Le paquet impnattypo

Raphaël Pinson
raphink@gmail.com

1.5 en date du 2019/03/04

1 Introduction

En matière de typographie française, le *Lexique des règles typographiques en usage à l'Imprimerie Nationale* est une référence incontournable.

Si la majorité des recommandations de cet ouvrage est implémentée dans le module frenchb pour babel, certaines autres recommandations méritent encore d'être automatisées pour être implémentées en LaTeX.

C'est le but original de ce paquet, initié par une question sur le site tex.stackexchange.com [1], et qui implémente plusieurs règles édictées dans ce lexique afin de les rendre plus facilement applicables aux textes édités avec LaTeX.

Au fur et à mesure que ce paquet a grandi, des fonctionnalités sont venues s'ajouter, dont certaines ne sont pas directement liées au lexique, mais améliorent la qualité typographique des documents.

2 Utilisation

Pour utiliser le paquet `impnattypo`, entrez la ligne:

```
\usepackage[<options>]{impnattypo}
```

Les options du paquet sont décrites dans les sections suivantes.

2.1 Césures

`hyphenation`

En dehors des règles générales de coupure des mots, le lexique indique qu'il faut « [éviter] les coupures de mots sur plus de trois lignes consécutives. »

Afin de simplifier le code, l'implémentation proposée décourage fortement les césures en fin de page, ainsi que les césures sur deux lignes consécutives.

Pour activer cette fonctionnalité, utilisez l'option `hyphenation`:

```
\usepackage[hyphenation]{impnattypo}
```

1. http://tex.stackexchange.com/questions/20493/french-typography-recommendations

2.2 Formatage des paragraphes

parindent

Le lexique conseille une indentation des paragraphes de 1em. Ce réglage de `\parindent` peut être obtenu par l'utilisation de l'option `parindent`:

```
\usepackage[parindent]{impnattypo}
```

lastparline

De plus, il est indiqué dans la section « Coupure des mots » que « la dernière ligne d'un alinéa doit comporter un mot ou une fin de mot de longueur au moins égale au double du renfoncement de l'alinéa suivant. » À défaut d'implémenter exactement cette solution, l'option `lastparline` s'assure que la dernière ligne d'un alinéa est au moins aussi longue que le double de la valeur de `\parindent`. [2]

Lorsque LuaTEX est utilisé, la solution de Patrick Gundlach [3] est utilisée. Avec les autres moteurs de rendu, c'est la solution native de Enrico Gregorio [4] qui fait office d'implémentation:

```
\usepackage[lastparline]{impnattypo}
```

Lorsque l'option `draft` est activée et que LuaTEX est utilisé, les espaces insécables insérés sont colorés en teal. La couleur utilisée peut être ajustée par l'option `lastparlinecolor`.

nosingleletter

Il est également recommandé d'éviter les coupures isolant une lettre. La solution proposée par Patrick Gundlach [5] permet de remédier à cela en utilisant LuaTEX. Pour activer cette fonctionalité, il faut utiliser l'option `nosingleletter`:

```
\usepackage[nosingleletter]{impnattypo}
```

Lorsque cette option est activée, seul LuaTEX (via la commande `lualatex`) pourra effectuer le rendu du document.

Lorsque l'option `draft` est activée, les espaces insécables insérés sont colorés en brown. La couleur utilisée peut être ajustée par l'option `nosingleletter color`.

homeoarchy

Lorsque deux lignes consécutives commencent ou finissent par le même mot ou la même série de lettres, cela peut induire le lecteur en erreur et cela est donc à éviter.

La correction automatique de ce phénomène est très complexe et en général non souhaitable. [6] C'est pourquoi l'option `homeoarchy` de ce paquet se contente de les détecter et de les afficher. Leur correction consistera en général en l'introduction d'une espace insécable dans le paragraphe:

2. http://tex.stackexchange.com/questions/28357/ensure-minimal-length-of-last-line
3. http://tex.stackexchange.com/questions/28357/ensure-minimal-length-of-last-line/28361#28361
4. http://tex.stackexchange.com/questions/28357/ensure-minimal-length-of-last-line/28358#28358
5. http://tex.stackexchange.com/questions/27780/one-letter-word-at-the-end-of-line
6. http://tex.stackexchange.com/questions/27588/repetition-of-a-word-on-two-lines

```
\usepackage[homeoarchy]{impnattypo}
```

Lorsque cette option est activée, seul LuaTEX (via la commande `lualatex`) pourra effectuer le rendu du document.

Cette option n'est effective que si l'option `draft` est activée.

Les espaces insécables insérées sont colorées de deux couleurs :

— Les mots entiers sont colorés en red et cette couleur peut être ajustée par l'option `homeoarchywordcolor` ;

— Les mots partiels sont colorés en orange et cette couleur peut être ajustée par l'option `homeoarchycharcolor` ;

Une séquence de glyphes est considérée comme problématique si :

— Le nombre de mots entiers trouvés dans les deux lignes consécutives est supérieur à 1. Ce paramètre peut être ajusté par l'option `homeoarchymaxwords` ;

— Le nombre de caractères trouvés dans les deux lignes consécutives est supérieur à 3. Ce paramètre peut être ajusté par l'option `homeoarchymaxchars` ;

`rivers` Une lézarde est un alignement vertical d'espaces dans un paragraphe. L'option `rivers` permet de colorer les lézardes afin de les identifier. Cette option ne corrige pas les lézardes détectées :

```
\usepackage[rivers]{impnattypo}
```

Lorsque cette option est activée, seul LuaTEX (via la commande `lualatex`) pourra effectuer le rendu du document.

Cette option n'est effective que si l'option `draft` est activée.

Les espaces insécables insérées sont colorées en lime. Cette couleur peut être ajustée par l'option `riverscolor`.

2.3 Numérotation des chapitres

`frenchchapters` Concernant la numérotation des chapitres, le lexique indique : « Dans un titre, on compose en chiffres romains grandes capitales les numéros de chapitres, à l'exception de l'ordinal « premier » en toutes lettres malgré la tendance actuelle qui tend à lui substituer la forme cardinale Chapitre I. »

L'option `frenchchapters` du paquet implémente cette recommandation :

```
\usepackage[frenchchapters]{impnattypo}
```

Si vous souhaitez bénéficier de la forme ordinale « premier » sans pour autant utiliser une numérotation des chapitres en chiffres romains, il est possible de redéfinir la macro `frenchchapter`, par exemple :

```
\let\frenchchapter\arabic % numérotation en chiffres arabes
\let\frenchchapter\babylonian % numérotation en chiffres babyloniens
```

2.4 Lignes orphelines et veuves

Il est fortement recommandé de ne pas laisser de lignes orphelines dans un document. Pour cela, nous vous conseillons d'utiliser le paquet `nowidow`:

```
\usepackage[all]{nowidow}
```

Voir la documentation de ce paquet pour plus d'options.

2.5 Mode brouillon

`draft`

Le paquet impnattypo dispose d'un mode brouillon permettant de visualiser les pénalités (espaces insécables) ajoutés par les options `nosingleletter` et `lastparline`, ainsi que les informations ajoutées par les options `homeoarchy` et `rivers`. En mode brouillon, les emplacements des espaces insécables insérés sont marqués par des rectangles de couleur.

Pour activer le mode brouillon, utilisez l'option `draft`, par exemple:

```
\usepackage[draft,lastparline]{impnattypo}
```

Cet document est générée avec l'option `draft` afin d'en montrer les effets.

3 Implementation

```
\ProvidesPackage{impnattypo}
\RequirePackage{ifluatex}
\RequirePackage{kvoptions}
\SetupKeyvalOptions{
   family=impnattypo,
   prefix=int,
}
\DeclareBoolOption{draft}
\DeclareBoolOption{frenchchapters}
\DeclareBoolOption{hyphenation}
\DeclareBoolOption{nosingleletter}
\DeclareBoolOption{parindent}
\DeclareBoolOption{lastparline}
\DeclareBoolOption{homeoarchy}
\DeclareBoolOption{rivers}
\DeclareStringOption[red]{homeoarchywordcolor}
\DeclareStringOption[orange]{homeoarchycharcolor}
\DeclareStringOption[brown]{nosingleletterscolor}
\DeclareStringOption[teal]{lastparlinecolor}
\DeclareStringOption[lime]{riverscolor}
\DeclareStringOption[1]{homeoarchymaxwords}
\DeclareStringOption[3]{homeoarchymaxchars}
\ProcessKeyvalOptions*
```

No page finishes with an hyphenated word

Discourage hyphenation on two lines in a row

Number chapters

No single letter

```
\RequirePackage{xcolor}
\def\usecolor#1{\csname\string\color@#1\endcsname\space}

\ifinthyphenation
   \brokenpenalty=10000

   \doublehyphendemerits=1000000000
\fi

\ifintfrenchchapters
   \let\frenchchapter\Roman
   \renewcommand{\thechapter}{%
     \ifnum\value{chapter}=1
       premier%
     \else
       \frenchchapter{chapter}%
     \fi
   }
\fi

\ifintnosingleletter
   \ifluatex
      \RequirePackage{luatexbase,luacode}
      \begin{luacode}
      local glyph_id = node.id "glyph"
      local glue_id  = node.id "glue"
      local hlist_id = node.id "hlist"

      local prevent_single_letter = function (head)
        while head do
          if head.id == glyph_id then                                          -- glyph
            if unicode.utf8.match(unicode.utf8.char(head.char),"%a") then   -- some kind of let
               if head.prev.id == glue_id and head.next.id == glue_id then            -- only i

                 local p = node.new("penalty")
                 p.penalty = 10000

                 \ifintdraft
                    local w = node.new("whatsit","pdf_literal")
                    w.data = "q \usecolor{\intnosinglelettercolor} 0 0 m 0 5 l 2 5 l 2 0 l b Q"

                    node.insert_after(head,head,w)
                    node.insert_after(head,w,p)
                 \else
                    node.insert_after(head,head,p)
                 \fi
               end
            end
          end
          head = head.next
        end
        return true
```

```
    end

    luatexbase.add_to_callback("pre_linebreak_filter",prevent_single_letter,"~")
    \end{luacode}
  \else
    \PackageError{The nosingleleter option only works with LuaTeX}
  \fi
\fi
```

Paragraph indentation

```
\ifintparindent
\setlength{\parindent}{1em}
\fi
```

Last line of paragraph

```
\ifintlastparline
  \ifluatex
    \RequirePackage{luatexbase,luacode}
    \begin{luacode}
    local glyph_id = node.id "glyph"
    local glue_id  = node.id "glue"
    local hlist_id = node.id "hlist"

    last_line_twice_parindent = function (head)
      while head do
        local _w,_h,_d = node.dimensions(head)
        if head.id == glue_id and head.subtype ~= 15 and (_w < 2 * tex.parindent) then

            -- we are at a glue and have less then 2*\parindent to go
            local p = node.new("penalty")
            p.penalty = 10000

            \ifintdraft
               local w = node.new("whatsit","pdf_literal")
               w.data = "q \usecolor{\intlastparlinecolor} 0 0 m 0 5 l 2 5 l 2 0 l b Q"

               node.insert_after(head,head.prev,w)
               node.insert_after(head,w,p)
            \else
               node.insert_after(head,head.prev,p)
            \fi
        end

        head = head.next
      end
      return true
    end

    luatexbase.add_to_callback("pre_linebreak_filter",last_line_twice_parindent,"lastparline"
    \end{luacode}
  \else
    \setlength{\parfillskip}{0pt plus\dimexpr\textwidth-2\parindent}
  \fi
```

Detect homeoarchies

```
\fi
\ifinthomeoarchy
 \ifintdraft
  \ifluatex
     \RequirePackage{luatexbase,luacode}
     \begin{luacode}
     local glyph_id = node.id "glyph"
     local glue_id  = node.id "glue"
     local hlist_id = node.id "hlist"

     compare_lines = function (line1,line2)
       local head1 = line1.head
       local head2 = line2.head

       local char_count = 0
       local word_count = 0

       while head1 and head2 do
          if (head1.id == glyph_id and head2.id == glyph_id
                 and head1.char == head2.char)          -- identical glyph
             or (head1.id == glue_id and head2.id == glue_id) then  -- glue

             if head1.id == glyph_id then -- glyph
                char_count = char_count + 1
             elseif char_count > 0 and head1.id == glue_id then -- glue
                word_count = word_count + 1
             end
             head1 = head1.next
             head2 = head2.next
          elseif (head1.id == 0 or head2.id == 0) then -- end of line
             break
          elseif (head1.id ~= glyph_id and head1.id ~= glue_id) then -- some other kind of nod
             head1 = head1.next
          elseif (head2.id ~= glyph_id and head2.id ~= glue_id) then -- some other kind of nod
             head2 = head2.next
          else -- no match, no special node
             break
          end
       end
       -- analyze last non-matching node, check for punctuation
       if ((head1 and head1.id == glyph_id and head1.char > 49)
            or (head2 and head2.id == glyph_id and head2.char > 49)) then
          -- not a word
       elseif char_count > 0 then
          word_count = word_count + 1
       end
       return char_count,word_count,head1,head2
     end

```

```
170     compare_lines_reverse = function (line1,line2)
171       local head1 = node.tail(line1.head)
172       local head2 = node.tail(line2.head)
173
174       local char_count = 0
175       local word_count = 0
176
177       while head1 and head2 do
178          if (head1.id == glyph_id and head2.id == glyph_id
179                and head1.char == head2.char)          -- identical glyph
180             or (head1.id == glyph_id and head2.id == glyph_id) then  -- glue
181
182             if head1.id == glyph_id then -- glyph
183                char_count = char_count + 1
184             elseif char_count > 0 and head1.id == glue_id then -- glue
185                word_count = word_count + 1
186             end
187             head1 = head1.prev
188             head2 = head2.prev
189          elseif (head1.id == 0 or head2.id == 0) then -- start of line
190             break
191          elseif (head1.id ~= glyph_id and head1.id ~= glue_id) then -- some other kind of nod
192             head1 = head1.prev
193          elseif (head2.id ~= glyph_id and head2.id ~= glue_id) then -- some other kind of nod
194             head2 = head2.prev
195          elseif (head1.id == glyph_id and head1.char < 48) then -- punctuation
196             head1 = head1.prev
197          elseif (head2.id == glyph_id and head2.char < 48) then -- punctuation
198             head2 = head2.prev
199          else -- no match, no special node
200             break
201          end
202       end
203       -- analyze last non-matching node, check for punctuation
204       if ((head1 and head1.id == glyph_id and head1.char > 49)
205            or (head2 and head2.id == glyph_id and head2.char > 49)) then
206          -- not a word
207       elseif char_count > 0 then
208          word_count = word_count + 1
209       end
210       return char_count,word_count,head1,head2
211     end
212
213     highlight = function (line,nend,color)
214        local n = node.new("whatsit","pdf_literal")
215
216        -- get dimensions
217        local w,h,d = node.dimensions(line.head,nend)
218        local w_pts = w/65536 -- scaled points to points
219
```

```
      -- set data
      n.data = "q " .. color .. " 0 0 m 0 5 l " .. w_pts .. " 5 l " .. w_pts .. " 0 l b Q"

      -- insert node
      n.next = line.head
      line.head = n
      node.slide(line.head)
   end

   highlight_reverse = function (nstart,line,color)
      local n = node.new("whatsit","pdf_literal")


      -- get dimensions
      local w,h,d = node.dimensions(nstart,node.tail(line.head))
      local w_pts = w/65536 -- scaled points to points

      -- set data
      n.data = "q " .. color .. " 0 0 m 0 5 l " .. w_pts .. " 5 l " .. w_pts .. " 0 l b Q"

      -- insert node
      node.insert_after(line.head,nstart,n)
   end

   homeoarchy = function (head)
     local cur_line = head
     local prev_line -- initiate prev_line

     local max_char = tonumber(\inthomeoarchymaxchars)
     local max_word = tonumber(\inthomeoarchymaxwords)

     while head do
       if head.id == hlist_id then -- new line
         prev_line = cur_line
         cur_line = head
         if prev_line.id == hlist_id then
            -- homeoarchy
            char_count,word_count,prev_head,cur_head = compare_lines(prev_line,cur_line)
            if char_count >= max_char or word_count >= max_word then
               local color
               if word_count >= max_word then
                  color = "q \usecolor{\inthomeoarchywordcolor}"
               else
                  color = "q \usecolor{\inthomeoarchycharcolor}"
               end

               -- highlight both lines
               highlight(prev_line,prev_head,color)
               highlight(cur_line,cur_head,color)
            end
```

```
        end
      end
      head = head.next
    end
    return true
  end

  luatexbase.add_to_callback("post_linebreak_filter",homeoarchy,"homeoarchy")

  homoioteleuton = function (head)
    local cur_line = head
    local prev_line -- initiate prev_line

    local max_char = tonumber(\inthomeoarchymaxchars)
    local max_word = tonumber(\inthomeoarchymaxwords)

    local linecounter = 0

    while head do
      if head.id == hlist_id then -- new line
        linecounter = linecounter + 1
        if linecounter > 1 then
           prev_line = cur_line
           cur_line = head
           if prev_line.id == hlist_id then
              -- homoioteleuton
              char_count,word_count,prev_head,cur_head = compare_lines_reverse(prev_line,cu
              if char_count >= max_char or word_count >= max_word then
                 local color
                 if word_count >= max_word then
                    color = "q \usecolor{\inthomeoarchywordcolor}"
                 else
                    color = "q \usecolor{\inthomeoarchycharcolor}"
                 end

                 -- highlight both lines
                 highlight_reverse(prev_head,prev_line,color)
                 highlight_reverse(cur_head,cur_line,color)
              end
           end
        end
      end
      head = head.next
    end

    return true
  end

  luatexbase.add_to_callback("post_linebreak_filter",homoioteleuton,"homoioteleuton")
  \end{luacode}
```

```
  \else
     \PackageError{The homeoarchy option only works with LuaTeX}
  \fi
 \fi
\fi
```

Detect rivers

```
\ifintrivers
 \ifintdraft
  \ifluatex
     \RequirePackage{luatexbase,luacode}
     \begin{luacode}
local glyph_id = node.id "glyph"
local glue_id  = node.id "glue"
local hlist_id = node.id "hlist"

river_analyze_line = function(line,dim1,dim2,precision)
   local head = line.head

   while head do
      if head.id == glue_id then  -- glue node
         local w1,h1,d1 = node.dimensions(line.glue_set,line.glue_sign,line.glue_order,line.hea
         local w2,h2,d2 = node.dimensions(line.glue_set,line.glue_sign,line.glue_order,line.hea
         --print("dim1:"..dim1.." ; dim2:"..dim2.." ; w1:"..w1.." ; w2:"..w2)
         if w1 > dim2 + precision then  -- out of range
            return false,head
         elseif w1 < (dim2 + precision) and w2 > (dim1 - precision) then -- found
            return true,head
         end
      end
      head = head.next
   end

   return false,head
end

rivers = function (head)
   local prev_prev_line
   local prev_line
   local cur_line = head
   local cur_node
   local char_count

   local linecounter = 0

   while head do
      if head.id == hlist_id then -- new line
         linecounter = linecounter + 1
         prev_prev_line = prev_line
         prev_line = cur_line
         cur_line = head
```

```
         if linecounter > 2 then
            cur_node = cur_line.head
            char_count = 0

            while cur_node do
               if cur_node.id == glyph_id then  -- glyph
                  char_count = char_count + 1
               elseif cur_node.id == glue_id and char_count > 0 and cur_node.next then  -- glue
                  -- prev_line
                  local w1,h1,d1 = node.dimensions(head.glue_set,head.glue_sign,head.glue_order
                  local w2,h2,d2 = node.dimensions(head.glue_set,head.glue_sign,head.glue_order
                  -- if we allow up to 45° diagonal rivers, then there can be up to + or - line
                  local w_p,h_p,d_p = node.dimensions(prev_line.head,cur_line.head) -- calculat
                  found_p,head_p = river_analyze_line(prev_line,w1,w2,h_p)

                  if found_p then
                     -- prev_prev_line
                     local w1,h1,d1 = node.dimensions(prev_line.glue_set,prev_line.glue_sign,pr
                     local w2,h2,d2 = node.dimensions(prev_line.glue_set,prev_line.glue_sign,pr
                     -- if we allow up to 45° diagonal rivers, then there can be up to + or - l
                     local w_p,h_p,d_p = node.dimensions(prev_prev_line.head,prev_line.head) --
                     found_pp,head_pp = river_analyze_line(prev_prev_line,w1,w2,h_p)

                     if found_pp then
                        local n_pp = node.new("whatsit","pdf_literal")
                        n_pp.data = "q \usecolor{\intriverscolor} 0 0 m 0 5 l 5 5 l 5 0 l b Q"
                        node.insert_after(prev_prev_line,head_pp.prev,n_pp)

                        local n_p = node.new("whatsit","pdf_literal")
                        n_p.data = "q \usecolor{\intriverscolor} 0 0 m 0 5 l 5 5 l 5 0 l b Q"
                        node.insert_after(prev_line,head_p.prev,n_p)

                        local n_c = node.new("whatsit","pdf_literal")
                        n_c.data = "q \usecolor{\intriverscolor} 0 0 m 0 5 l 5 5 l 5 0 l b Q"
                        node.insert_after(cur_line,cur_node.prev,n_c)
                     end
                  end
               end
               cur_node = cur_node.next
            end
         end
      end
      head = head.next
   end

   return true

end
```

```
419 luatexbase.add_to_callback("post_linebreak_filter",rivers,"rivers")
420       \end{luacode}
421   \else
422       \PackageError{The homeoarchy option only works with LuaTeX}
423   \fi
424 \fi
425 \fi
```

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.