use the `\definecolor` from `color` package to define yours (as done in the config file for ‘LTgrey’ and ‘LTred’) 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. Rollback version 0.65 is provided for users who would have a LaTeX kernel older than 2022/06/01.

```
\DeclareRelease{v0.4}{2021-01-01}{lua-typo-2021-04-18.sty}
\DeclareRelease{v0.65}{2023-03-08}{lua-typo-2023-03-08.sty}
\DeclareCurrentRelease{}{2023-09-13}
```

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

```
\ifdefined\directlua
  \RequirePackage{luatexbase,luacode,luacolor,atveryend}
\else
  \PackageError{This package is meant for LuaTeX only! Aborting}
               {No more information available, sorry!}
\fi
```

Let’s define the necessary internal counters, dimens, token registers and commands…

```
\newdimen\luatypoLLminWD
\newdimen\luatypoBackPI
\newdimen\luatypoBackFuzz
\newdimen\luatypoMarginparTol
\newcount\luatypoStretchMax
\newcount\luatypoHyphMax
\newcount\luatypoPageMin
\newcount\luatypoMinFull
\newcount\luatypoMinPart
\newcount\luatypoMinLen
\newcount\luatypo@LANGno
```

```
\newcount\luatypo@options
\newtoks\luatypo@single
\newtoks\luatypo@double
```

… and define a global table for this package.

```
\begin{luacode}
luatypo = { }
\end{luacode}
```

Set up `ltkeys` initializations. Option `All` resets all booleans relative to specific typographic checks to `true`.

```
\DeclareKeys[luatypo]
  {
   ShowOptions.if      = LT@ShowOptions      ,
   None.if             = LT@None             ,
   BackParindent.if    = LT@BackParindent    ,
   ShortLines.if       = LT@ShortLines       ,
   ShortPages.if       = LT@ShortPages       ,
   OverfullLines.if    = LT@OverfullLines    ,
   UnderfullLines.if   = LT@UnderfullLines   ,
   Widows.if           = LT@Widows           ,
   Orphans.if          = LT@Orphans          ,
   EOPHyphens.if       = LT@EOPHyphens       ,
   RepeatedHyphens.if  = LT@RepeatedHyphens  ,
   ParLastHyphen.if    = LT@ParLastHyphen    ,
   EOLShortWords.if    = LT@EOLShortWords    ,
   FirstWordMatch.if   = LT@FirstWordMatch   ,
   LastWordMatch.if    = LT@LastWordMatch    ,
   FootnoteSplit.if    = LT@FootnoteSplit    ,
   ShortFinalWord.if   = LT@ShortFinalWord   ,
   MarginparPos.if     = LT@MarginparPos     ,
   All.if              = LT@All              ,
   All.code            = \LT@ShortLinestrue     \LT@ShortPagestrue
                         \LT@OverfullLinestrue  \LT@UnderfullLinestrue
                         \LT@Widowstrue         \LT@Orphanstrue
                         \LT@EOPHyphenstrue     \LT@RepeatedHyphenstrue
                         \LT@ParLastHyphentrue  \LT@EOLShortWordstrue
                         \LT@FirstWordMatchtrue \LT@LastWordMatchtrue
                         \LT@BackParindenttrue  \LT@FootnoteSplittrue
                         \LT@ShortFinalWordtrue \LT@MarginparPostrue
  }
\ProcessKeyOptions[luatypo]
```

Forward these options to the `luatypo` global table. Wait until the config file `luatypo.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.

```
\AtEndOfPackage{%
  \ifLT@None
    \directlua{ luatypo.None = true }%
  \else
    \directlua{ luatypo.None = false }%
  \fi
  \ifLT@BackParindent
    \advance\luatypo@options by 1
    \directlua{ luatypo.BackParindent = true }%
  \else
    \directlua{ luatypo.BackParindent = false }%
  \fi
  \ifLT@ShortLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.ShortLines = true }%
  \else
    \directlua{ luatypo.ShortLines = false }%
  \fi
  \ifLT@ShortPages
    \advance\luatypo@options by 1
    \directlua{ luatypo.ShortPages = true }%
  \else
    \directlua{ luatypo.ShortPages = false }%
  \fi
  \ifLT@OverfullLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.OverfullLines = true }%
  \else
    \directlua{ luatypo.OverfullLines = false }%
  \fi
  \ifLT@UnderfullLines
    \advance\luatypo@options by 1
    \directlua{ luatypo.UnderfullLines = true }%
  \else
    \directlua{ luatypo.UnderfullLines = false }%
  \fi
  \ifLT@Widows
    \advance\luatypo@options by 1
    \directlua{ luatypo.Widows = true }%
  \else
    \directlua{ luatypo.Widows = false }%
  \fi
  \ifLT@Orphans
    \advance\luatypo@options by 1
    \directlua{ luatypo.Orphans = true }%
  \else
    \directlua{ luatypo.Orphans = false }%
  \fi
  \ifLT@EOPHyphens
    \advance\luatypo@options by 1
    \directlua{ luatypo.EOPHyphens = true }%
  \else
    \directlua{ luatypo.EOPHyphens = false }%
  \fi
  \ifLT@RepeatedHyphens
    \advance\luatypo@options by 1
    \directlua{ luatypo.RepeatedHyphens = true }%
  \else
    \directlua{ luatypo.RepeatedHyphens = false }%
  \fi
  \ifLT@ParLastHyphen
```

```
    \advance\luatypo@options by 1
    \directlua{ luatypo.ParLastHyphen = true }%
  \else
    \directlua{ luatypo.ParLastHyphen = false }%
  \fi
  \ifLT@EOLShortWords
    \advance\luatypo@options by 1
    \directlua{ luatypo.EOLShortWords = true }%
  \else
    \directlua{ luatypo.EOLShortWords = false }%
  \fi
  \ifLT@FirstWordMatch
    \advance\luatypo@options by 1
    \directlua{ luatypo.FirstWordMatch = true }%
  \else
    \directlua{ luatypo.FirstWordMatch = false }%
  \fi
  \ifLT@LastWordMatch
    \advance\luatypo@options by 1
    \directlua{ luatypo.LastWordMatch = true }%
  \else
    \directlua{ luatypo.LastWordMatch = false }%
  \fi
  \ifLT@FootnoteSplit
    \advance\luatypo@options by 1
    \directlua{ luatypo.FootnoteSplit = true }%
  \else
    \directlua{ luatypo.FootnoteSplit = false }%
  \fi
  \ifLT@ShortFinalWord
    \advance\luatypo@options by 1
    \directlua{ luatypo.ShortFinalWord = true }%
  \else
    \directlua{ luatypo.ShortFinalWord = false }%
  \fi
  \ifLT@MarginparPos
    \advance\luatypo@options by 1
    \directlua{ luatypo.MarginparPos = true }%
  \else
    \directlua{ luatypo.MarginparPos = false }%
  \fi
}
```

ShowOptions is specific:

```
\ifLT@ShowOptions
  \GenericWarning{* }{%
     *** List of possible options for lua-typo ***\MessageBreak
     [Default values between brackets]%
     \MessageBreak
     ShowOptions     [false]\MessageBreak
     None            [false]\MessageBreak
     All             [false]\MessageBreak
     BackParindent   [false]\MessageBreak
     ShortLines      [false]\MessageBreak
```

```
    ShortPages      [false]\MessageBreak
    OverfullLines   [false]\MessageBreak
    UnderfullLines  [false]\MessageBreak
    Widows          [false]\MessageBreak
    Orphans         [false]\MessageBreak
    EOPHyphens      [false]\MessageBreak
    RepeatedHyphens [false]\MessageBreak
    ParLastHyphen   [false]\MessageBreak
    EOLShortWords   [false]\MessageBreak
    FirstWordMatch  [false]\MessageBreak
    LastWordMatch   [false]\MessageBreak
    FootnoteSplit   [false]\MessageBreak
    ShortFinalWord  [false]\MessageBreak
    MarginparPos    [false]\MessageBreak
    \MessageBreak
    **********************************************%
    \MessageBreak Lua-typo [ShowOptions]
  }%
\fi
```

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

```
\AtBeginDocument{%
  \directlua{
    luatypo.HYPHmax = tex.count.luatypoHyphMax
    luatypo.PAGEmin = tex.count.luatypoPageMin
    luatypo.Stretch = tex.count.luatypoStretchMax
    luatypo.MinFull = tex.count.luatypoMinFull
    luatypo.MinPart = tex.count.luatypoMinPart
```

Ensure `MinFull`≤`MinPart`.

```
    luatypo.MinFull  = math.min(luatypo.MinPart,luatypo.MinFull)
    luatypo.MinLen   = tex.count.luatypoMinLen
    luatypo.LLminWD  = tex.dimen.luatypoLLminWD
    luatypo.BackPI   = tex.dimen.luatypoBackPI
    luatypo.BackFuzz = tex.dimen.luatypoBackFuzz
    luatypo.MParTol  = tex.dimen.luatypoMarginparTol
```

Build a compact table holding all colours defined by `lua-typo` (no duplicates).

```
    local tbl = luatypo.colortbl
    local map = { }
    for i,v in ipairs (luatypo.colortbl) do
      if i == 1 or v > tbl[i-1] then
         table.insert(map, v)
      end
    end
    luatypo.map = map
  }%
}
```

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.

On every page, at least one line of text should be found. Otherwise, `lua-typo` presumes something went wrong and writes the page number to a `failedlist` list. In case `pagelist` is empty *and* `failedlist` *is not*, a warning is issued instead of the `No Typo Flaws found.` message (new to version 0.85).

```
\AtVeryEndDocument{%
\ifnum\luatypo@options = 0 \LT@Nonetrue \fi
\ifLT@None
  \directlua{
    texio.write_nl(' ')
    texio.write_nl('***********************************')
    texio.write_nl('*** lua-typo loaded with NO option:')
    texio.write_nl('*** NO CHECK PERFORMED! ***')
    texio.write_nl('***********************************')
    texio.write_nl(' ')
   }%
\else
  \directlua{
    texio.write_nl(' ')
    texio.write_nl('*************************************')
    if luatypo.pagelist == " " then
       if luatypo.failedlist == " " then
          texio.write_nl('*** lua-typo: No Typo Flaws found.')
       else
          texio.write_nl('*** WARNING: ')
          texio.write('lua-typo failed to scan these pages:')
          texio.write_nl('***' .. luatypo.failedlist)
          texio.write_nl('*** Please report to the maintainer.')
       end
    else
       texio.write_nl('*** lua-typo: WARNING *************')
       texio.write_nl('The following pages need attention:')
       texio.write(luatypo.pagelist)
    end
    texio.write_nl('*************************************')
    texio.write_nl(' ')
    if luatypo.failedlist == " " then
    else
       local prt = "WARNING: lua-typo failed to scan pages " ..
                   luatypo.failedlist .. "\string\n\string\n"
       luatypo.buffer = prt .. luatypo.buffer
    end
    local fileout= tex.jobname .. ".typo"
    local out=io.open(fileout,"w+")
    out:write(luatypo.buffer)
    io.close(out)
   }%
\fi}
```

`\luatypoOneChar` `\luatypoTwoChars` 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 utf-8 string entered as second argument has to be converted into the font internal coding.

```
\newcommand*{\luatypoOneChar}[2]{%
  \def\luatypo@LANG{#1}\luatypo@single={#2}%
  \ifcsname l@\luatypo@LANG\endcsname
    \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
    \directlua{
      local langno = \the\luatypo@LANGno
      local string = \the\luatypo@single
      luatypo.single[langno] = " "
      for p, c in utf8.codes(string) do
        local s = utf8.char(c)
        luatypo.single[langno] = luatypo.single[langno] .. s
      end
⟨dbg⟩      texio.write_nl('SINGLE=' .. luatypo.single[langno])
⟨dbg⟩      texio.write_nl(' ')
     }%
  \else
    \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
       \MessageBreak \protect\luatypoOneChar\space command ignored}%
  \fi}
\newcommand*{\luatypoTwoChars}[2]{%
  \def\luatypo@LANG{#1}\luatypo@double={#2}%
  \ifcsname l@\luatypo@LANG\endcsname
    \luatypo@LANGno=\the\csname l@\luatypo@LANG\endcsname \relax
    \directlua{
      local langno = \the\luatypo@LANGno
      local string = \the\luatypo@double
      luatypo.double[langno] = " "
      for p, c in utf8.codes(string) do
        local s = utf8.char(c)
        luatypo.double[langno] = luatypo.double[langno] .. s
      end
⟨dbg⟩      texio.write_nl('DOUBLE=' .. luatypo.double[langno])
⟨dbg⟩      texio.write_nl(' ')
    }%
  \else
    \PackageWarning{luatypo}{Unknown language "\luatypo@LANG",
       \MessageBreak \protect\luatypoTwoChars\space command ignored}%
  \fi}
```

`\luatypoSetColor` This is a user-level command to customise the colours highlighting the sixteen types of possible typographic flaws. The first argument is a number (flaw type: 1-16), the second the named colour associated to it. The colour support is based on the `luacolor` package (colour attributes).

```
\newcommand*{\luatypoSetColor}[2]{%
  \begingroup
    \color{#2}%
    \directlua{luatypo.colortbl[#1]=\the\LuaCol@Attribute}%
  \endgroup
}
%\luatypoSetColor{0}{black}
```

The Lua code now, initialisations.

```
\begin{luacode}
luatypo.colortbl   = { }
luatypo.map        = { }
luatypo.single     = { }
luatypo.double     = { }
luatypo.pagelist   = " "
luatypo.failedlist = " "
luatypo.buffer     = "List of typographic flaws found for "
                      .. tex.jobname .. ".pdf:\string\n\string\n"

local char_to_discard = { }
char_to_discard[string.byte(",")] = true
char_to_discard[string.byte(".")] = true
char_to_discard[string.byte("!")] = true
char_to_discard[string.byte("?")] = true
char_to_discard[string.byte(":")] = true
char_to_discard[string.byte(";")] = true
char_to_discard[string.byte("-")] = true

local eow_char = { }
eow_char[string.byte(".")] = true
eow_char[string.byte("!")] = true
eow_char[string.byte("?")] = true
eow_char[utf8.codepoint("…")] = true

local DISC  = node.id("disc")
local GLYPH = node.id("glyph")
local GLUE  = node.id("glue")
local KERN  = node.id("kern")
local RULE  = node.id("rule")
local HLIST = node.id("hlist")
local VLIST = node.id("vlist")
local LPAR  = node.id("local_par")
local MKERN = node.id("margin_kern")
local PENALTY = node.id("penalty")
local WHATSIT = node.id("whatsit")
```

Glue subtypes:

```
local USRSKIP  = 0
local PARSKIP  = 3
local LFTSKIP  = 8
local RGTSKIP  = 9
local TOPSKIP = 10
local PARFILL = 15
```

Hlist subtypes:

```
local LINE    = 1
local BOX     = 2
local INDENT  = 3
local ALIGN   = 4
local EQN     = 6
```

Penalty subtypes:

```
local USER = 0
local HYPH = 0x2D
```

Glyph subtypes:

```
local LIGA = 0x102
```

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

```
local parline = 0
```

Local definitions for the 'node' library:

```
local dimensions = node.dimensions
local rangedimensions = node.rangedimensions
local effective_glue = node.effective_glue
local set_attribute = node.set_attribute
local get_attribute = node.get_attribute
local slide = node.slide
local traverse = node.traverse
local traverse_id = node.traverse_id
local has_field = node.has_field
local uses_font = node.uses_font
local is_glyph  = node.is_glyph
local utf8_len  = utf8.len
```

Local definitions from the 'unicode.utf8' library: replacements are needed for functions `string.gsub()`, `string.sub()`, `string.find()` and `string.reverse()` which are meant for one-byte characters only.

`utf8_find` requires an utf-8 string and a 'pattern' (also utf-8), it returns `nil` if pattern is not found, or the *byte* position of the first match otherwise [not an issue as we only care for true/false].

```
local utf8_find = unicode.utf8.find
```

`utf8_gsub` mimics `string.gsub` for utf-8 strings.

```
local utf8_gsub = unicode.utf8.gsub
```

`utf8_reverse` returns the reversed string (utf-8 chars read from end to beginning) [same as `string.reverse` but for utf-8 strings].

```
local utf8_reverse = function (s)
  if utf8_len(s) > 1 then
     local so = ""
     for p, c in utf8.codes(s) do
         so = utf8.char(c) .. so
     end
     s = so
  end
  return s
end
```

`utf8_sub` returns the substring of s that starts at i and continues until j (j-i-1 utf8 chars.). *Warning: it requires* $i \geq 1$ *and* $j \geq i$.

```
local utf8_sub = function (s,i,j)
```

```
    i=utf8.offset(s,i)
    j=utf8.offset(s,j+1)-1
    return string.sub(s,i,j)
end
```

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

```
local color_node = function (node, color)
  local attr = oberdiek.luacolor.getattribute()
  if node and node.id == DISC then
     local pre = node.pre
     local post = node.post
     local repl = node.replace
     if pre then
        set_attribute(pre,attr,color)
     end
     if post then
        set_attribute(post,attr,color)
     end
     if repl then
        set_attribute(repl,attr,color)
     end
  elseif node then
     set_attribute(node,attr,color)
  end
end
```

The next function colours a whole line without overriding previously set colours by f.i. homeoarchy, repeated hyphens etc. 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.

```
local color_line = function (head, color)
  local first = head.head
  local map = luatypo.map
  local color_node_if = function (node, color)
    local c = oberdiek.luacolor.getattribute()
    local att = get_attribute(node,c)
    local uncolored = true
    for i,v in ipairs (map) do
      if att == v then
         uncolored = false
         break
      end
    end
    if uncolored then
       color_node (node, color)
    end
  end
  for n in traverse(first) do
      if n.id == HLIST or n.id == VLIST then
         local ff = n.head
         for nn in traverse(ff) do
           if nn.id == HLIST or nn.id == VLIST then
```

```
              local f3 = nn.head
              for n3 in traverse(f3) do
                if n3.id == HLIST or n3.id == VLIST then
                   local f4 = n3.head
                   for n4 in traverse(f4) do
                     if n4.id == HLIST or n4.id == VLIST then
                        local f5 = n4.head
                        for n5 in traverse(f5) do
                          if n5.id == HLIST or n5.id == VLIST then
                             local f6 = n5.head
                             for n6 in traverse(f6) do
                               color_node_if(n6, color)
                             end
                          else
                             color_node_if(n5, color)
                          end
                        end
                     else
                        color_node_if(n4, color)
                     end
                   end
                else
                   color_node_if(n3, color)
                end
              end
           else
              color_node_if(nn, color)
           end
         end
      else
         color_node_if(n, color)
      end
  end
end
```

The next function takes four arguments: a string, two numbers (which can be NIL) and a flag. It appends a line to a buffer which will be written to file '`\jobname.typo`'.

```
log_flaw= function (msg, line, colno, footnote)
  local pageno = tex.getcount("c@page")
  local prt ="p. " .. pageno
  if colno then
     prt = prt .. ", col." .. colno
  end
  if line then
     local l = string.format("%2d, ", line)
     if footnote then
        prt = prt .. ", (ftn.) line " .. l
     else
        prt = prt .. ", line " .. l
     end
  end
  prt =  prt .. msg
  luatypo.buffer = luatypo.buffer .. prt .. "\string\n"
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, dicretionnaries other than ligatures, kerns (letterspacing) should be discarded. For each word to be compared we build a “signature” made of glyphs, split ligatures and underscores (representing glues).

The first function adds a (non-nil) node to a signature of type string, nil nodes are ignored. It returns the augmented string and its length (underscores are omitted in the length computation). The last argument is a boolean needed when building a signature backwards (see `check_line_last_word`).

```
local signature = function (node, string, swap)
  local n = node
  local str = string
  if n and n.id == GLYPH then
     local b = n.char
```

Punctuation has to be discarded; other glyphs may be ligatures, then they have a `components` field which holds the list of glyphs which compose the ligature.

```
     if b and not char_to_discard[b] then
        if n.components then
           local c = ""
           for nn in traverse_id(GLYPH, n.components) do
             c = c .. utf8.char(nn.char)
           end
           if swap then
              str = str .. utf8_reverse(c)
           else
              str = str .. c
           end
        else
           str = str .. utf8.char(b)
        end
     end
  elseif n and n.id == DISC then
```

Discretionaries are split into `pre` and `post` and both parts are stored. They might be ligatures (*ffl*, *ffi*)…

```
    local pre = n.pre
    local post = n.post
    local c1 = ""
    local c2 = ""
    if pre and pre.char then
       if pre.components then
          for nn in traverse_id(GLYPH, pre.components) do
            c1 = c1 .. utf8.char(nn.char)
          end
       else
          c1 = utf8.char(pre.char)
       end
       c1 = utf8_gsub(c1, "-", "")
    end
    if post and post.char then
       if post.components then
```

```
          for nn in traverse_id(GLYPH, post.components) do
            c2 = c2 .. utf8.char(nn.char)
          end
       else
          c2 = utf8.char(post.char)
       end
    end
    if swap then
       str = str .. utf8_reverse(c2) .. c1
    else
       str = str .. c1 .. c2
    end
  elseif n and n.id == GLUE then
       str = str .. "_"
  end
```

The returned length is the number of *letters*.

```
  local s = utf8_gsub(str, "_", "")
  local len = utf8_len(s)
  return len, str
end
```

The next function looks for consecutive lines ending with the same letters.

It requires five arguments: a string (previous line's signature), a node (the last one on the current line), a line number, a column number (possibly `nil`) and a boolean to cancel checking 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).

```
local check_line_last_word =
            function (old, node, line, colno, flag, footnote)
  local COLOR = luatypo.colortbl[12]
  local match = false
  local new = ""
  local maxlen = 0
  local MinFull = luatypo.MinFull
  local MinPart = luatypo.MinPart
  if node then
     local swap = true
     local box, go
```

Step back to the last glyph or discretionary or hbox.

```
     local lastn = node
     while lastn and lastn.id ~= GLYPH and lastn.id ~= DISC and
           lastn.id ~= HLIST do
       lastn = lastn.prev
     end
```

A signature is built from the last two (or more) words on the current line.

```
     local n = lastn
     local words = 0
     while n and (words <= 2 or maxlen < MinPart) do
```

Go down inside boxes, read their content from end to beginning, then step out.

```
      if n and n.id == HLIST then
         box = n
         local first = n.head
         local lastn = slide(first)
         n = lastn
         while n do
           maxlen, new = signature (n, new, swap)
           n = n.prev
         end
         n = box.prev
         local w = utf8_gsub(new, "_", "")
         words = words + utf8_len(new) - utf8_len(w) + 1
      else
         repeat
           maxlen, new = signature (n, new, swap)
           n = n.prev
         until not n or n.id == GLUE or n.id == HLIST
         if n and n.id == GLUE then
            maxlen, new = signature (n, new, swap)
            words = words + 1
            n = n.prev
         end
      end
    end
    new = utf8_reverse(new)
    new = utf8_gsub(new, "_+$", "")  -- $
    new = utf8_gsub(new, "^_+", "")
    maxlen = math.min(utf8_len(old), utf8_len(new))
⟨dbg⟩     texio.write_nl('EOLsigold=' .. old)
⟨dbg⟩     texio.write('   EOLsig=' .. new)
```

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

```
    if flag and old ~= "" then
```

`oldlast` and `newlast` hold the last (full) words to be compared later:

```
       local oldlast = utf8_gsub (old, ".*_", "")
       local newlast = utf8_gsub (new, ".*_", "")
```

Let's look for a partial match: build `oldsub` and `newsub`, reading (backwards) the last `MinPart` *non-space* characters of both lines.

```
       local oldsub = ""
       local newsub = ""
       local dlo = utf8_reverse(old)
       local wen = utf8_reverse(new)
       for p, c in utf8.codes(dlo) do
         local s = utf8_gsub(oldsub, "_", "")
         if utf8_len(s) < MinPart then
            oldsub = utf8.char(c) .. oldsub
         end
       end
       for p, c in utf8.codes(wen) do
```

```
      local s = utf8_gsub(newsub, "_", "")
      if utf8_len(s) < MinPart then
         newsub = utf8.char(c) .. newsub
      end
    end
    if oldsub == newsub then
⟨dbg⟩       texio.write_nl('EOLnewsub=' .. newsub)
       match = true
    end
    if oldlast == newlast and utf8_len(newlast) >= MinFull then
⟨dbg⟩       texio.write_nl('EOLnewlast=' .. newlast)
       if utf8_len(newlast) > MinPart or not match then
          oldsub = oldlast
          newsub = newlast
       end
       match = true
    end
    if match then
```

Minimal full or partial match `newsub` of length `k`; any more glyphs matching?

```
       local k = utf8_len(newsub)
       local osub = utf8_reverse(oldsub)
       local nsub = utf8_reverse(newsub)
       while osub == nsub and k < maxlen do
         k = k + 1
         osub = utf8_sub(dlo,1,k)
         nsub = utf8_sub(wen,1,k)
         if osub == nsub then
            newsub = utf8_reverse(nsub)
         end
       end
       newsub = utf8_gsub(newsub, "^_+", "")
⟨dbg⟩       texio.write_nl("EOLfullmatch=" .. newsub)
       local msg = "E.O.L. MATCH=" .. newsub
       log_flaw(msg, line, colno, footnote)
```

Lest's colour the matching string.

```
       local ns = utf8_gsub(newsub, "_", "")
       k = utf8_len(ns)
       oldsub = utf8_reverse(newsub)
       local newsub = ""
       local n = lastn
       local l = 0
       local lo = 0
       local li = 0
       while n and newsub ~= oldsub and l < k do
         if n and n.id == HLIST then
            local first = n.head
            for nn in traverse_id(GLYPH, first) do
              color_node(nn, COLOR)
              local c = nn.char
              if not char_to_discard[c] then l = l + 1 end
            end
⟨dbg⟩       texio.write_nl('l (box)=' .. l)
```

```
             elseif n then
                color_node(n, COLOR)
                li, newsub = signature(n, newsub, swap)
                l = l + li - lo
                lo = li
⟨dbg⟩           texio.write_nl('l=' .. l)
             end
             n = n.prev
           end
        end
     end
  end
  return new, match
end
```

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

```
local check_line_first_word =
            function (old, node, line, colno, flag, footnote)
  local COLOR = luatypo.colortbl[11]
  local match = false
  local swap = false
  local new = ""
  local maxlen = 0
  local MinFull = luatypo.MinFull
  local MinPart = luatypo.MinPart
  local n = node
  local box, go
  while n and n.id ~= GLYPH and n.id ~= DISC and
        (n.id ~= HLIST or n.subtype == INDENT) do
     n = n.next
  end
  start = n
  local words = 0
  while n and (words <= 2 or maxlen < MinPart) do
    if n and n.id == HLIST then
       box = n
       n = n.head
       while n do
         maxlen, new = signature (n, new, swap)
         n = n.next
       end
       n = box.next
       local w = utf8_gsub(new, "_", "")
       words = words + utf8_len(new) - utf8_len(w) + 1
    else
       repeat
         maxlen, new = signature (n, new, swap)
         n = n.next
       until not n or n.id == GLUE or n.id == HLIST
       if n and n.id == GLUE then
          maxlen, new = signature (n, new, swap)
          words = words + 1
          n = n.next
       end
    end
  end
  new = utf8_gsub(new, "_+$", "") -- $
  new = utf8_gsub(new, "^_+", "")
  maxlen = math.min(utf8_len(old), utf8_len(new))
  ⟨dbg⟩  texio.write_nl('BOLsigold=' .. old)
  ⟨dbg⟩  texio.write('   BOLsig=' .. new)
```

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

```
if flag and old ~= "" then
   local oldfirst = utf8_gsub (old, "_.*", "")
   local newfirst = utf8_gsub (new, "_.*", "")
   local oldsub = ""
   local newsub = ""
   for p, c in utf8.codes(old) do
     local s = utf8_gsub(oldsub, "_", "")
     if utf8_len(s) < MinPart then
        oldsub = oldsub .. utf8.char(c)
     end
   end
   for p, c in utf8.codes(new) do
     local s = utf8_gsub(newsub, "_", "")
     if utf8_len(s) < MinPart then
        newsub = newsub .. utf8.char(c)
     end
   end
   if oldsub == newsub then
⟨dbg⟩        texio.write_nl('BOLnewsub=' .. newsub)
      match = true
   end
   if oldfirst == newfirst  and utf8_len(newfirst) >= MinFull then
⟨dbg⟩        texio.write_nl('BOLnewfirst=' .. newfirst)
      if utf8_len(newfirst) > MinPart or not match then
         oldsub = oldfirst
         newsub = newfirst
      end
      match = true
   end
   if match then
```

Minimal full or partial match `newsub` of length `k`; any more glyphs matching?

```
      local k = utf8_len(newsub)
      local osub = oldsub
      local nsub = newsub
      while osub == nsub and k < maxlen do
        k = k + 1
        osub = utf8_sub(old,1,k)
        nsub = utf8_sub(new,1,k)
        if osub == nsub then
           newsub = nsub
        end
      end
```

```
        newsub = utf8_gsub(newsub, "_+$", "")   --$
⟨dbg⟩        texio.write_nl('BOLfullmatch=' .. newsub)
        local msg = "B.O.L. MATCH=" .. newsub
        log_flaw(msg, line, colno, footnote)
```

Lest's colour the matching string.

```
        local ns = utf8_gsub(newsub, "_", "")
        k = utf8_len(ns)
        oldsub = newsub
        local newsub = ""
        local n = start
        local l = 0
        local lo = 0
        local li = 0
        while n and newsub ~= oldsub and l < k do
          if n and n.id == HLIST then
             local nn = n.head
             for nnn in traverse(nn) do
               color_node(nnn, COLOR)
               local c = nn.char
               if not char_to_discard[c] then l = l + 1 end
             end
          elseif n then
             color_node(n, COLOR)
             li, newsub = signature(n, newsub, swap)
             l = l + li - lo
             lo = li
          end
          n = n.next
        end
     end
  end
  return new, match
end
```

The next function is meant to be called on the first line of a new page. It checks the first word: if it ends a sentence and is short (up to `\luatypoMinLen` characters), the function returns `true` and colours the offending word. Otherwise it just returns `false`. The function requires two arguments: the line's first node and a column number (possibly `nil`).

```
local check_page_first_word = function (node, colno, footnote)
  local COLOR = luatypo.colortbl[15]
  local match = false
  local swap = false
  local new = ""
  local minlen = luatypo.MinLen
  local len = 0
  local n = node
  local pn
  while n and n.id ~= GLYPH and n.id ~= DISC and
        (n.id ~= HLIST or n.subtype == INDENT) do
     n = n.next
  end
  local start = n
  if n and n.id == HLIST then
     start = n.head
     n = n.head
  end
  repeat
    len, new = signature (n, new, swap)
    n = n.next
  until len > minlen or (n and n.id == GLYPH and eow_char[n.char]) or
        (n and n.id == GLUE) or
        (n and n.id == KERN and n.subtype == 1)
```

In French '?' and '!' are preceded by a glue (babel) or a kern (polyglossia).

```
  if n and (n.id == GLUE or n.id == KERN) then
     pn = n
     n = n.next
  end
  if len <= minlen and n and n.id == GLYPH and eow_char[n.char] then
```

If the line does not ends here, set `match` to `true` (otherwise this line is just a short line):

```
     repeat
       n = n.next
     until not n or n.id == GLYPH or
           (n.id == GLUE and n.subtype == PARFILL)
     if n and n.id == GLYPH then
        match = true
     end
  end
⟨dbg⟩  texio.write_nl('FinalWord=' .. new)
  if match then
     local msg = "ShortFinalWord=" .. new
     log_flaw(msg, 1, colno, footnote)
```

Lest's colour the final word and punctuation sign.

```
     local n = start
     repeat
       color_node(n, COLOR)
       n = n.next
     until eow_char[n.char]
     color_node(n, COLOR)
  end
  return match
end
```

The next 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 next two are the line and column number.

```
local check_regexpr = function (glyph, line, colno, footnote)
  local COLOR = luatypo.colortbl[4]
  local lang = glyph.lang
  local match = false
  local retflag = false
  local lchar, id = is_glyph(glyph)
local previous = glyph.prev
```

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

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

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

```
   if lchar and previous and previous.id == GLUE then
      match = utf8_find(luatypo.single[lang], utf8.char(lchar))
      if match then
         retflag = true
         local msg = "RGX MATCH=" .. utf8.char(lchar)
         log_flaw(msg, line, colno, footnote)
         color_node(glyph,COLOR)
      end
   end
end
```

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

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

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

```
      if pchar and pprev and pprev.id == GLUE then
         local pattern = utf8.char(pchar) .. utf8.char(lchar)
         match = utf8_find(luatypo.double[lang], pattern)
         if match then
            retflag = true
            local msg = "RGX MATCH=" .. pattern
            log_flaw(msg, line, colno, footnote)
            color_node(previous,COLOR)
            color_node(glyph,COLOR)
         end
      end
```

…unless a kern is found between the two chars.

```
   elseif lchar and previous and previous.id == KERN then
      local pprev = previous.prev
      if pprev and pprev.id == GLYPH then
         local pchar, id = is_glyph(pprev)
         local ppprev = pprev.prev
         if pchar and ppprev and ppprev.id == GLUE then
            local pattern = utf8.char(pchar) .. utf8.char(lchar)
            match = utf8_find(luatypo.double[lang], pattern)
            if match then
               retflag = true
               local msg = "REGEXP MATCH=" .. pattern
               log_flaw(msg, line, colno, footnote)
               color_node(pprev,COLOR)
               color_node(glyph,COLOR)
            end
         end
```

```
        end
     end
  end
return retflag
end
```

The next 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.

```
local show_pre_disc = function (disc, color)
  local n = disc
  while n and n.id ~= GLUE do
    color_node(n, color)
    n = n.prev
  end
  return n
  end
```

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

```
local footnoterule_ahead = function (head)
  local n = head
  local flag = false
  local totalht, ruleht, ht1, ht2, ht3
  if n and n.id == KERN and n.subtype == 1 then
     totalht = n.kern
     n = n.next
⟨dbg⟩     ht1 = string.format("%.2fpt", totalht/65536)

     while n and n.id == GLUE do n = n.next end
     if n and n.id == RULE and n.subtype == 0 then
        ruleht = n.height
⟨dbg⟩   ht2 = string.format("%.2fpt", ruleht/65536)
        totalht = totalht + ruleht
        n = n.next
        if n and n.id == KERN and n.subtype == 1 then
⟨dbg⟩     ht3 = string.format("%.2fpt", n.kern/65536)
           totalht = totalht + n.kern
           if totalht == 0 or totalht == ruleht then
              flag = true
           else
⟨dbg⟩             texio.write_nl(' ')
⟨dbg⟩             texio.write_nl('Not a footnoterule:')
⟨dbg⟩             texio.write('  KERN height=' .. ht1)
⟨dbg⟩             texio.write('  RULE height=' .. ht2)
⟨dbg⟩             texio.write('  KERN height=' .. ht3)
           end
         end
     end
  end
  return flag
end
```

check-EOP This function looks ahead of node in search of a page end or a footnote rule and returns the flags page_bottom and body_bottom [used in text and display math lines].

```
local check_EOP = function (node)
  local n = node
  local page_bot = false
  local body_bot = false
  while n and (n.id == GLUE    or n.id == PENALTY or
               n.id == WHATSIT )    do
    n = n.next
  end
  if not n then
     page_bot = true
     body_bot = true
  elseif footnoterule_ahead(n) then
     body_bot = true
⟨dbg⟩     texio.write_nl('=> FOOTNOTE RULE ahead')
⟨dbg⟩     texio.write_nl('check_vtop: last line before footnotes')
⟨dbg⟩     texio.write_nl(' ')
  end
  return page_bot, body_bot
end
```

check-marginnote This function checks margin notes for overfull/underfull lines; It also warns if a margin note ends too low under the last line of text.

```
local check_marginnote = function (head, line, colno, vpos, bpmn)
  local OverfullLines   = luatypo.OverfullLines
  local UnderfullLines  = luatypo.UnderfullLines
  local MarginparPos    = luatypo.MarginparPos
  local margintol       = luatypo.MParTol
  local marginpp        = tex.getdimen("marginparpush")
  local pflag  = false
  local ofirst = true
  local ufirst = true
  local n = head.head
  local bottom = vpos
  if vpos <= bpmn then
     bottom = bpmn + marginpp
  end
⟨dbg⟩  texio.write_nl('*** Margin note? ***')
  repeat
    if n and (n.id == GLUE or n.id == PENALTY) then
⟨dbg⟩     texio.write_nl('    Found GLUE or PENALTY')
       n = n.next
    elseif n and n.id == VLIST then
⟨dbg⟩     texio.write_nl('    Found VLIST')
       n = n.head
    end
  until not n or (n.id == HLIST and n.subtype == LINE)
  local head = n
  if head then
⟨dbg⟩     texio.write_nl('    Found HLIST')
  else
```

```
⟨dbg⟩    texio.write_nl('    No text line found.')
end
⟨dbg⟩ local l = 0
local last = head
while head do
  local next = head.next
  if head.id == HLIST and head.subtype == LINE then
⟨dbg⟩      l = l + 1
⟨dbg⟩      texio.write_nl('    Checking line ' .. l)
     bottom = bottom + head.height + head.depth
     local first = head.head
     local linewd = head.width
     local hmax = linewd + tex.hfuzz
     local w,h,d = dimensions(1,2,0, first)
     local Stretch = math.max(luatypo.Stretch/100,1)
     if w > hmax and OverfullLines then
⟨dbg⟩         texio.write(': Overfull!')
        pflag = true
        local COLOR = luatypo.colortbl[8]
        color_line (head, COLOR)
        if ofirst then
           local msg = "OVERFULL line(s) in margin note"
           log_flaw(msg, line, colno, false)
           ofirst = false
        end
     elseif head.glue_set > Stretch and head.glue_sign == 1 and
            head.glue_order == 0 and UnderfullLines then
⟨dbg⟩         texio.write(': Underfull!')
        pflag = true
        local COLOR = luatypo.colortbl[9]
        color_line (head, COLOR)
        if ufirst then
           local msg = "UNDERFULL line(s) in margin note"
           log_flaw(msg, line, colno, false)
           ufirst = false
        end
     end
  end
  last = head
  head = next
end
local textht = tex.getdimen("textheight")
⟨dbg⟩ local tht  = string.format("%.1fpt", textht/65536)
⟨dbg⟩ local bott = string.format("%.1fpt", bottom/65536)
⟨dbg⟩ texio.write_nl('    Bottom=' .. bott)
⟨dbg⟩ texio.write('  TextBottom=' ..tht)
if bottom > textht + marginto1 and MarginparPos then
   pflag = true
   local COLOR = luatypo.colortbl[17]
   color_line (last, COLOR)
   local msg = "Margin note too low"
   log_flaw(msg, line, colno, false)
end
return bottom, pflag
```

```
end
```

get-pagebody The next 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.

```
local get_pagebody = function (head)
  local textht = tex.getdimen("textheight")
  local fn = head.list
  local body
  repeat
    fn = fn.next
  until fn.id == VLIST and fn.height > 0
⟨dbg⟩  texio.write_nl(' ')
⟨dbg⟩  local ht = string.format("%.1fpt", fn.height/65536)
⟨dbg⟩  local dp = string.format("%.1fpt", fn.depth/65536)
⟨dbg⟩  texio.write_nl('get_pagebody: TOP VLIST')
⟨dbg⟩  texio.write(' ht=' .. ht .. '  dp=' .. dp)
  first = fn.list
  for n in traverse_id(VLIST,first) do
```

Package ‘stfloats’ seems to add 1sp to the external `\vbox` for each float found on the page. Add ±8sp tolerance when comparing `n.height` to `\textheight`.

```
      if n.subtype == 0 and n.height >= textht-1 and
         n.height <= textht+8                        then
⟨dbg⟩         local ht = string.format("%.1fpt",  n.height/65536)
⟨dbg⟩         texio.write_nl('BODY found: ht=' .. ht)
⟨dbg⟩         texio.write_nl(', ' .. n.height .. 'sp')
⟨dbg⟩         texio.write_nl(' ')
         body = n
         break
      else
⟨dbg⟩         texio.write_nl('Skip short VLIST:')
⟨dbg⟩         local ht = string.format("%.1fpt",  n.height/65536)
⟨dbg⟩         local dp = string.format("%.1fpt",  n.depth/65536)
⟨dbg⟩         texio.write('ht=' .. ht .. ', ' .. n.height .. 'sp')
⟨dbg⟩         texio.write(';  dp=' .. dp)
         local ff = n.list
         for nn in traverse_id(VLIST,ff) do
⟨dbg⟩           ht = string.format("%.1fpt",  nn.height/65536)
⟨dbg⟩           texio.write_nl('  VLIST: ht=' .. ht)
⟨dbg⟩           texio.write(', ' .. nn.height .. 'sp')
             if nn.subtype == 0 and nn.height >= textht-1 and
                nn.height <= textht+8                          then
⟨dbg⟩          texio.write('  got BODY!')
                body = nn
                break
             end
         end
      end
  end
  if not body then
     texio.write_nl('***lua-typo ERROR: PAGE BODY *NOT* FOUND!***')
  end
  return body
```

```
end
```

check-vtop The next 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.

```
check_vtop = function (top, colno, vpos)
  local head = top.list
  local PAGEmin   = luatypo.PAGEmin
  local HYPHmax   = luatypo.HYPHmax
  local LLminWD   = luatypo.LLminWD
  local BackPI    = luatypo.BackPI
  local BackFuzz  = luatypo.BackFuzz
  local BackParindent   = luatypo.BackParindent
  local ShortLines      = luatypo.ShortLines
  local ShortPages      = luatypo.ShortPages
  local OverfullLines   = luatypo.OverfullLines
  local UnderfullLines  = luatypo.UnderfullLines
  local Widows          = luatypo.Widows
  local Orphans         = luatypo.Orphans
  local EOPHyphens      = luatypo.EOPHyphens
  local RepeatedHyphens = luatypo.RepeatedHyphens
  local FirstWordMatch  = luatypo.FirstWordMatch
  local ParLastHyphen   = luatypo.ParLastHyphen
  local EOLShortWords   = luatypo.EOLShortWords
  local LastWordMatch   = luatypo.LastWordMatch
  local FootnoteSplit   = luatypo.FootnoteSplit
  local ShortFinalWord  = luatypo.ShortFinalWord
  local Stretch  = math.max(luatypo.Stretch/100,1)
  local blskip   = tex.getglue("baselineskip")
  local vpos_min = PAGEmin * blskip
  vpos_min = vpos_min * 1.5
  local linewd = tex.getdimen("textwidth")
  local first_bot  = true
  local done       = false
  local footnote   = false
  local ftnsplit   = false
  local orphanflag = false
  local widowflag  = false
  local pageshort  = false
  local overfull   = false
  local underfull  = false
  local shortline  = false
  local backpar    = false
  local firstwd = ""
  local lastwd  = ""
  local hyphcount = 0
  local pageline = 0
  local ftnline = 0
  local line = 0
  local bpmn = 0
  local body_bottom = false
  local page_bottom = false
  local pageflag = false
  local pageno = tex.getcount("c@page")
```

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

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

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

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

This is a text line, store its width, increment counters `pageline` or `ftnline` and `line` (for `log_flaw`). Let's update `vpos` (vertical position in 'sp' units) and set flag `done` to `true`.

```
     vpos = vpos + head.height + head.depth
     done = true
     local linewd = head.width
     local first = head.head
     local ListItem = false
     if footnote then
        ftnline = ftnline + 1
        line = ftnline
     else
        pageline = pageline + 1
        line = pageline
     end
```

Is this line the last one on the page or before footnotes? This has to be known early in order to set the flags `orphanflag` and `ftnsplit`.

```
     page_bottom, body_bottom = check_EOP(nextnode)
```

Is the current line overfull or underfull?

```
     local hmax = linewd + tex.hfuzz
     local w,h,d = dimensions(1,2,0, first)
     if w > hmax and OverfullLines then
        pageflag = true
        overfull = true
        local wpt = string.format("%.2fpt", (w-head.width)/65536)
        local msg = "OVERFULL line " .. wpt
        log_flaw(msg, line, colno, footnote)
     elseif head.glue_set > Stretch and head.glue_sign == 1 and
            head.glue_order == 0 and UnderfullLines then
        pageflag = true
        underfull = true
        local s = string.format("%.0f%s", 100*head.glue_set, "%")
        local msg = "UNDERFULL line stretch=" .. s
        log_flaw(msg, line, colno, footnote)
     end
```

In footnotes, set flag `ftnsplit` to `true` on page's last line. This flag will be reset to false if the current line ends a paragraph.

```
     if footnote and page_bottom then
        ftnsplit = true
```

```
      end
```

The current node being a line, `first` is its first node. Skip margin kern and/or leftskip if any.

```
      while first.id == MKERN or
            (first.id == GLUE and first.subtype == LFTSKIP) do
         first = first.next
      end
```

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

```
      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).

```
         hyphcount = 0
         firstwd = ""
         lastwd = ""
         if not footnote then
            parline = 1
            if body_bottom then
```

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

```
               orphanflag = true
            end
         end
```

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

```
         local nn = first.next
         if nn and nn.id == HLIST and nn.subtype == BOX then
            ListItem = true
         end
      elseif not footnote then
         parline = parline + 1
      end
```

Does the first word and the one on the previous line match (except lists)?

```
      if FirstWordMatch then
         local flag = not ListItem and (line > 1)
         firstwd, flag =
            check_line_first_word(firstwd, first, line, colno,
                                  flag, footnote)
         if flag then
            pageflag = true
         end
      end
```

Check the page's first word (end of sentence?).

```
      if ShortFinalWord and pageline == 1 and parline > 1 and
         check_page_first_word(first, colno, footnote) then
         pageflag = true
      end
```

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

```
    local ln = slide(first)
```

Skip a possible RULE pointing an overfull line.

```
    if ln.id == RULE and ln.subtype == 0 then
       ln = ln.prev
    end
    local pn = ln.prev
    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…

```
⟨dbg⟩  texio.write_nl('EOL CASE 1: end of paragraph')
       hyphcount = 0
       ftnsplit = false
       orphanflag = false
```

it is a widow if it is the page’s first line and it does’nt start a new paragraph. If so, we flag this line as ‘widow’; colouring full lines will take place later.

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

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

```
       local PFskip = effective_glue(pn,head)
       if ShortLines then
          local llwd = linewd - PFskip
⟨dbg⟩         local PFskip_pt = string.format("%.1fpt", PFskip/65536)
⟨dbg⟩         local llwd_pt = string.format("%.1fpt", llwd/65536)
⟨dbg⟩         texio.write_nl('PFskip= ' .. PFskip_pt)
⟨dbg⟩         texio.write('  llwd= ' .. llwd_pt)
```

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

```
          if llwd < LLminWD then
             pageflag = true
             shortline = true
             local msg = "SHORT LINE: length=" ..
                         string.format("%.0fpt", llwd/65536)
             log_flaw(msg, line, colno, footnote)
          end
       end
```

Does this (end of paragraph) line ends too close to the right margin?

```
       if BackParindent and PFskip < BackPI and
          PFskip >= BackFuzz and parline > 1 then
          pageflag = true
          backpar = true
          local msg = "NEARLY FULL line: backskip=" ..
                      string.format("%.1fpt", PFskip/65536)
          log_flaw(msg, line, colno, footnote)
       end
```

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

```
         if LastWordMatch then
            local flag = true
            if PFskip > BackPI or line == 1 then
               flag = false
            end
            local pnp = pn.prev
            lastwd, flag =
               check_line_last_word(lastwd, pnp, line, colno,
                                    flag, footnote)
            if flag then
               pageflag = true
            end
         end
      elseif pn and pn.id == DISC then
```

CASE 2: the current line ends with an hyphen.

```
⟨dbg⟩    texio.write_nl('EOL CASE 2: hyphen')
         hyphcount = hyphcount + 1
         if hyphcount > HYPHmax and RepeatedHyphens then
            local COLOR = luatypo.colortbl[3]
            local pg = show_pre_disc (pn,COLOR)
            pageflag = true
            local msg = "REPEATED HYPHENS: more than " .. HYPHmax
            log_flaw(msg, line, colno, footnote)
         end
         if (page_bottom or body_bottom) and EOPHyphens then
```

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

```
            pageflag = true
            local msg = "LAST WORD SPLIT"
            log_flaw(msg, line, colno, footnote)
            local COLOR = luatypo.colortbl[2]
            local pg = show_pre_disc (pn,COLOR)
         end
```

Track matching words at end of line.

```
         if LastWordMatch then
            local flag = true
            lastwd, flag =
               check_line_last_word(lastwd, pn, line, colno,
                                    flag, footnote)
            if flag then
               pageflag = true
            end
         end
         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 is a HLIST-1 with `glue_order=2`.

```
            local nn = nextnode.next
            local nnn = nil
            if nn and nn.next then
```

```
               nnn = nn.next
               if nnn.id == HLIST and nnn.subtype == LINE and
                  nnn.glue_order == 2 then
                  pageflag = true
                  local msg = "HYPHEN on next to last line"
                  log_flaw(msg, line, colno, footnote)
                  local COLOR = luatypo.colortbl[1]
                  local pg = show_pre_disc (pn,COLOR)
               end
            end
         end
```

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

```
      else
⟨dbg⟩   texio.write_nl('EOL CASE 3')
         hyphcount = 0
```

Track matching words at end of line and short words.

```
         if LastWordMatch and pn then
            local flag = true
            lastwd, flag =
               check_line_last_word(lastwd, pn, line, colno,
                                    flag, footnote)
            if flag then
               pageflag = true
            end
         end
         if EOLShortWords then
            while pn and pn.id ~= GLYPH and pn.id ~= HLIST do
              pn = pn.prev
            end
            if pn and pn.id == GLYPH then
               if check_regexpr(pn, line, colno, footnote) then
                  pageflag = true
               end
            end
         end
      end
```

End of scanning for the main type of node (text lines). Let's colour the whole line if necessary. If more than one kind of flaw *affecting the whole line* has been detected, a special colour is used [homearchy, repeated hyphens, etc. will still be coloured properly: `color_line` doesn't override previously set colours].

```
      if widowflag and Widows then
         pageflag = true
         local msg = "WIDOW"
         log_flaw(msg, line, colno, footnote)
         local COLOR = luatypo.colortbl[5]
         if backpar or shortline or overfull or underfull then
            COLOR = luatypo.colortbl[16]
            if backpar then backpar = false end
            if shortline then shortline = false end
```

```
             if overfull then overfull = false end
             if underfull then underfull = false end
          end
          color_line (head, COLOR)
          widowflag = false
       elseif orphanflag and Orphans then
          pageflag = true
          local msg = "ORPHAN"
          log_flaw(msg, line, colno, footnote)
          local COLOR = luatypo.colortbl[6]
          if overfull or underfull then
             COLOR = luatypo.colortbl[16]
          end
          color_line (head, COLOR)
       elseif ftnsplit and FootnoteSplit then
          pageflag = true
          local msg = "FOOTNOTE SPLIT"
          log_flaw(msg, line, colno, footnote)
          local COLOR = luatypo.colortbl[14]
          if overfull or underfull then
             COLOR = luatypo.colortbl[16]
          end
          color_line (head, COLOR)
       elseif shortline then
          local COLOR = luatypo.colortbl[7]
          color_line (head, COLOR)
          shortline = false
       elseif overfull then
          local COLOR = luatypo.colortbl[8]
          color_line (head, COLOR)
          overfull = false
       elseif underfull then
          local COLOR = luatypo.colortbl[9]
          color_line (head, COLOR)
          underfull = false
       elseif backpar then
          local COLOR = luatypo.colortbl[13]
          color_line (head, COLOR)
          backpar = false
       end
    elseif head and head.id == HLIST and head.subtype == BOX and
           head.width > 0                                    then
      if head.height == 0 then
```

This is a possible margin note.

```
         bpmn, pflag = check_marginnote(head, line, colno, vpos, bpmn)
         if pflag then pageflag = true end
      else
```

Leave `check_vtop` if a two columns box starts.

```
         local hf = head.list
         if hf and hf.id == VLIST and hf.subtype == 0 then
⟨dbg⟩          texio.write_nl('check_vtop: BREAK => multicol')
⟨dbg⟩          texio.write_nl(' ')
```

```
        break
     else
```

This is an `\hbox` (f.i. centred), let's update `vpos`, line and check for page bottom

```
        vpos = vpos + head.height + head.depth
        pageline = pageline + 1
        line = pageline
        page_bottom, body_bottom = check_EOP (nextnode)
     end
  end
elseif head.id == HLIST and
      (head.subtype == EQN or head.subtype == ALIGN) and
      (head.height > 0 or head.depth > 0) then
```

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

```
   vpos = vpos + head.height + head.depth
   if footnote then
      ftnline = ftnline + 1
      line = ftnline
   else
      pageline = pageline + 1
      line = pageline
   end
```

Is this line the last one on the page or before footnotes? (information needed to set the `pageshort` flag).

```
   page_bottom, body_bottom = check_EOP (nextnode)
```

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.

```
   local fl = true
   local wd = 0
   local hmax = 0
   if head.subtype == EQN then
      local f = head.list
      wd = rangedimensions(head,f)
      hmax = head.width + tex.hfuzz
   else
      wd = head.width
      hmax = tex.getdimen("linewidth") + tex.hfuzz
   end
   if wd > hmax and OverfullLines then
      if head.subtype == ALIGN then
         local first = head.list
         for n in traverse_id(HLIST, first) do
             local last = slide(n.list)
             if last.id == GLUE and last.subtype == USER then
                wd = wd - effective_glue(last,n)
                if wd <= hmax then fl = false end
             end
         end
      end
```

```
         if fl then
            pageflag = true
            local w = wd - hmax + tex.hfuzz
            local wpt = string.format("%.2fpt", w/65536)
            local msg = "OVERFULL equation " .. wpt
            log_flaw(msg, line, colno, footnote)
            local COLOR = luatypo.colortbl[8]
            color_line (head, COLOR)
         end
      end
   elseif head and head.id == RULE and head.subtype == 0  then
      vpos = vpos + head.height + head.depth
```

This is a RULE, possibly a footnote rule. It has most likely been detected on the previous line (then `body_bottom=true`) but might have no text before (footnote-only page!).

```
      local prev = head.prev
      if body_bottom or footnoterule_ahead (prev) then
```

If it is, set the `footnote` flag and reset some counters and flags for the coming footnote lines.

```
⟨dbg⟩       texio.write_nl('check_vtop: footnotes start')
⟨dbg⟩       texio.write_nl(' ')
         footnote = true
         ftnline = 0
         body_bottom = false
         orphanflag = false
         hyphcount = 0
         firstwd = ""
         lastwd = ""
      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.

```
   elseif body_bottom and head.id == GLUE and head.subtype == 0 then
      if first_bot then
⟨dbg⟩          local vpos_pt = string.format("%.1fpt", vpos/65536)
⟨dbg⟩          local vmin_pt = string.format("%.1fpt", vpos_min/65536)
⟨dbg⟩          texio.write_nl('pageline=' .. pageline)
⟨dbg⟩          texio.write_nl('vpos=' .. vpos_pt)
⟨dbg⟩          texio.write('   vpos_min=' .. vmin_pt)
⟨dbg⟩          if page_bottom then
⟨dbg⟩             local tht    = tex.getdimen("textheight")
⟨dbg⟩             local tht_pt = string.format("%.1fpt", tht/65536)
⟨dbg⟩             texio.write('   textheight=' .. tht_pt)
⟨dbg⟩          end
⟨dbg⟩          texio.write_nl(' ')
         if pageline > 1 and pageline < PAGEmin
            and vpos < vpos_min  and ShortPages  then
            pageshort = true
            pageflag = true
            local msg = "SHORT PAGE: only " .. pageline .. " lines"
            log_flaw(msg, line, colno, footnote)
            local COLOR = luatypo.colortbl[10]
```

```
             local n = head
             repeat
               n = n.prev
             until n.id == HLIST
             color_line (n, COLOR)
          end
          first_bot = false
       end
    elseif head.id == GLUE then
```

Increment vpos on other vertical glues.

```
       vpos = vpos + effective_glue(head,top)
    elseif head.id == KERN and head.subtype == 1 then
```

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

```
       vpos = vpos + head.kern
    elseif head.id == VLIST then
```

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

```
       vpos = vpos + head.height + head.depth
⟨dbg⟩    local tht = head.height + head.depth
⟨dbg⟩    local tht_pt = string.format("%.1fpt", tht/65536)
⟨dbg⟩    texio.write(' vbox: height=' .. tht_pt)
    end
  head = nextnode
  end
⟨dbg⟩  if nextnode then
⟨dbg⟩     texio.write('Exit check_vtop,  next=')
⟨dbg⟩     texio.write(tostring(node.type(nextnode.id)))
⟨dbg⟩     texio.write('-'.. nextnode.subtype)
⟨dbg⟩  else
⟨dbg⟩     texio.write_nl('Exit check_vtop,  next=nil')
⟨dbg⟩  end
⟨dbg⟩  texio.write_nl('')
```

Update the list of flagged pages avoiding duplicates:

```
  if pageflag then
     local plist = luatypo.pagelist
     local lastp = tonumber(string.match(plist, "%s(%d+),%s$"))
     if not lastp or pageno > lastp then
        luatypo.pagelist = luatypo.pagelist .. tostring(pageno) .. ", "
     end
  end
  return head, done
```

head is nil unless check_vtop exited on a two column start. done is true unless check_vtop found no text line.

```
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.

```
luatypo.check_page = function (head)
  local pageno = tex.getcount("c@page")
  local body = get_pagebody(head)
  local textwd, textht, checked, boxed
  local top, first, next
  local n2, n3, col, colno
  local vpos = 0
  local footnote = false
  local count = 0
  if body then
     top = body
     first = body.list
     textwd = tex.getdimen("textwidth")
     textht = tex.getdimen("textheight")
⟨dbg⟩     texio.write_nl('Body=' .. tostring(node.type(top.id)))
⟨dbg⟩     texio.write('-' .. tostring(top.subtype))
⟨dbg⟩     texio.write(';  First=' .. tostring(node.type(first.id)))
⟨dbg⟩     texio.write('-' .. tostring(first.subtype))
⟨dbg⟩     texio.write_nl(' ')
  end
  if ((first and first.id == HLIST and first.subtype == BOX) or
      (first and first.id == VLIST and first.subtype == 0))      and
     (first.width == textwd and first.height > 0 and not boxed)  then
```

Some classes (memoir, tugboat …) use one more level of bowing for two columns, let’s step down one level.

```
⟨dbg⟩     local boxwd = string.format("%.1fpt", first.width/65536)
⟨dbg⟩     texio.write_nl('One step down: boxwd=' .. boxwd)
⟨dbg⟩     texio.write_nl('Glue order=' .. tostring(first.glue_order))
⟨dbg⟩     texio.write_nl(' ')
     top = body.list
```

A float on top of a page is a VLIST-0 included in a VLIST-0 (body), it should not trigger this step down. Workaround: the body will be scanned again.

```
     if first.id == VLIST then
        boxed = body
     end
  end
```

Main loop:

```
  while top do
    first = top.list
    next = top.next
⟨dbg⟩    count = count + 1
⟨dbg⟩    texio.write_nl('Page loop' .. count)
⟨dbg⟩    texio.write(': top=' .. tostring(node.type(top.id)))
⟨dbg⟩    texio.write('-' .. tostring(top.subtype))
⟨dbg⟩    if first then
⟨dbg⟩      texio.write('  first=' .. tostring(node.type(first.id)))
⟨dbg⟩      texio.write('-' .. tostring(first.subtype))
```

```
⟨dbg⟩    end
    if top and top.id == VLIST and top.subtype == 0 and
       top.width > textwd/2                             then
```

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

```
⟨dbg⟩       local boxht = string.format("%.1fpt", top.height/65536)
⟨dbg⟩       local boxwd = string.format("%.1fpt", top.width/65536)
⟨dbg⟩       texio.write_nl('**VLIST: ')
⟨dbg⟩       texio.write(tostring(node.type(top.id)))
⟨dbg⟩       texio.write('-' .. tostring(top.subtype))
⟨dbg⟩       texio.write('  wd=' .. boxwd .. '  ht=' .. boxht)
⟨dbg⟩       texio.write_nl(' ')
       local n, ok = check_vtop(top,colno,vpos)
       if ok then checked = true end
       if n then
          next = n
       end
    elseif (top and top.id == HLIST and top.subtype == BOX) and
           (first and first.id == VLIST and first.subtype == 0) and
           (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.

```
⟨dbg⟩           texio.write_nl('**MULTICOL type1:')
⟨dbg⟩           texio.write_nl(' ')
       colno = 0
       for col in traverse_id(VLIST, first) do
           colno = colno + 1
⟨dbg⟩           texio.write_nl('Start of col.' .. colno)
⟨dbg⟩           texio.write_nl(' ')
       local n, ok = check_vtop(col,colno,vpos)
       if ok then checked = true end
⟨dbg⟩           texio.write_nl('End of col.' .. colno)
⟨dbg⟩           texio.write_nl(' ')
       end
       colno = nil
       top = top.next
⟨dbg⟩       texio.write_nl('MULTICOL type1 END: next=')
⟨dbg⟩       texio.write(tostring(node.type(top.id)))
⟨dbg⟩       texio.write('-' .. tostring(top.subtype))
⟨dbg⟩       texio.write_nl(' ')
    elseif (top and top.id == HLIST and top.subtype == BOX) and
           (first and first.id == HLIST and first.subtype == BOX) and
           (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.

```
⟨dbg⟩     texio.write_nl('**MULTICOL type2:')
⟨dbg⟩     texio.write_nl(' ')
       colno = 0
       for n in traverse_id(HLIST, first) do
           colno = colno + 1
           local col = n.list
           if col and col.list then
⟨dbg⟩              texio.write_nl('Start of col.' .. colno)
⟨dbg⟩              texio.write_nl(' ')
              local n, ok = check_vtop(col,colno,vpos)
              if ok then checked = true end
⟨dbg⟩              texio.write_nl('End of col.' .. colno)
⟨dbg⟩              texio.write_nl(' ')
           end
       end
       colno = nil
    end
```

Workaround for top floats: check the whole body again.

```
    if boxed and not next then
       next = boxed
       boxed = nil
    end
    top = next
  end
  if not checked then
     luatypo.failedlist = luatypo.failedlist .. tostring(pageno) .. ", "
⟨dbg⟩     texio.write_nl(' ')
⟨dbg⟩     texio.write_nl('WARNING: no text line found on page ')
⟨dbg⟩     texio.write(tostring(pageno))
⟨dbg⟩     texio.write_nl(' ')
  end
  return true
end
return luatypo.check_page
\end{luacode}
```

NOTE: `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`.

Add the `luatypo.check_page` function to the `pre_shipout_filter` callback (with priority 1 for colour attributes to be effective), unless option `None` is selected.

```
\AtEndOfPackage{%
  \directlua{
    if not luatypo.None then
       luatexbase.add_to_callback
           ("pre_shipout_filter",luatypo.check_page,"check_page",1)
    end
  }%
}
```

Load a config file if present in LaTeX's search path or set reasonnable defaults.

```
\InputIfFileExists{lua-typo.cfg}%
   {\PackageInfo{lua-typo.sty}{"lua-typo.cfg" file loaded}}%
   {\PackageInfo{lua-typo.sty}{"lua-typo.cfg" file not found.
                               \MessageBreak Providing default values.}%
    \definecolor{LTgrey}{gray}{0.6}%
    \definecolor{LTred}{rgb}{1,0.55,0}
    \definecolor{LTline}{rgb}{0.7,0,0.3}
```

```
  \luatypoSetColor1{red}%       Paragraph last full line hyphenated
  \luatypoSetColor2{red}%       Page last word hyphenated
  \luatypoSetColor3{red}%       Hyphens on to many consecutive lines
  \luatypoSetColor4{red}%       Short word at end of line
  \luatypoSetColor5{cyan}%      Widow
  \luatypoSetColor6{cyan}%      Orphan
  \luatypoSetColor7{cyan}%      Paragraph ending on a short line
  \luatypoSetColor8{blue}%      Overfull lines
  \luatypoSetColor9{blue}%      Underfull lines
  \luatypoSetColor{10}{red}%    Nearly empty page
  \luatypoSetColor{11}{LTred}%  First word matches
  \luatypoSetColor{12}{LTred}%  Last word matches
  \luatypoSetColor{13}{LTgrey}% Paragraph ending on a nearly full line
  \luatypoSetColor{14}{cyan}%   Footnote split
  \luatypoSetColor{15}{red}%    Too short first (final) word on the page
  \luatypoSetColor{16}{LTline}% Line color for multiple flaws
  \luatypoSetColor{17}{red}%    Margin note ending too low
  \luatypoBackPI=1em\relax
  \luatypoBackFuzz=2pt\relax
  \ifdim\parindent=0pt \luatypoLLminWD=20pt\relax
  \else\luatypoLLminWD=2\parindent\relax\fi
  \luatypoStretchMax=200\relax
  \luatypoHyphMax=2\relax
  \luatypoPageMin=5\relax
  \luatypoMinFull=3\relax
  \luatypoMinPart=4\relax
  \luatypoMinLen=4\relax
  \luatypoMarginparTol=\baselineskip
 }%
```

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

%% Minimum number of characters for the first word on a page if it ends
%% a sentence (version >= 0.65).
\ifdefined\luatypoMinLen \luatypoMinLen=4\relax\fi

%% Acceptable marginpars must end at |\luatypoMarginparTol| under
%% the page’s last line or above (version >= 0.85).
\ifdefined\luatypoMarginparTol \luatypoMarginparTol=\baselineskip \fi

%% Default colours = red, cyan, blue, LTgrey, LTred, LTline.
\definecolor{LTgrey}{gray}{0.6}
\definecolor{LTred}{rgb}{1,0.55,0}
\definecolor{LTline}{rgb}{0.7,0,0.3}
\luatypoSetColor1{red}%      Paragraph last full line hyphenated
\luatypoSetColor2{red}%      Page last word hyphenated
\luatypoSetColor3{red}%      Hyphens on to many consecutive lines
\luatypoSetColor4{red}%      Short word at end of line
\luatypoSetColor5{cyan}%     Widow
\luatypoSetColor6{cyan}%     Orphan
\luatypoSetColor7{cyan}%     Paragraph ending on a short line
\luatypoSetColor8{blue}%     Overfull lines
\luatypoSetColor9{blue}%     Underfull lines
\luatypoSetColor{10}{red}%   Nearly empty page
\luatypoSetColor{11}{LTred}% First word matches
\luatypoSetColor{12}{LTred}% Last word matches
\luatypoSetColor{13}{LTgrey}% Paragraph ending on a nearly full line
\luatypoSetColor{14}{cyan}%  Footnote split
\luatypoSetColor{15}{red}%   Too short first (final) word on the page
\luatypoSetColor{16}{LTline}% Line color for multiple flaws
\luatypoSetColor{17}{red}%   Margin note ending too low

%% Language specific settings (example for French):
%% short words (two letters max) to be avoided at end of lines.
%%\luatypoOneChar{french}{"A À Ô Y"}
%%\luatypoTwoChars{french}{"Ah Au Ça Çà Ce De Il Je La Là Le Ma Me Ne Ni
%%                          Oh On Or Ou Où Sa Se Si Ta Tu Va Vu"}
```

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.

v0.86

General: Typo corrected in the signature function. 16

get-pagebody: Package 'stfloats' adds 1sp to the external \vbox. Be less picky regarding height test. 28

v0.85

General: New function 'check_marginnote'. 26

Warn in case some pages failed to be checked properly. 10

v0.80

General: 'check_line_first_word' and 'check_line_last_word': argument footnote added. 17

'color_line' no longer overwrites colors set previously. 14

New table 'luatypo.map' for colours. 9

check-vtop: Colouring lines deferred until the full line is scanned. . . . 30

hlist-2: added detection of page bottom and increment line number and vpos. 36

v0.70

General: 'check_line_first_word' and 'check_line_last_word': length of matches corrected. 17

Package options no longer require 'kvoptions', they rely on LaTeX 'ltkeys' package. 6

v0.65

General: All ligatures are now split using the node's 'components' field rather than a table. 16

New 'check_page_first_word' function. 22

Three new functions for utf-8 strings' manipulations. 13

v0.61

General: 'check_line_first_word' returns a flag to set pageflag. . . . 20

'check_line_last_word' returns a flag to set pageflag. 17

'check_regexpr' returns a flag to set pageflag in 'check_vtop'. 23

Colours mygrey, myred renamed as LTgrey, LTred. 41

v0.60

General: Debugging stuff added. . . 43

check-page: Loop redesigned to properly handle two colums. . . . 39

check-vtop: Break 'check_vtop' loop if a two columns box starts. 29

Loop redesigned. 29

Typographical flaws are recorded here (formerly in check_page). . . 29

v0.51

footnoterule-ahead: In some cases glue nodes might preceed the footnote rule; next line added . . 25

v0.50

General: Callback 'pre_output_filter' replaced by 'pre_shipout_filter', in the former the material is not boxed yet and footnotes are not visible. 41

Go down deeper into hlists and vlists to colour nodes. 14

Homeoarchy detection added for lines starting or ending on \mbox. 17

Rollback mechanism used for recovering older versions. 5

Summary of flaws written to file '\jobname.typo'. 15

get-pagebody: New function 'get_pagebody' required for callback 'pre_shipout_filter'. . . . 28

check-vtop: Consider displayed and aligned equations too for overfull boxes. 36

Detection of overfull boxes fixed: the former code didn't work for typewriter fonts. 30

footnoterule-ahead: New function 'footnoterule_ahead'. 25

v0.40

check-vtop: All hlists of subtype LINE now count as a pageline. . . 31

Both MKERN and LFTSKIP may occur on the same line. 31

Title pages, pages with figures and/or tables may not be empty pages: check 'vpos' last line's position. 29

v0.32

General: Better protection against