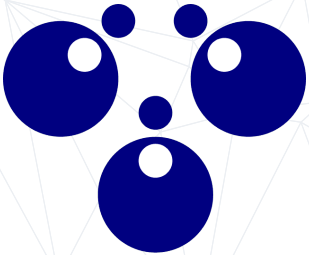


Maxime Chupin
notezik@gmail.com

Groupe des utilisateurs francophone de (LA)TEX

12 septembre 2024

LuaLATEX : petite introduction à l'utilisation de quelques *callbacks*

Exposés mensuels GUTenberg

gutenberg-asso.fr

TikZ

tikz.fr

MP

metapost.gutenberg-asso.fr

Cahiers GUTenberg

cahiers.gutenberg-asso.fr

FAQ LaTeX

faq.gutenberg-asso.fr

La Lettre GUTenberg

lettre.gutenberg-asso.fr

texnique.fr

gitlab.gutenberg-asso.fr

gut@ens.fr

Samedi 16 novembre

à l'École normale supérieure
45 rue d'Ulm - 75005 Paris
en salle Henri Cartan.

09h30-10h00 Accueil et café
10h00-11h00 Exposé sur le système KerTEX par Thierry Larronde
11h00-12h00 Exposé sur les fontes variables par Jacques André
12h00-14h00 Repas de groupe proposé au restaurant Mauzac (à régler individuellement)
14h00-15h00 Exposé sur le package `xint` par Jean-François Burnol
15h00-17h00 Assemblée générale
17h00-17h30 et plus Moment convivial

GUTenberg

Adhérez !

Plan général

LuaTEX

- moteur TEX réécrit en embarquant le langage de programmation Lua

1. ou *fonction de rappel*

LuaTEX

- moteur TEX réécrit en embarquant le langage de programmation Lua
- de nombreuses améliorations : unicode, fontes (TrueType, OpenType), bibliothèques Lua (epdf, kpse, mplib, node, tex, token), mathématiques.

1. ou *fonction de rappel*

LuaTEX

- moteur TEX réécrit en embarquant le langage de programmation Lua
- de nombreuses améliorations : unicode, fontes (TrueType, OpenType), bibliothèques Lua (epdf, kpse, mplib, node, tex, token), mathématiques.

1. ou *fonction de rappel*

LuaTEX

- moteur TEX réécrit en embarquant le langage de programmation Lua
- de nombreuses améliorations : unicode, fontes (TrueType, OpenType), bibliothèques Lua (epdf, kpse, mplib, node, tex, token), mathématiques.

callbacks[1]

- une porte donnant sur les opérations internes de TEX

1. ou *fonction de rappel*

LuaTEX

- moteur TEX réécrit en embarquant le langage de programmation Lua
- de nombreuses améliorations : unicode, fontes (TrueType, OpenType), bibliothèques Lua (epdf, kpse, mplib, node, tex, token), mathématiques.

callbacks[1]

- une porte donnant sur les opérations internes de TEX
- enrichir ou remplacer le fonctionnement par défaut de TEX

1. ou *fonction de rappel*

LuaTeX

- moteur TeX réécrit en embarquant le langage de programmation Lua
- de nombreuses améliorations : unicode, fontes (TrueType, OpenType), bibliothèques Lua (epdf, kpse, mplib, node, tex, token), mathématiques.

callbacks[1]

- une porte donnant sur les opérations internes de TeX
- enrichir ou remplacer le fonctionnement par défaut de TeX
- un grand nombre de callbacks (plus de 50) : gestion de fichier (ouverture, écriture), gestion des fontes, intercepter les lexèmes (*tokens*), créer les paragraphes, etc.

1. ou *fonction de rappel*

Exécuter du code Lua

- `\directlua` (primitive)

```
$\pi=\directlua{tex.print(math.pi)}$
```

π = 3.1415926535898

Exécuter du code Lua

- `\directlua` (primitive)

```
$\pi=\directlua{tex.print(math.pi)}$
```

$\pi = 3.1415926535898$

- Ouvre les portes à une programmation plus « classique »

```
\directlua{
function fibonacci(n)
  if (n==0) then
    return 0
  elseif(n==1) then
    return 1
  else
    return fibonacci(n-1)+fibonacci(n
          -2)
  end
end}
Le terme de rang $16$ de la suite de
      Fibonacci est
\(u_{16}=\directlua{tex.print(
      fibonacci(16))}\).
```

Le terme de rang 16 de la suite de Fibonacci est $u_{16} = 987$.

Lua, LaTeX et le package luacode (I)

- On mélange des langages dans le même source
- Package `luacode`

	`\luadirect`	`\luaexec`	`luacode`	`luacode*`
Macros	Oui	Oui	Oui	Non
Une contre-oblique	`\string\`	`\string\`	`\string\`	Simplement `\`
Deux contre-oblique	`\string\\`	`\\`	`\\`	`\\`
Tilde	`\string~`	`~`	`~`	`~`
Dièse	`\string#`	`\#`	`\#` ou `#`	`#`
Pourcent	Pas de façon simple	`\%`	`\%` ou `%`	`%`
Commentaire TeX	Oui	Oui	Non	Non
Commentaire Lua	Non	Non	Oui	Oui

- Exemple d'utilisation de \luaexec

```
\newcommand\FF{19}
\(u_{\FF}=\luaexec{tex.print(fibonacci(\FF))}\)
```

u_{19} = 4181

- Possibilité d'exécuter du code Lua depuis un fichier externe

```
1 \luadirect{require("fichier.lua")}%
```

Callbacks de fabrication d'un paragraphe et enregistrement

Principaux callbacks de la fabrication d'un paragraphe

- `process_input_buffer` détermine comment TEX lit chaque ligne en entrée (source)
- `hyphenate` détermine où et comment sont insérées les césures
- `ligaturing` détermine où les ligatures doivent être produites
- `kerning` détermine où les crénages sont insérés
- `pre_linebreak_filter` permet de faire des opérations avant que le paragraphe soit construit
- `linebreak_filter` construit le paragraphe
- `post_linebreak_filter` permet de faire des opérations après que le paragraphe est construit

Enregistrer un callback (I)

Avec LuaTEX

- enregistrer une fonction

```
1 id, error = callback.register(<string> cb_name, <function
      > func)
```

Code Lua

Enregistrer un callback (I)

Avec LuaTEX

- enregistrer une fonction

```lua
id, error = callback.register(<string> cb_name, <function
    > func)
```

Code Lua

- rétablir le comportement par défaut

```lua
id, error = callback.register(<string> cb_name, nil)
```

Code Lua

Avec LuaLaTeX et le package `ltluatex`

- package chargé automatiquement

Avec LuaLaTeX et le package `ltluatex`

- package chargé automatiquement
- enregistrer une fonction

```lua
luatexbase.add_to_callback(<string> cb_name, <function>
    func, <string> description)
```

Code Lua

Enregistrer un callback (II)

Avec LuaLaTeX et le package `ltluatex`

- package chargé automatiquement
- enregistrer une fonction

```lua
luatexbase.add_to_callback(<string> cb_name, <function>
    func, <string> description)
```

Code Lua

- rétirer l'enregistrement sous le nom `description`

```lua
luatexbase.remove_from_callback(<string> cb_name, <string
    > description)
```

Code Lua

Insérer des césures

Insérer des césures (I)

- callback `hyphenate` prend deux arguments : `head` et `tail`, respectivement le premier et le dernier nœud de la liste à traiter

Insérer des césures (I)

- callback `hyphenate` prend deux arguments : `head` et `tail`, respectivement le premier et le dernier nœud de la liste à traiter
- LuaTEX travaille, comme TEX, avec des nœuds : `node` en Lua

Insérer des césures (I)

- callback `hyphenate` prend deux arguments : `head` et `tail`, respectivement le premier et le dernier nœud de la liste à traiter
- LuaTEX travaille, comme TEX, avec des nœuds : `node` en Lua

Insérer des césures (I)

- callback `hyphenate` prend deux arguments : `head` et `tail`, respectivement le premier et le dernier nœud de la liste à traiter
- LuaTEX travaille, comme TEX, avec des nœuds : `node` en Lua

Nœuds

- Type : `node.types()` (ou champs `id`) : `hlist (0)`, `vlist (1)`, `rule (2)`, `ins (3)`, `mark (4)`, `adjust (5)`, `boundary (6)`, `disc (7)`, `whatsit (8)`, etc. (jusqu'à 49 !)

Insérer des césures (I)

- callback `hyphenate` prend deux arguments : `head` et `tail`, respectivement le premier et le dernier nœud de la liste à traiter
- LuaTEX travaille, comme TEX, avec des nœuds : `node` en Lua

Nœuds

- Type : `node.types()` (ou champs `id`) : `hlist (0)`, `vlist (1)`, `rule (2)`, `ins (3)`, `mark (4)`, `adjust (5)`, `boundary (6)`, `disc (7)`, `whatsit (8)`, etc. (jusqu'à 49 !)
- d'autres champs : `next`, `subtype`, et suivant les types `width`, `height`, `depth`, `attr`, `stretch`, `shrink`, etc.

Insérer des césures (II)

Une fonction d'affichage

```
1  local GLYF = node.id("glyph")
2  function show_nodes (head)
3    local nodes = ""
4    for item in node.traverse(head) do
5      local i = item.id
6      if i == GLYF then
7        i = unicode.utf8.char(item.char)
8      end
9      nodes = nodes .. i .. " "
10   end
11   texio.write_nl(nodes)
12 end
```

Code Lua

Regardons ce qu'hyphenate reçoit

```
\luadirect{
  luatexbase.add_to_callback("hyphenate",show_nodes,"Show
  nodes")
}
Est-ce effectif?
\luadirect{
  luatexbase.remove_from_callback("hyphenate","Show nodes")
}
```

Regardons ce qu'hyphenate reçoit

```
\luadirect{
  luatexbase.add_to_callback("hyphenate",show_nodes,"Show
  nodes")
}
Est-ce effectif?
\luadirect{
  luatexbase.remove_from_callback("hyphenate","Show nodes")
}
```

```
41 9 0 E s t - c e 12 e f f e c t i f ?
```

Insérer des césures (IV)

41 9 0 E s t - c e 12 e f f e c t i f ?

- 41 : raison technique
- 9 : *whatsit* (élément extraordinaire), de sous type 6, `local_par` indiquant en particulier la direction de composition (gauche à droit ici)
- 0 : liste horizontale, `hbox`, et sous type (3) indique une boîte d'indentation
- 12 : *glue*, ou ressort, avec comme champs `width`, `stretch`, `stretch_order`, `shrink` et `shrink_order`

Que fait hyphenate ?

```
\begin{luacode}
  function myhyph(head, tail)
    lang.hyphenate(head)
    show_nodes(head)
  end
\end{luacode}
\luadirect{
  luatexbase.add_to_callback("hyphenate",myhyph,"My Hyphenate")
}
Est-ce effectif?
\luadirect{
  luatexbase.remove_from_callback("hyphenate","My Hyphenate")
}
```

Insérer des césures (VI)

- `lang.hyphenate(head)` insère les césures

Insérer des césures (VI)

- `lang.hyphenate(head)` insère les césures
- sortie : `41` `9` `0` `E` `s` `t` `7` `c` `e` `12` `e` `f` `7` `f` `e` `c` `7` `t` `i` `f` `?`

Insérer des césures (VI)

- `lang.hyphenate(head)` insère les césures
- sortie : `41` `9` `0` `E` `s` `t` `7` `c` `e` `12` `e` `f` `7` `f` `e` `c` `7` `t` `i` `f` `?`
- ajout de nœuds `7` de césure avec trois champs : `pre`, `post` et `replace`, équivalents des arguments de `\discretionary`

Insérer des césures (VI)

- `lang.hyphenate(head)` insère les césures
- sortie : `41` `9` `0` `E` `s` `t` `7` `c` `e` `12` `e` `f` `7` `f` `e` `c` `7` `t` `i` `f` `?`
- ajout de nœuds `7` de césure avec trois champs : `pre`, `post` et `replace`, équivalents des arguments de `\discretionary`

entre `Est` et `ce` : un tiret donc point de césure potentiel. Champ `replace` contenant le caractère - lorsqu'il n'y pas de coupure, un champ `pre` contenant lui aussi - (lorsque la ligne se coupe à cet endroit) et champ `post` vide

Insérer des césures (VI)

- `lang.hyphenate(head)` insère les césures
- sortie : `41` `9` `0` `E` `s` `t` `7` `c` `e` `12` `e` `f` `7` `f` `e` `c` `7` `t` `i` `f` `?`
- ajout de nœuds `7` de césure avec trois champs : `pre`, `post` et `replace`, équivalents des arguments de `\discretionary`

entre `Est` et `ce` : un tiret donc point de césure potentiel. Champ `replace` contenant le caractère - lorsqu'il n'y pas de coupure, un champ `pre` contenant lui aussi - (lorsque la ligne se coupe à cet endroit) et champ `post` vide

entre les deux `f` et entre le `c` et le `t` : point de césure classique, sans champ `replace`, un champ `pre` contenant -, et un champ `post` vide.

Un dernier mot

- avec TEX

```
\hyphenation{man-u-script man-u-scripts ap-pen-dix}
```

- avec LuaTEX, équivalent de la structure `pre`, `post` et `replace`

```
\hyphenation{ba{k-}{}{c}ken}
```

Les ligatures (I)

- Callback : `ligaturing`

Les ligatures (I)

- Callback : `ligaturing`
- Attention : même si on dispose du callback gérant les ligatures, ce travail est laissé aux fontes OpenType ou TrueType si elles sont utilisées

Les ligatures (I)

- Callback : `ligaturing`
- Attention : même si on dispose du callback gérant les ligatures, ce travail est laissé aux fontes OpenType ou TrueType si elles sont utilisées
- En pratique, callback inutile sauf pour jouer et comprendre

Les ligatures (I)

- Callback : `ligaturing`
- Attention : même si on dispose du callback gérant les ligatures, ce travail est laissé aux fontes OpenType ou TrueType si elles sont utilisées
- En pratique, callback inutile sauf pour jouer et comprendre

- Callback : `ligaturing`
- Attention : même si on dispose du callback gérant les ligatures, ce travail est laissé aux fontes OpenType ou TrueType si elles sont utilisées
- En pratique, callback inutile sauf pour jouer et comprendre

Avec LuaLaTeX forcer l'utilisation d'une fonte T1

```
\usepackage[T1]{fontenc}
```

préambule

Les ligatures (II)

```
\begin{luacode}
function mylig (head, tail)
  node.ligaturing(head)
  show_nodes(head)
end
\end{luacode}
\luadirect{
  luatexbase.add_to_callback("ligaturing", mylig, "My
  Ligaturing")
}
Est-ce effectif?
\luadirect{
  luatexbase.remove_from_callback("ligaturing", "My Ligaturing
  ")
}
```

Les ligatures (III)

- `41` `9` `0` `E` `s` `t` `7` `c` `e` `12` `e` `7` `e` `c` `7` `t` `i` `f` `?`
- Qu'est devenu `f` `7` `f` ?
- Interaction entre les césures et les ligatures
- La présence de la ligature est conditionnée à la non césure
- Nœud de césure `7` aura `pre=f-`, `post=f` et `replace=<ff>` où `<ff>` représente la ligature
- Nœud de ligature, sous type 2, champs `components` qui pointe vers une liste de nœud de glyphe qui le compose (ici `f` et `f`)

Exercice : des guillemets français (I)

- Transformer toutes les occurrences du type `"mot"` par `«mot»`

Exercice : des guillemets français (I)

- Transformer toutes les occurrences du type `"mot"` par `«mot»`
- alert*Substitution contextuelle*

Exercice : des guillemets français (I)

- Transformer toutes les occurrences du type `"mot"` par «`mot`»
- alert*Substitution contextuelle*
- Algorithme : pour chaque glyphe "
  - si " est précédé par un glyphe autre qu'une parenthèse, on le remplace par »
  - sinon on le remplace par «

Exercice : des guillemets français (I)

- Transformer toutes les occurrences du type `"mot"` par «`mot`»
- alert*Substitution contextuelle*
- Algorithme : pour chaque glyphe "
  - si " est précédé par un glyphe autre qu'une parenthèse, on le remplace par »
  - sinon on le remplace par «

Exercice : des guillemets français (I)

- Transformer toutes les occurrences du type `"mot"` par «`mot`»
- alert*Substitution contextuelle*
- Algorithme : pour chaque glyphe "
  - si " est précédé par un glyphe autre qu'une parenthèse, on le remplace par »
  - sinon on le remplace par «

C'est un exercice « jouet », très simple, mais qui peut-être amélioré !

Exercice : des guillemets français (II)

```lua
local GLYF = node.id("glyph")
function guillemets(head, tail)
  for glyph in node.traverse_id(GLYF, head) do
    if glyph.char == 34 then
      if glyph.prev and glyph.prev.id == GLYF
            and not (glyph.prev.char == 40)
      then
        glyph.char = 187
      else
        glyph.char = 171
      end
    end
  end
  node.ligaturing(head)
end
```

Code Lua

Exercice : des guillemets français (III)

```
\luadirect{
  luatexbase.add_to_callback("ligaturing", guillemets,
  "Guillemets")
}

Voici un "test" !

\luadirect{
   luatexbase.remove_from_callback("ligaturing","Guillemets")
}
```

Voici un «test»!

Guillemets, amélioration (I)

```lua
function guillemets(head, tail)
  local GLYF = node.id("glyph")
  for glyph in node.traverse_glyph(head) do
    if glyph.char == 34 then
      local fine = node.new(GLYF)
      fine.font=font.current()
      fine.char = 160
      if glyph.prev and glyph.prev.id == GLYF and not (glyph.prev.char == 40)
      then
        glyph.char = 187
        head,fine = node.insert_before(head,glyph,fine)
      else
        glyph.char = 171
        head,fine = node.insert_after(head,glyph,fine)
      end
    end
  end
  node.ligaturing(head)
end
```

Code Lua

Guillemets, amélioration (II)

```
\luadirect{
  luatexbase.add_to_callback("ligaturing", guillemets,
  "Guillemets")
}

Voici un "test" !

\luadirect{
   luatexbase.remove_from_callback("ligaturing","Guillemets")
}
```

Voici un « test »!

Conclusion

Et bien plus!

- simple introduction

Et bien plus!

- simple introduction
- de nombreux autres aspects aux callbacks

Et bien plus!

- simple introduction
- de nombreux autres aspects aux callbacks
- article dans la prochaine *Lettre* GUTenberg avec plus d'exemples

Et bien plus!

- simple introduction
- de nombreux autres aspects aux callbacks
- article dans la prochaine *Lettre* GUTenberg avec plus d'exemples
- seulement un aspect de LuaTeX, bien d'autres aussi intéressants!

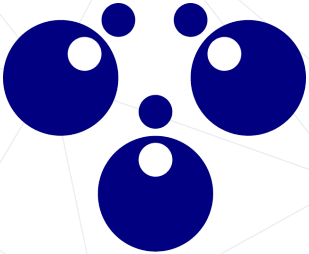
Bibliographie

[1] Paul ISAMBERT. « LuaTEX : What it takes to make a paragraph ». In : *TUGboat* 32.1 (2011).

[2] Paul ISAMBERT. « Three things you can do with LuaTEX that would be extremely painful otherwise ». In : *TUGboat* 31.3 (2010).

[3] *Cahiers GUTenberg : Introduction à LuaTEX* 2010.54-55 (2010). URL : http://www.numdam.org/issues/CG_2010___54-55/.

[4] THE LUATEX TEAM. *The luatex package. The LuaTeX engine.* URL : http://luatex.org (visité le 12/09/2024).



Merci

Du verbatim (I)

- callback `process_input_buffer(string)`
- à ce stade, la ligne n'a pas été traitée du tout
- peut servir à composer du texte verbatim

```lua
local verb_table
local function store_lines (str)
  if str == "\\Endverbatim" then
    luatexbase.remove_from_callback("process_input_buffer",
    "Store lines of verbatim")
  else
    table.insert(verb_table, str)
  end
  return ""
end
```

Code Lua

Du verbatim (II)

```lua
function register_verbatim ()
  verb_table = {}
  luatexbase.add_to_callback("process_input_buffer",
    store_lines, "Store lines of verbatim")
end
```

Code Lua

```
\newcommand\myVerbatim{\luadirect{register_verbatim()}}
```

Du verbatim (III)

```lua
function print_lines (catcode)
  if catcode then
    tex.print(catcode, verb_table)
  else
    tex.print(verb_table)
  end
end
```

Code Lua

Du verbatim (IV)

```
\newcatcodetable\verbcatcode
\newcommand\createcatcodes{
  \begingroup
  \catcode`\\ = 12
  \catcode`\{ = 12
  \catcode`\} = 12
  \catcode`\$ = 12
  \catcode`\& = 12
  \catcode`\# = 12
  \catcode`\^ = 12
  \catcode`\_ = 12
  \catcode`\% = 12
  \catcode`\ = 13
  \catcode`\^^M=13
\savecatcodetable\verbcatcode
\endgroup}
\createcatcodes
```

Du verbatim (V)

```
\newcommand\useverbatim{%
  \luadirect{print_lines()}%
}
\newcommand\printverbatim{%
  \bgroup\parindent=0pt \ttfamily
  \luadirect{
    print_lines(luatexbase.catcodetables.verbcatcode)
  }
  \egroup
}
```

```
\myVerbatim
\newcommand\myluatex{%
   Lua\kern-.01em\TeX
}%
\Endverbatim
\useverbatim
Avec le code :\par
\printverbatim\par
on définit la commande
permettant de générer
\myluatex.
```

Avec le code :
\newcommand\myluatex{%
Lua\kern-.01em\TeX
}%
on définit la commande permettant de générer LuaTeX.