»The Monthy Pythons, were they TEX users,
could have written the chickenize macro.«
Paul Isambert

chickenize

Arno Trautmann
arno.trautmann@gmx.de

July 5, 2011

Abstract

This is the documentation of the package `chickenize`. It allows you to substitute or change the contents of a LuaLATEX document.[1] You have e. g. the possibility to substitute every word of a document with the word "chicken", translate it into 1337 speak, make it totally colorfull or use upper/lowercase all randomly. Of course this package is *not* meant for any serious document, but only for fun and – because we can!

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 Usage

This package should be useable some time …

2 Working Principle

We make use of LuaTEXs callbacks, especially the `pre_linebreak_filter` and the `post_linebreak_filter`. Hooking a function into these, we can chanke the input (into "chicken") or add/transform the input (putting color in, changing lower/uppercase).

3 Package Options

There surely will be some options etc.

4 Commands

You have a number of commands at your hand, each of which does some manipulating with the input. In fact, the code is easy and straightforward, but the result may blow your mind. Be careful, especially when combining things …

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, after a full-stop

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 …

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 …

5 Implementation

```
\input{luatexbase.sty}
\directlua{dofile("chickenize.lua")}

\def\chickenize{
  \directlua{luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize the input
}
\def\uppercasecolor{
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",uppercasecolor,"color all uc ch
}
\def\randomuclc{
  \directlua{luatexbase.add_to_callback("pre_linebreak_filter",randomuclc,"randomize uc/lc char
}
\def\colorstretch{
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",colorstretch,"show stretch and
}
\def\leetspeak{
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",leet,"transform input to 1337",
}
```

6 Preparation

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

```
\input{chickenize}
\RequirePackage{
  expl3,
  xkeyval,
  xparse
}
%% So far, no keys are defined. This will change …
\ExplSyntaxOn
\keys_define:nn {chick} {
  keeptext.code:n = \directlua{keeptext = true},
  colorexpansion.code:n = \directlua{colorexpansion = true},
  nokeeptext.code:n = \directlua{keeptext = false},
  nocolorexpansion.code:n = \directlua{colorexpansion = false}
}
\NewDocumentCommand\chicksetup{m}{
  \keys_set:nn{chick}{#1}
}
```

7 Definition of User-Level Macros

```
\DeclareDocumentCommand\chickenize{}{
  \directlua{luatexbase.add_to_callback("pre_linebreak_filter",chickenize,"chickenize the input
  %% We want to "chickenize" figures, too. So …
  \DeclareDocumentCommand\includegraphics{O{}m}{
     \fbox{Chicken}  %% actually, I'd love to draw a mp graph showing a chicken …
  }
}
\DeclareDocumentCommand\uppercasecolor{}{
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",uppercasecolor,"color all uc ch
}
\DeclareDocumentCommand\randomuclc{}{
  \directlua{luatexbase.add_to_callback("pre_linebreak_filter",randomuclc,"randomize uc/lc char
}

\DeclareDocumentCommand\colorstretch{O{}}{
  \keys_set:nn { chick } {#1}
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",colorstretch,"show stretch and
}
\DeclareDocumentCommand\leetspeak{}{
  \directlua{luatexbase.add_to_callback("post_linebreak_filter",leet,"transform input to 1337",
}

%% specials: the balmerpeak. A tribute to http://xkcd.com/323/.
%%           (most probable only available for \LaTeX)

\ExplSyntaxOff  %% because of the : in the domain …
\NewDocumentCommand\balmerpeak{G{}O{-4cm}}{
  \begin{tikzpicture}
  \hspace*{#2}  %% anyhow necessary to fix centering … strange :(
  \begin{axis}
  [width=10cm,height=7cm,
   xmin=-0.005,xmax=0.28,ymin=-0.05,ymax=1,
   xtick={0,0.02,...,0.27},ytick=\empty,
   /pgf/number format/precision=3,/pgf/number format/fixed,
   tick label style={font=\small},
   label style = {font=\Large},
   xlabel = \fontspec{Punk Nova} BLOOD ALCOHOL CONCENTRATION (\%),
   ylabel = \fontspec{Punk Nova} \rotatebox{-90}{\parbox{3cm}{\center programming\\ skills}}]
    \addplot
      [domain=-0.01:0.27,color=red,samples=250]
      {0.8*exp(-0.5*((x-0.1335)^2)/.00002)+
       0.5*exp(-0.5*((x+0.015)^2)/0.01)
      };
  \end{axis}
  \end{tikzpicture}
}
\ExplSyntaxOn
```

8 Lua Module

This file contains all the necessary functions.

```
local Hhead = node.id("hhead")
local RULE = node.id("rule")
local GLUE = node.id("glue")
local WHAT = node.id("whatsit")
local COL = node.subtype("pdf_colorstack")
local GLYPH = node.id("glyph")

uppercasecolor = function (head)
  for line in node.traverse_id(Hhead,head) do
    for upper in node.traverse_id(GLYPH,line.head) do
      if (((upper.char > 64) and (upper.char < 91)) or
          ((upper.char > 57424) and (upper.char < 57451)))  then  -- for small caps! nice
        color_push.data = math.random()..math.random()..math.random().." rg"
        line.head = node.insert_before(line.head,upper,node.copy(color_push))
        node.insert_after(line.head,upper,node.copy(color_pop))
      end
    end
  end
  return head
end

randomuclc = function(head)
  for i in node.traverse_id(37,head) do
    if math.random() < 0.5 then
      i.char = tex.uccode[i.char]
    else
      i.char = tex.lccode[i.char]
      i.yoffset = "15 pt"
end
  end
  return head
end

function chickenize(head)
  for i in node.traverse_id(37,head) do  --find start of a word
    while ((i.next.id == 37) or (i.next.id == 11) or (i.next.id == 7) or (i.next.id == 0)) do
      i.next = i.next.next
    end

    chicken = {}
    chicken[0] = node.new(37,1)
    for i = 1,7 do
      chicken[i] = node.new(37,1)
      chicken[i].font = font.current()
    end
    node.insert_before(head,i,chicken[1])

  -- randomize upper/lower case to get a more natural output.
  -- however, this may make break points inconsistent!
if (math.random() > 0.8) then
    chicken[7].char = 67  else
    chicken[7].char = 99
end

    chicken[6].char = 104
    chicken[5].char = 105
    chicken[4].char = 99
    chicken[3].char = 107
    chicken[2].char = 101
    chicken[1].char = 110
lang.hyphenate(chicken[1])
    for k = 1,6 do
      node.insert_before(head,chicken[k],chicken[k+1])
    end
    chicken[1].next = i.next
  end

  return head
end

leettable = {
  [101] = 51, -- e
  [105] = 49, -- i
  [108] = 49, -- l
  [111] = 48, -- o
  [115] = 53, -- s
  [116] = 55, -- t

  [101-32] = 51, -- e
  [105-32] = 49, -- i
  [108-32] = 49, -- l
  [111-32] = 48, -- o
  [115-32] = 53, -- s
  [116-32] = 55, -- t
}

function leet(head)
  for line in node.traverse_id(Hhead,head) do
    for i in node.traverse_id(GLYPH,line.head) do
      if leettable[i.char] then
        i.char = leettable[i.char]
      end
    end
  end
  return head
end
```

8.0.1 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!

First, we set up some constants that will be used later on. Two booleans, `keeptext`, and `colorexpansion`, are used to control the behaviour of the function.

```
local color_push = node.new(WHAT,COL)
local color_pop = node.new(WHAT,COL)
color_push.stack = 0
color_pop.stack = 0
color_push.cmd = 1
color_pop.cmd = 2

local keeptext = true
local 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.

```
colorstretch = function (head)

  local f = font.getfont(font.current()).characters
  for line in node.traverse_id(Hhead,head) do
    local rule_bad = node.new(RULE)

if colorexpansion then  -- if also the stretch/shrink of letters should be shown
      local g = line.head
        while not(g.id == 37) do
         g = g.next
        end
      exp_factor = g.width / f[g.char].width
      exp_color = .5 + (1-exp_factor)*10 .. " g"
      rule_bad.width = 0.5*line.width  -- we need two rules on each line!
    else
      rule_bad.width = line.width  -- only the space expansion should be shown, only one rule
    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.

```
  rule_bad.height = tex.baselineskip.width*4/5  -- this should give a quite nice output!
  rule_bad.depth = tex.baselineskip.width*1/5

  local glue_ratio = 0
  if line.glue_order == 0 then
    if line.glue_sign == 1 then
      glue_ratio = .5 * math.min(line.glue_set,1)
    else
      glue_ratio = -.5 * math.min(line.glue_set,1)
    end
  end
  color_push.data = .5 + glue_ratio .. " g"
```

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

```
-- set up output
  local p = line.head

 -- a rule to immitate kerning all the way back
  local kern_back = node.new(RULE)
  kern_back.width = -line.width

 -- if the text should still be displayed, the color and box nodes are inserted additionally
 -- and the head is set to the color node
  if keeptext then
    line.head = node.insert_before(line.head,line.head,node.copy(color_push)) -- make the col
  else
    node.flush_list(p)
    line.head = node.copy(color_push)
  end
  node.insert_after(line.head,line.head,rule_bad)  -- then the rule
  node.insert_after(line.head,line.head.next,node.copy(color_pop)) -- and then pop!
  tmpnode =  node.insert_after(line.head,line.head.next.next,kern_back)

  -- then a rule with the expansion color
  if colorexpansion then  -- if also the stretch/shrink of letters should be shown
    color_push.data = exp_color
    node.insert_after(line.head,tmpnode,node.copy(color_push))
    node.insert_after(line.head,tmpnode.next,node.copy(rule_bad))
    node.insert_after(line.head,tmpnode.next.next,node.copy(color_pop))
    end
  end
  return head
end
```

And that's it :)

9 Known Bugs

There are surely some bugs …

???

10 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!