chickenize

Arno Trautmann
arno.trautmann@gmx.de

July 21, 2011

Abstract

This is the package `chickenize`. It allows you to substitute or change the contents of a LuaLATEX document[1], but is actually only for fun. Please *never* use any of the functionality of this package for a production document. The following table informs you shortly about your possibilities and provides links to the Lua functions. A LATEX and plainTEX interface is also offered, see below.

function	effect
chickenize	replaces every word with "chicken"
colorstretch	shows grey boxes that depict the badness of a line
leetspeak	translates every letter into the corresponding 1337 letter
randomuclc	changes randomly between uppercase and lowercase
randomfonts	changes the font randomly between every letter
randomchars	randomizes the whole input
uppercasecolor	adds a color to every uppercase letter

If you have any suggestions or comments, just drop me a mail, I'll be happy to get any response!

Contents

[1]The code is based on pure LuaTEX features, so don't try to use it with any other TEX flavour.

1   How It Works

We make use of LuaTEXs callbacks, especially the `pre_linebreak_filter` and
the `post_linebreak_filter`. Hooking a function into these, we can nearly ar-
bitrarily change the contents of the document. If the changes should be on the
input-side (replacing with `chicken`), one can use the `pre_linebreak_filter`.
Hower, changes like inserting color are best made after the linebreak is finalized,
so `post_linebreak_filter` is used for such things.

2   How You Can Use It

There are several ways to make use of this package. As all features are imple-
mented on the Lua side, you can use these functions on their own. If you do so,
please consult the corresponding subsections in the ec:implementation] part, be-
cause there are some variables that can be adapted to your need.

    You can use the following code inside a `\directlua` statement or in a `luacode`
environment (or the corresponding thing in your format):

```
luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize the input",1)
```

Replace `"pre` by `"post` to register into the post linebreak filter. The second argu-
ment gives the function name; find a list of available functions below. You can
give a label as you like in the third argument, and the last argument gives the
order in which the functions in the callback are used. If you have no fancy stuff
going on, you can safely use 1.

    If you don't want to mess with the Lua side (but please, try it, you'll learn
much!), there is a LATEX- as well as a plainTEX interface described in the next sec-
tion. The commands may not always be on the latest code base – if anything does
not work as expected, please tell me and I'll correct it.

2.1 Commands

You have a number of commands at your hand, each of which does some manipulation of the input or output. In fact, the code is easy and straightforward, but be careful, especially when combining things. Apply features step by step so your brain won't be damaged …

Some commands have optional arguments that are *only* available for LaTeX. plainTeX users are mostly capable of finding out how to change things themselfs, but if you are willing to wrap up the code for optional argument processing, don't hesitate sharing it with me ;)

chickenize Replaces every word of the input with the word "chicken". Maybe sometime the replaced word can be changed, but up to now, it's only chicken. To be a bit less static, about every 10th chicken is uppercase. However, the beginning of a sentence is not recognized automatically.[2]

uppercasecolor Makes every uppercase character in the input colored. At the moment, the color is randomized over the full rgb scale, but that will be adjustable once options are well implemented.

randomuclc Changes every character of the input into its uppercase or lowercase variant. Well, guess what the "random" means …

randomfonts Changes the font randomly for every character. If no parameters are given, all fonts that have been loaded are used, especially including math fonts.

leetspeak Translates the input into 1337 speak. If you don't understand that, lern it, n00b.

colorstretch Inspired by Paul Isambert's code, this command prints boxes instead of lines. The greyness of the first (left-hand) box corresponds to the badness of the line, i. e. it is a measure for how much the space between words has been extended to get proper paragraph justification. The second box on the right-hand side shows the amount of stretching/shrinking when font expansion is used. Together the box greyness give you information about how well the overall greyness of the typeset page is. You may specifiy the optional arguments [(no)keeptext] to display the text or delete it, also [(no)colorexpansion] controls wether or not the font expansion should be evaluated or not.

This functionality is actually the only really usefull implementation of this package …

2.2 text-Versions

The functions of this package might be much more useful if applied only to a short sequence of words or single words instead of the whole document or paragraph. Therefore, most of the above-mentioned commands have a \text-version

---

[2]If you have a nice implementation idea, I'd love to include this!

that takes an argument. `\textrandomcolor{foo}` results in a colored foo while the rest of the document keeps its color. However, to achieve this effect, still the whole node list has to be traversed, so it may slow down your document, even if you use `\textrandomcolor` only once. Fortunately, the effect is very small and mostly negligible.[3]

Please don't fool around by mixing a `\text`-version with the non-`\text`-version. If you feel like and are not please with the result, it is up to *you* to provide a stable and working solution.

3   Implementation

```
1 \input{luatexbase.sty}
2 \directlua{dofile("chickenize.lua")}
3
4 \def\chickenize{
5   \directlua{luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize the input
6 }
7 \def\colorstretch{
8   \directlua{luatexbase.add_to_callback("post_linebreak_filter",colorstretch,"show stretch and
9 }
10 \def\leetspeak{
11   \directlua{luatexbase.add_to_callback("post_linebreak_filter",leet,"transform input to 1337",
12 }
13 \def\randomcolor{
14   \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomcolor,"random color",1)}
15 }
16 \def\randomfonts{
17   \directlua{luatexbase.add_to_callback("post_linebreak_filter",randomfonts,"random fonts",1)}
18 }
19 \def\randomuclc{
20   \directlua{luatexbase.add_to_callback("pre_linebreak_filter",randomuclc,"randomize uc/lc char
21 }
22 \def\uppercasecolor{
23   \directlua{luatexbase.add_to_callback("post_linebreak_filter",uppercasecolor,"color all uc ch
24 }
25
26 \newluatexattribute\randcolorattr
27 \def\textrandomcolor#1{%
28 \randomcolor%
29 \setluatexattribute\randcolorattr{42}#1%
30 \unsetluatexattribute\randcolorattr%
31 \directlua{randomcolor_onlytext=true}%
32 \gdef\textrandomcolor#1{%
33 \setluatexattribute\randcolorattr{42}#1%
34 \unsetluatexattribute\randcolorattr}
```

---

[3]On a 500 pages text-only LATEX document the dilation is on the order of 10% with textrandomcolor, but other manipulations can take much more time. However, you are not supposed to make such long documents with chickenize!

```
35 } %% to turn off automatic all-colorizing
36
```

4   Preparation

Loading of packages and defition of constants. Will change somewhat when migrating to expl3 (?)

```
37 \input{chickenize}
38 \RequirePackage{
39   expl3,
40   xkeyval,
41   xparse
42 }
43 %% So far, no keys are defined. This will change …
44 \ExplSyntaxOn
45 \NewDocumentCommand\chickenizesetup{m}{
46   \directlua{#1}
47 }
```

5   Definition of User-Level Macros

```
48 \DeclareDocumentCommand\chickenize{}{
49   \directlua{luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize the input
50   %% We want to "chickenize" figures, too. So …
51   \DeclareDocumentCommand\includegraphics{O{}m}{
52     \fbox{Chicken}  %% actually, I'd love to draw a mp graph showing a chicken …
53   }
54 }
55 %% specials: the balmerpeak. A tribute to http://xkcd.com/323/.
56 %%            (most probable only available for \LaTeX)
57
58 \ExplSyntaxOff  %% because of the : in the domain …
59 \NewDocumentCommand\balmerpeak{G{}O{-4cm}}{
60   \begin{tikzpicture}
61   \hspace*{#2}  %% anyhow necessary to fix centering … strange :(
62   \begin{axis}
63   [width=10cm,height=7cm,
64    xmin=-0.005,xmax=0.28,ymin=-0.05,ymax=1,
65    xtick={0,0.02,...,0.27},ytick=\empty,
66    /pgf/number format/precision=3,/pgf/number format/fixed,
67    tick label style={font=\small},
68    label style = {font=\Large},
69    xlabel = \fontspec{Punk Nova} BLOOD ALCOHOL CONCENTRATION (\%),
70    ylabel = \fontspec{Punk Nova} \rotatebox{-90}{\parbox{3cm}{\center programming\\ skills}}]
71     \addplot
72       [domain=-0.01:0.27,color=red,samples=250]
73       {0.8*exp(-0.5*((x-0.1335)^2)/.00002)+
74        0.5*exp(-0.5*((x+0.015)^2)/0.01)
```

```
75        };
76     \end{axis}
77     \end{tikzpicture}
78 }
79 \ExplSyntaxOn
```

6 Lua Module

This file contains all the necessary functions, sorted alphabetically, not by sense.

First, we set up some constants. These are made global so the code can be manipulated on document level, too.

```
80 Hhead = node.id("hhead")
81 RULE = node.id("rule")
82 GLUE = node.id("glue")
83 WHAT = node.id("whatsit")
84 COL = node.subtype("pdf_colorstack")
85 GLYPH = node.id("glyph")
```

Now we set up the nodes used for all color things. The nodes are whatsits of subtype `pdf_colorstack`.

```
86 color_push = node.new(WHAT,COL)
87 color_pop = node.new(WHAT,COL)
88 color_push.stack = 0
89 color_pop.stack = 0
90 color_push.cmd = 1
91 color_pop.cmd = 2
92
```

6.1 chickenize

The infamous `\chickenize` macro. Substitutes every word of the input with the given string. This can be elaborated arbitrarily, and whenever I feel like, I might add functionality. So far, only the string replaces the word, and even hyphenation is not possible.

```
93 chickenstring = "Chicken"
94
95 local tbl = font.getfont(font.current())
96 local space = tbl.parameters.space
97 local shrink = tbl.parameters.space_shrink
98 local stretch = tbl.parameters.space_stretch
99 local match = unicode.utf8.match
100
101 function chickenize(head)
102   for i in node.traverse_id(37,head) do  --find start of a word
103     while ((i.next.id == 37) or (i.next.id == 11) or (i.next.id == 7) or (i.next.id == 0)) do
104       i.next = i.next.next
105     end
106
107     chicken = {}
```

```
108    chicken[0] = node.new(37,1)  -- only a dummy for the loop
109    for i = 1,string.len(chickenstring) do
110      chicken[i] = node.new(37,1)
111      chicken[i].font = font.current()
112      chicken[i-1].next = chicken[i]
113    end
114
115    j = 1
116    for s in string.utfvalues(chickenstring) do
117    local char = unicode.utf8.char(s)
118      chicken[j].char = s
119      if match(char,"%s") then
120        chicken[j] = node.new(10)
121        chicken[j].spec = node.new(47)
122        chicken[j].spec.width = space
123        chicken[j].spec.shrink = shrink
124        chicken[j].spec.stretch = stretch
125      end
126      j = j+1
127    end
128
129    node.insert_before(head,i,chicken[1])
130    chicken[1].next = chicken[2] -- seems to be necessary … to be fixed
131    chicken[string.len(chickenstring)].next = i.next
132  end
133
134  return head
135 end
```

6.2 leet

The `leettable` is the substitution scheme. Just add items if you feel to. Maybe
we will differ between a light-weight version and a hardcore 1337.

```
136 leettable = {
137   [101] = 51, -- e
138   [105] = 49, -- i
139   [108] = 49, -- l
140   [111] = 48, -- o
141   [115] = 53, -- s
142   [116] = 55, -- t
143
144   [101-32] = 51, -- e
145   [105-32] = 49, -- i
146   [108-32] = 49, -- l
147   [111-32] = 48, -- o
148   [115-32] = 53, -- s
149   [116-32] = 55, -- t
150 }
```

And the function. So simple that I will not write any

```
151 function leet(head)
152   for line in node.traverse_id(Hhead,head) do
153     for i in node.traverse_id(GLYPH,line.head) do
154       if leettable[i.char] then
155         i.char = leettable[i.char]
156       end
157     end
158   end
159   return head
160 end
```

6.3   randomfonts

Traverses the output and substitutes fonts randomly. A check is done so that the font number is existing. One day, the fonts should be easily given explicitly in terms of \bf etc.

```
161 randomfontslower = 1
162 randomfontsupper = 0
163 %
164 function randomfonts(head)
165   if (randomfontsupper > 0) then rfub = randomfontsupper else rfub = font.max() end  -- either
166   for line in node.traverse_id(Hhead,head) do
167     for i in node.traverse_id(GLYPH,line.head) do
168       i.font = math.random(randomfontslower,rfub)
169     end
170   end
171   return head
172 end
```

6.4   randomuclc

Traverses the input list and changes lowercase/uppercase codes.

```
173 uclcratio = 0.5 -- so, this can even be changed!
174 randomuclc = function(head)
175   for i in node.traverse_id(37,head) do
176     if math.random() < uclcratio then
177       i.char = tex.uccode[i.char]
178     else
179       i.char = tex.lccode[i.char]
180 end
181   end
182   return head
183 end
```

6.5   randomchars

```
184 randomchars = function (head)
```

8

```
185   for line in node.traverse_id(Hhead,head) do
186     for i in node.traverse_id(GLYPH,line.head) do
187       i.char = math.floor(math.random()*512)
188     end
189   end
190   return head
191 end
```

6.6   randomcolor

Setup of the boolean for grey/color, and boundaries for the colors. rgb space is
fully used, but greyscale is only used in a visible range, i. e. to 90% instead of 100%
white.

```
192 randomcolor_grey = false
193 randomcolor_onlytext = false --switch between local and global colorization
194 -- false means "color everything"
195 Rgb_lower = 1
196 rGb_lower = 1
197 rgB_lower = 1
198 Rgb_upper = 254
199 rGb_upper = 254
200 rgB_upper = 254
201 grey_lower = 0
202 grey_upper = 900
```

This function produces the string needed for the pdf color stack. We need values
0]..[1 for the colors.

```
203 function randomcolorstring()
204   if randomcolorgrey then
205     return (0.001*math.random(grey_lower,grey_upper)).." g"
206   else
207     Rgb = math.random(Rgb_lower,Rgb_upper)/255
208     rGb = math.random(rGb_lower,rGb_upper)/255
209     rgB = math.random(rgB_lower,rgB_upper)/255
210     return Rgb..rGb..rgB.." rg"
211   end
212 end
```

The function that does all the colorizing action. It goes through the whole para-
graph and looks at every glyph. If the boolean `randomcolor_onlytext` is set, only
glyphs with the set attribute will be colored. Elsewise, all glyphs are taken.

```
213 function randomcolor(head)
214   for line in node.traverse_id(0,head) do
215     for i in node.traverse_id(37,line.head) do
216       if not(randomcolor_onlytext) or (node.has_attribute(i,luatexbase.attributes.randcolorattr
217         color_push.data = randomcolorstring()  -- color or grey string
218         line.head = node.insert_before(line.head,i,node.copy(color_push))
219         node.insert_after(line.head,i,node.copy(color_pop))
220       end
221     end
```

```
222    end
223    return head
224 end
```

6.7 uppercasecolor

Loop through all the nodes and checking whether it is uppercase. If so (and also for small caps), color it.

```
225 uppercasecolor = function (head)
226   for line in node.traverse_id(Hhead,head) do
227     for upper in node.traverse_id(GLYPH,line.head) do
228       if (((upper.char > 64) and (upper.char < 91)) or
229          ((upper.char > 57424) and (upper.char < 57451)))  then  -- for small caps! nice
230         color_push.data = randomcolorstring()  -- color or grey string
231         line.head = node.insert_before(line.head,upper,node.copy(color_push))
232         node.insert_after(line.head,upper,node.copy(color_pop))
233       end
234     end
235   end
236   return head
237 end
```

6.7.1 colorstretch

This function displays the amount of stretching that has been done for each line of an arbitrary document. A well-typeset document should be equally grey over all lines, which is not always possible.

The function shows in fact two boxes: The first (left) box shows the badness, i. e. the amount of stretching the spaces between words. Too much space results in ligth gray, whereas a too dense line is indicated by a dark grey box.

The second box is only usefull if microtypographic extensions are used, e. g. with the microtype package under LaTeX. The box color then corresponds to the amount of font expansion in the line. This can be greatly used to show the positive effect of font expansion on the badness of a line!

The base structure of the following code is written by Paul Isambert. Thanks for the code and support, Paul!

Two booleans, keeptext, and colorexpansion, are used to control the behaviour of the function.

```
238 keeptext = true
239 colorexpansion = true
```

After setting the constants, the function starts. It receives the vertical list of the typeset paragraph as head, and loops through all horizontal lists.

If font expansion should be shown (colorexpansion == true), then the first glyph node is determined and its width compared with the width of the unexpanded glyph. This gives a measure for the expansion factor and is translated into a grey scale.

```
240 colorstretch = function (head)
241
242   local f = font.getfont(font.current()).characters
243   for line in node.traverse_id(Hhead,head) do
244     local rule_bad = node.new(RULE)
245
246 if colorexpansion then  -- if also the font expansion should be shown
247       local g = line.head
248         while not(g.id == 37) do
249          g = g.next
250         end
251       exp_factor = g.width / f[g.char].width
252       exp_color = .5 + (1-exp_factor)*10 .. " g"
253       rule_bad.width = 0.5*line.width  -- we need two rules on each line!
254     else
255       rule_bad.width = line.width  -- only the space expansion should be shown, only one rule
256     end
```

Height and depth of the rules are adapted to print a closed grey pattern, so no white interspace is left.

The glue order and sign can be obtained directly and are translated into a grey scale.

```
257     rule_bad.height = tex.baselineskip.width*4/5  -- this should give a quite nice output!
258     rule_bad.depth = tex.baselineskip.width*1/5
259
260     local glue_ratio = 0
261     if line.glue_order == 0 then
262       if line.glue_sign == 1 then
263         glue_ratio = .5 * math.min(line.glue_set,1)
264       else
265         glue_ratio = -.5 * math.min(line.glue_set,1)
266       end
267     end
268     color_push.data = .5 + glue_ratio .. " g"
```

Now, we throw everything together in a way that works. Somehow …

```
269 -- set up output
270     local p = line.head
271
272   -- a rule to immitate kerning all the way back
273     local kern_back = node.new(RULE)
274     kern_back.width = -line.width
275
276   -- if the text should still be displayed, the color and box nodes are inserted additionally
277   -- and the head is set to the color node
278     if keeptext then
279       line.head = node.insert_before(line.head,line.head,node.copy(color_push)) -- make the col
280     else
281       node.flush_list(p)
282       line.head = node.copy(color_push)
```

```
283       end
284       node.insert_after(line.head,line.head,rule_bad)  -- then the rule
285       node.insert_after(line.head,line.head.next,node.copy(color_pop)) -- and then pop!
286       tmpnode =  node.insert_after(line.head,line.head.next.next,kern_back)
287
288       -- then a rule with the expansion color
289       if colorexpansion then  -- if also the stretch/shrink of letters should be shown
290         color_push.data = exp_color
291         node.insert_after(line.head,tmpnode,node.copy(color_push))
292         node.insert_after(line.head,tmpnode.next,node.copy(rule_bad))
293         node.insert_after(line.head,tmpnode.next.next,node.copy(color_pop))
294       end
295     end
296   return head
297 end
```

And that's it :)

7   Known Bugs

There are surely some bugs …

  ???

8   To Dos

Some things that should be implemented but aren't so far or are very poor at the moment:

  ?

This is the README file that should contain some important information. So far I can only tell you to run the lualatex on the file chickenize.dtx to produce the four files chickenize.pdf (documentation) chickenize.tex (low-level commands; plain-TeX) chickenize.sty (LaTeX user interface) chickenize.lua (Lua package code)

    You need an up-to-date TeX Live (2011, if possible) to use this package.

    For any comments or suggestions, contact me: arno dot trautmann at gmx dot de

    Hope you have fun with this!