# Utilitaires
## pour la manipulation de fontes OpenType

Jacques André*

Version provisoire du 14 mai 2022

**Résumé**

Cette note, évolutive, donne quelques procédures utilitaires pour manipuler, analyser, etc. les fontes OpenType en vue de leur utilisation avec LuaLaTeX. Voir introduction (page 2) pour plus de détails.

## Sommaire



---

Ce présent document est à l'url `http:jacques-andre.fr/fontex/utilitaires+OpenType.pdf`.
Les liens « cliquables » sont de couleur sienne.
* `Jacques.AndreNN@gmail.com` où NN=35.

# Introduction

Pour des travaux personnels, j'ai eu depuis quelques années besoin d'écrire en LaTeX divers documents où je devais utiliser des fontes OpenType. Et ainsi de développer diverses commandes, macros, etc. que j'ai peu à peu améliorées. M'étant rendu compte que beaucoup d'amis avaient les mêmes besoins, on a échangé à ce sujet et je me suis rendu compte que certaines de mes macros pouvaient avoir un intérêt plus large que mon propre usage privé. J'avais pensé faire un *package* de tout ça. Mais outre le côté très disparate de mes besoins, l'aspect très évolutif de ces commandes m'a fait reculer, d'autant que je continue à penser qu'il doit exister des procédures `lua` qui font mieux la même chose, mais je n'arrive pas à en trouver… Alors en attendant, j'ai accepté de réunir ces divers programmes et de les présenter de manière brute dans cette note…

Ces notes sont destinées à des utilisateurs de LaTeX, et même de LuaLaTeX, qu'ils sont donc censés posséder; il y a donc beaucoup de non-dit dans mon document. Toutetois, je fais quelques rappels, page suivante, sur Unicode et OpenType.
Je me suis efforcé de n'utiliser que des fontes du domaine public, et même en général disponibles dans TeXlive [1].
J'utilise beaucoup le package `picture` qui, dans ce contexte, me donne entière satisfaction sans avoir la lourdeur de *Tikz*.

Je suis bien sûr preneur (par mail, mon adresse est en page 1) de toute remarque, critique, etc. sur ces pages!


1. On trouvera plusieurs de ces fontes citées dans la note de Daniel Flipo, *Catalogue de fontes libres disponibles sous LuaTeX/XeTeX*, décembre 2021, http://daniel.flipo.free.fr/doc/luatex/fonts.pdf

## Rappels sur Unicode

Le codage Unicode est défini ici : https://unicode.org/standard/standard.html. On en est à la version 14 depuis septembre 2021.
À chaque caractère (au sens linguistique pour Unicode) est donné un numéro (ou code, en anglais *codepoint*) et un nom.

**Numéros** On note les numéros soit en décimal (p.ex. 65, code de A), soit en hexa (p.ex. 0x41), soit selon la notation Unicode : U+0041.

- Les numéros qui étaient initialement limités à 65 535 (c.-à-d. 0xFFFF) vont (depuis la version 9 ?) de 0 à 1 114 111 (0x10FFFF). Par abus de langage (?), on dit que le plan [0-FFFF] correspond au codage *Unicode-BMP*, et celui [0-10FFFF] au codage *Unicode-full*.
- Le codage Unicode est défini par plans (ou plages, ou zones...). Pour une liste, voir https://fr.wikipedia.org/wiki/Table_des_caractères_Unicode/U0000
- Certaines zones sont dites « privées » : ce sont là où les dessinateurs (humains ou logiciels) de fontes mettent les glyphes qui ne sont pas des caractères Unicode. Depuis quelques années, sans autre raison visible que le soucis de ne pas occuper une zone qui risquerait d'être occupée par d'autres, des fontes, typiquement *BaskervilleF* ou *EBGaramond*, mettent leurs ligatures, supérieures, petites capitales, etc. non pas dans la *Supplementary Private Use Area-B*, qui va de [U+100000 à U+10FFFD], mais carrément en dehors d'Unicode-full, typiquement après U+110000. Ces « très grands nombres » causent quelques problèmes quand utilisés par LaTeX(cf page ?).
- Les numéros n'ont théoriquement rien à voir (directement) avec le mode d'écriture condensée des caractères (UTF-8 & co).

**Noms** À chaque caractère est associé un nom, écrit traditionnellement en petites capitales, par exemple à U+00A1 (pour le caractère « ¡ ») correspond le nom INVERTED EXCLAMATION MARK. Il existe une version officielle de ces noms en français, donnée en http://hapax.qc.ca/ListeNoms-14.0.0.txt. Ce même caractère « ¡ » s'appelle POINT D'EXCLAMATION RENVERSÉ.

## Rappels sur OpenType

Mes fichiers .lua sont (par exemple) dans `/Users/jacquesandre/Library/texlive/2021/texmf-var/luatex-cache/generic/fonts/otl/garamondpremrpro-italic.lua`

# 1 Inventaire+casseau.ltx
# Inventaire des glyphes d'un casseau

## Objet

On appelle casseau des fontes contenant en général moins d'une centaine de caractères, lesquels sont souvent « non alphabétique ». Typiquement, ce sont des collections d'ornements, de signes spéciaux, mais aussi de lettres de titrage. Il est alors intéressant pour un utilisateur de connaitre le contenu exact de ces fontes.
Pour les fontes TEX, on utilise dans ce cas la procédure `nfssfonts` , mais elle n'est pas utilisable avec les fontes comOpenType ni même `Type1` ou `TrueType`. D'où l'intérêt de celle-ci.

Ce programme a été écrit vers 2010 avec l'aide René Fritz et d'Arthur Reuthenauer (merci...). Il existe une version de ce programme beaucoup plus générale, `Inventaire+OpenType.ltx`, que l'on trouvera ici après en section 2.

## Principe

On balaye la table des caractères de la fonte en suivant les numéros de codage séquentiellement, de 0 (on pourrait partir de 20 car les premiers caractères Ascii ne sont en principe pas affichables, mais on a vu des fontes qui ne respectent pas ça) à 255 (limite de Latin-1). Si le caractère existe dans la fonte on imprime son code (en hexadécimal selon la tradition Unicode), son glyphe et, pour aider à voir l'équivalence clavier [2] alphabétique/casseau, le caractère de latin-1 ayant le même code.

## Programme

Le programme `Inventaire+casseau.ltx` est listé page 36 et est téléchargeable depuis mon site à `http:jacques-andre.fr/fontex/Util+OT.zip`.

2. J'ai écrit un programme où la sortie se fait sur une image d'un clavier, ce qui peut aider par exemple pour les fontes d'ornements (voir `http://jacques-andre.fr/japublis/graphe44.pdf`). Mais cette version est spécifique à un clavier précis. Si quelqu'un en veut une copie, je peux la lui faire parvenir sans problèmes.

## Exemples

En utilisant le fichier des ornements de la fonte *Fourier*[3], `FourierOrns-Regular.otf`, c'est-à-dire en mettant son nom dans la commande
`\newcommand{\mafonte}{FourierOrns-Regular}`
on obtient le résultat de la figure 1.

Ce programme permet aussi de regarder une fonte de caractères de titres [4], pour voir l'allure de ces caps... À titre d'exemple, prenons la fonte casseau associée à la fonte *Infini*[5]. En mettant son nom à son tour dans notre programme, on obtient son inventaire qui se trouve aussi en figure 1. Ça donne un bon aperçu des « pictogrammes » (on n'ose plus parler de lettres) d'*Infini*.

## Notes

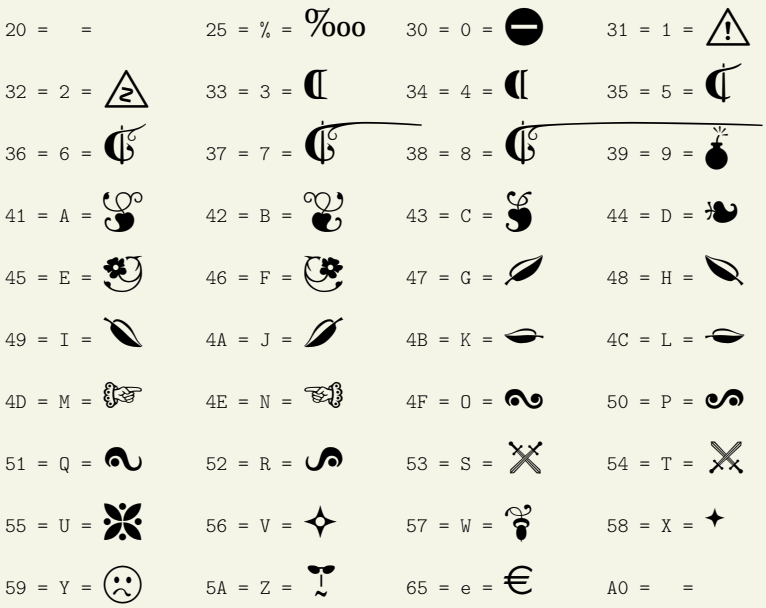
1. On pourrait augmenter le nombre de caractères analysés pour les fontes casseaux ayant plus de 256 caractères. Dans ce cas, on peut alors utiliser la procédure générale.
2. Inversement, on peut analyser ainsi une fonte OpenType très grosse ; mais seuls les 256 premiers caractères seront analysés !
3. Plus important : il faudrait donner pour chaque caractère son *glyphname* : souvent les fichiers de style associés à un casseau (c'est le cas de `fourier.sty`) définissent une commande de même nom que le glyphe. Par exemple, avec *Fourier*, on a :
   – Caractère de code 39 (même code que le chiffre 9) = 💣
   – Nom du glyphe de code 39 : `bomb`
   – Macro de *Fourier* permettant d'avoir ce glyphe : `\bomb`

3. Ce fichier est dans `TeXLive`.

4. Ce sont des fontes qui n'ont souvent que des capitales, parfois appelables par les bas de casse, et sont en fait des fontes casseau alphabétiques.

5. *Infini* est une fonte OpenType commandée par le CNAP à Sandrine Nugue et qui est dans le domaine public : https://www.cnap.fr/sites/infini/telechargement/specimen-infini_web.pdf. Ses caractères de titrage sont repris dans une fonte annexe `Infini-picto.otf`. Sur cette fonte voir *Lettre GUTenberg 45* (à paraitre dans https://www.gutenberg-asso.fr/Lettre-GUTenberg-45-mai-2022).

Glyphes du casseau **FourierOrns-Regular**
Code hexa = car Unicode = glyphe de FourierOrns-Regular




Glyphes du casseau **Infini-picto**
Notation: code hexa = car Unicode = glyphe de Infini-picto

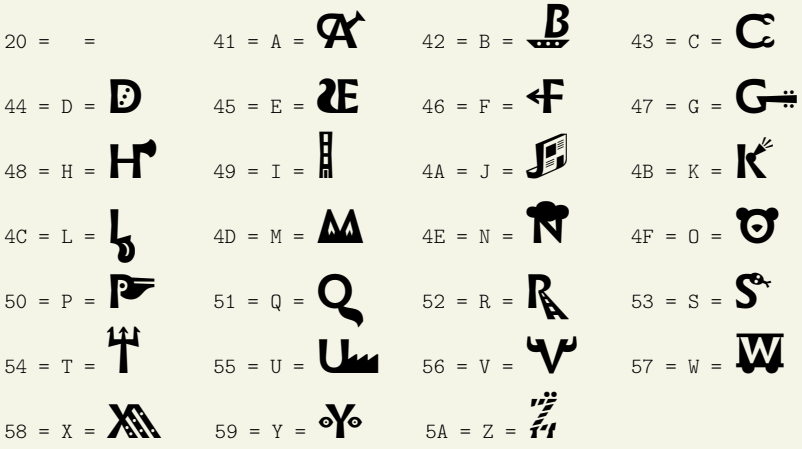



FIGURE 1 – Deux sorties du programme `Inventaire+casseau.ltx`.

# 2 Inventaire+OpenType.ltx
# Inventaire des glyphes d'une fonte OpenType

Ce programme permet de faire l'inventaire des glyphes d'une fonte OpenType. On les donne par ordre croissant de leur numéro (en hexa) avec pour chacun, outre ce numéro, un glyphe et son nom de glyphe.

## Programme

Ce programme a été trouvé sur le web [6] et à peine modifié ici (essentiellement la présentation de sortie, avec l'ajout du *glyphname*). On a donné une version équivalente mais limitée aux 256 premiers caractères d'une fonte casseau. Voir section 1.
Ce programme est listé page 37 et est téléchargeable depuis mon site à `http:jacques-andre.fr/fontex/Util+OT.zip`. Un exemple de sortie est en figure 2.

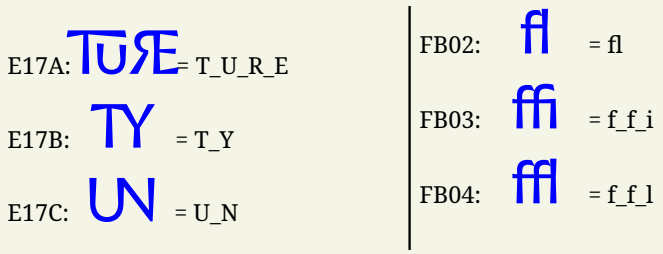



FIGURE 2 – Exemple de la dernière page de l'inventaire des glyphes de la fonte *infini-romain.otf*

## Améliorations et problèmes

La présentation de la sortie peut sûrement être améliorée…
— Lorsque le programme ne trouve pas la fonte demandée, on peut avoir le message *attempt to access a nil value.* =>?

6. https://tex.stackexchange.com/questions/98188/how-can-i-access-a-specific-glyph-in-lualatex-fontspec

— Pour certaines fontes, le programme sort des caractères inexistants, par exemple avec *GaramondPremrPro-Italic.otf* on obtient :
– 2265 ≥ = greaterequal (c'est effectivement GREATER-THAN OR EQUAL TO / SUPÉRIEUR OU ÉGAL À)
– 226A ⊠ = uni226A,
– mais pas 2270.
Or *fontforge* ne montre ni l'un ni l'autre (mais ce sont bien des caractères Unicode U+226A et U+2270 ) . De même, dans garamondpremrpro-italic.lua on trouve bien ["greaterequal"]=8805 (2265 en hexa), mais aucun de code 8810 (0x226A) ni 8816 (0x2270). Il en est de même dans garamondpremrpro-italic.sfd.
D'où viennent ces inventions???
— Réciproquement, ce programme rate des caractères... Par exemple, *fontforge* montre, pour EBGaramond-italic que de nombreux glyphes (en fait toutes les variantes non-Unicode) sont au delà de *Unicode-full*, c'est-à-dire en des positions >0x110000 ; par exemple en 11114970 (0x110362) *fontforge* montre un glyphe de *glyphname* "paragraph.01" (correspondant à une variante de U+00B6 PILCROW SIGN ¶). Ce glyphe n'apparait pas dans la liste fournie par notre programme (qui d'ailleurs s'arrête à 128077 (0x1F44D) qui est cependant supérieur à 65535 (0xFFFF)!). Pourtant, ebgaramond-italic.lua montre bien un glyphe de nom paragraph.01 mais de code 983 906 (0xF0362)
D'où viennent ces lacunes??? En fait la boucle utilisée s'arrête à glyphmax et cette valeur ne contient peut-être pas les variantes non unicode?
En tout cas, la boucle utilisée dans le programme casseau ci-dessus, à condition de la pousser très loin, donne bien ces glyphes très lointains! Mais je ne sais pas trouver le glyphname... À revoir donc!

# 3 AnalOT.pl/tex
# Analyse des substitutions d'une fonte OpenType

Les fontes OpenType sont souvent accompagnées d'un catalogue qui donne bien l'inventaire (parfois exhaustif) des glyphes accessibles, mais plus rarement de la façon précise d'y accéder. « Y-a-t'il, par exemple, un « s terminal » dans la fonte *EBGaramond-Regular* et si oui comme y accéder ? » Pour nous utilisateurs de cette fonte par TEX, ou plutôt par fontspec, le style est assez disert à ce sujet… Il existe de nombreuses procédures `Lua` écrites notamment pour XeTEX puis pour LuaLATEX, mais je n'ai pas trouvé d'analyse d'une fonte OT en donnant les détails des propriétés. J'ai donc essayé d'en écrire, mais je ne suis pas arrivé à quelque chose d'efficace, faute de connaissance des fichiers lua (et notamment sur la base de données correspondants à une fonte OT) utilisés par fontspec notamment. J'utilisais souvent *fontforge* pour « regarder » les fontes, mais ne savais pas comment écrire un programme qui y fasse des recherches. C'est Daniel Flipo (merci encore !) qui m'a suggéré d'utiliser le fichier `.sfd` dans lequel *fontforge* sauve une copie d'une fonte OpenType. Il m'a également indiqué un noyau de programme perl pour faire la recherche de substitutions liées aux ligatures. Dont je me suis inspiré pour écrire les programmes perl d'analyse (et ceux d'impression des résultats, écrits en LuaLATEX) dont je vais expliquer le fonctionnement ci-dessous, après avoir rappelé quelques notions sur *fontforge*.

## Utilisation de *fontforge*

*fontforge* est un logiciel libre de gestion de fontes (création, modification, analyse, sauvegardes, conversion, etc.) de fontes, y compris (et surtout) OpenType. Voir à son sujet, en français, le livre de Yannis Haralambous, *Fontes & codages*[7]. Je vais désormais utiliser la fonte *BaskervilleF-Italic.otf*, dont tout le matériel est dans TEXLive, dans les exemples à venir. Quand on ouvre cette fonte dans *fontforge*, on voit la fenêtre principale de celui-ci comme en fig. 3.

7. Yannis Haralambous, *Fontes & codages*, O'Reilly 2014. Version en ligne : https://hal.archives-ouvertes.fr/hal-02112931

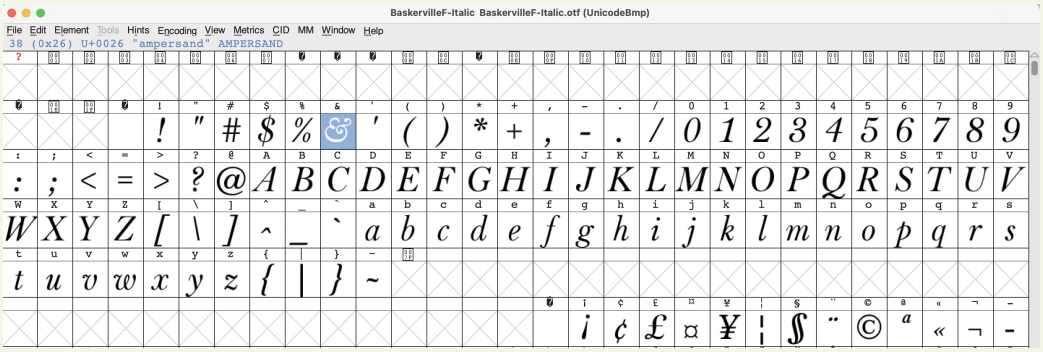



FIGURE 3 – Fenêtre principale de *fontforge* sur le début de la fonte *BaskervilleF-Italic.otf* ; on a cliqué sur le caractère &.

### Propriétés globales de la fonte

Dans la ligne horizontale des menus, en cliquant sur `Element` puis sur `FontInfo` on voit (fig. 4a) d'abord quelques infos globales sur la fonte (nom, copyright, etc.), et dans la colonne de gauche d'autres menus. En cliquant sur <Lookups on fait apparaître (fig. 4a) la liste des substitutions, comme on aurait pu le faire en lançant depuis le terminal la commande `otfinfo -f BaskervilleF-Italic.otf` (avec éventuellement les chemins d'accès à la fonte).

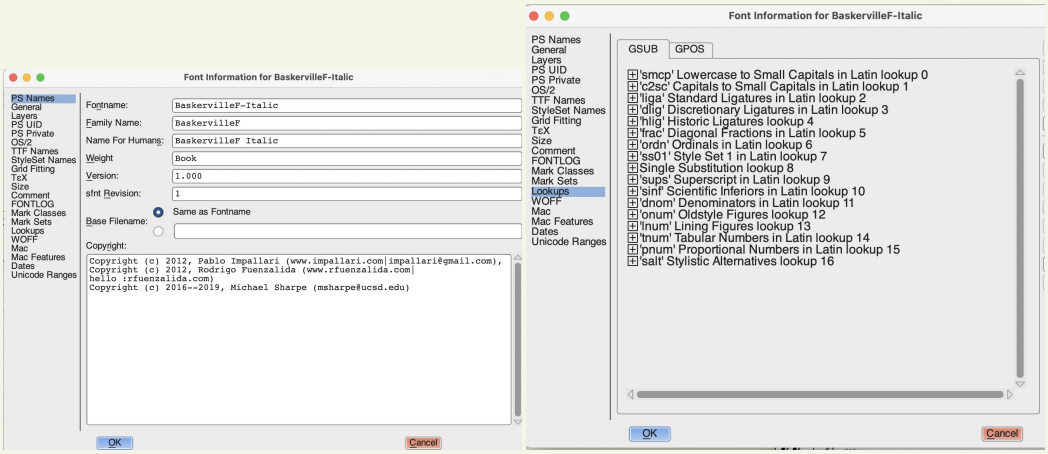



FIGURE 4 – Fenêtres du menu FontInfo de la fonte *EBGaramond-Italic.otf*

### Propriétés des glyphes

Les glyphes sont placés dans l'ordre hexadécimal des numéros Unicode. En cliquant sur une case (en bleu dans cette figure), on fait apparaître (en bleu, sous la ligne de menus, voir fig. 3) la position du caractère dans la fonte – ici en décimal 38 et en hexa (0x26) –, le numéro Unicode : du caractère U+0026, le *glyphname* (bas de casse entre quotes, ici 'ampersand') et le nom Unicode (en caps : AMPERSAND).
Si on clique-droit sur cette case esperluette, on fait apparaitre un menu où on voit `Glyph info`. Si on y clique puis sur `Substitutions` on fait apparaître (fig. 5) les substitutions possible de ce caractère : avec les *features* `c2sc` et `smcp` l'esperluette normale est remplacée par l'esperluette en petites capitales, dont le nom de glyphe est `ampersand.sc` (voir § suivant).
<Pb : comment trouver ligatures liées à un caractère, p.ex. f?>

### Recherche d'un glyphe par son nom

Les glyphes sont classés par numéro hexadécimal. On peut accéder à un glyphe par son nom. Depuis la fenêtre principale, cliquer (menu horizontal) sur `View` puis sur `Goto`; y taper alors le nom du glyphe cherché. Par exemple avec `ampersand.sc` (voir § précédent), on obtient la case de ce glyphe, dont on voit (fig. 6) que ce glyphe est en 65572 (0x10024) et qu'il correspond (à une variante du caractère) AMPERSAND (U+0026). Cette position 65572 correspond à la dernière zone privée prévue par Unicode [8] pour les caractères (en fait les glyphes) non-Unicode. C'est là donc que le dessinateur de la fonte a fait mettre tous les glyphes de substitutions dont, par exemple, les ligatures *fb ffb fh ffh* etc. (voir fig. 6). On verra (page **??**) que ce n'est pas toujours le cas ce qui pose quelques problèmes.
Il faut aussi noter que les fontes ne sont pas toujours parfaites... *fontforge* par exemple les analyse et signale presque toujours (dans une fenêtre spéciale) les erreurs repéées. En particulier, nombre de fontes ont des glyphes qui n'ont pas de nom ou dont le nom n'est pas correct. Dans ce cas, *fontforge* [9] affiche un nom

8. Du moins la dernière zône privée du BMP initial des premières versions d'Unicode; beaucoup de fontes utilisent maintenant la *Supplementary Private Use Area-B,* qui va de [U+100000 à U+10FFFD]; voir page 3.

9. Voir : https://fontforge.org/docs/ui/menus/encodingmenu.html#encodingmenu-namelist qui dit : *Any glyphs you do not explicitly name will be named "uniXXXX" or "uXXXXX" where XXXX is the unicode value in hex. The prefix "uni" will be used for glyphs in the BMP, the prefix "u" for glyphs outside.*

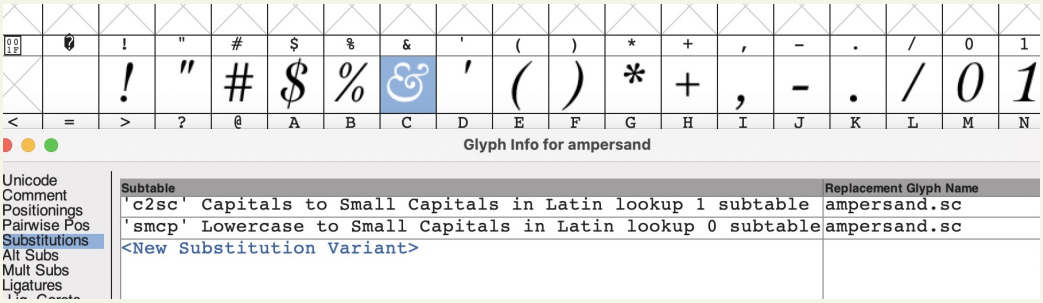



FIGURE 5 – Substitutions liées à un caractère.

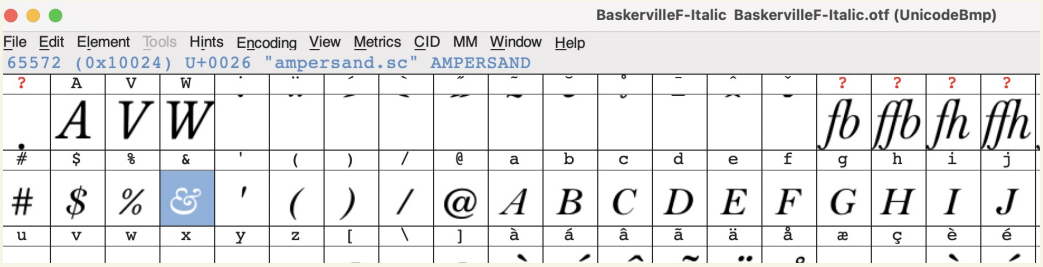



FIGURE 6 – Les glyphes de substitution sont dans des zones privées d'Unicode.

plus ou moins inventé (il me semble que lua utilise la même méthodologie), par exemple pour un glyphe de code hexa XXXX, le nom de glyphe créé sera uniXXXX.

### Production d'un fichier `.sfd`

Une fonte OpenType est un fichier binaire. *fontforge* permet d'en avoir une copie dans un format lisible (en Ascii) : `.sfd = SplineFontDB`.
Pour celà, une fonte étant ouverte, cliquer dans le menu horizontal sur `File/Save`.
En partant de `EBGaramond-Italic.otf` on obtient le fichier `EBGaramond-Italic.sfd`.

Ce fichier a la structure suivante [10] :
Au début, il y a quelques informations globales sur la fonte (exemples tirés de `EBGaramond-Italic.sfd`) :

```
SplineFontDB: 3.2
FontName: BaskervilleF-Italic
FullName: BaskervilleF Italic
```

10. Le format est défini à https://fontforge.org/docs/techref/sfdformat.html

```
FamilyName: BaskervilleF
Weight: Book
Copyright: Copyright (c) 2012, Pablo Impallari (www.impallari.com|impallari@gmail.com),
Copyright (c) 2012, Rodrigo Fuenzalida (www.rfuenzalida.com|hello :rfuenzalida.com)
Copyright (c) 2016--2019, Michael Sharpe (msharpe@ucsd.edu)
Version: 1.000
ItalicAngle: -15
UnderlinePosition: -100
UnderlineWidth: 50
Ascent: 800
Descent: 200
InvalidEm: 0
sfntRevision: 0x00010000
```

qui se comprennent naturellement.
Suivent ensuite de nombreuses tables qui ne nous concernent pas ici, sauf la dernière qui est en fait la table décrivant les glyphes et leurs propriétés. Parenthésée par `BeginChars:...EndChars` elle contient N définitions de caractères, elles-mêmes parenthésées par `StartChar:... EndChar` ; voir figure 7 où chaque ligne du type `Kerns2:...` peut être en plusieurs exemplaires, ou absentes. Ce sont elles qui définissent les features liées à un caractère (kern2 pour le crénage, Ligatures2, substitutions2, etc. [11] comme leur nom l'indique…). Ce sont elles qu'on va étudier!

## Programmes

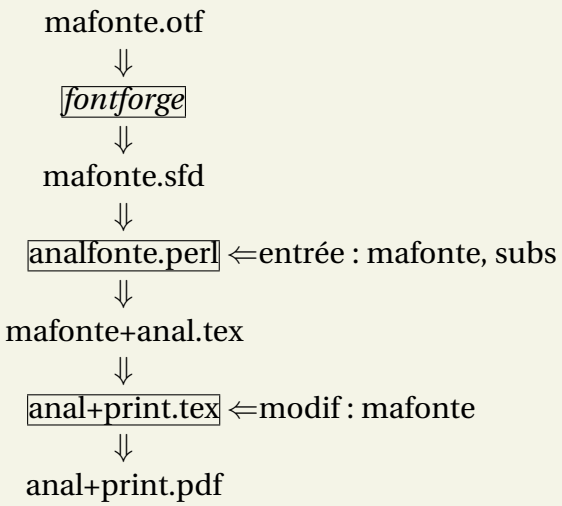
Le travail se fait en trois temps, comme schématisé ci-dessous (où les boîtes encadrées indiquent des programmes) :

11. On a repéré aussi moins les "tags" suivants : Position2, PairPos2, AlternateSubs2, MultipleSubs2, LCarets2.

```
SplineFontDB: 3.2
FontName: ...
... Tables...
BeginChars 65836 760 (nb de car.
                      possibles et réels)
    StartChar: premiernom
    ...
    EndChar
    StartChar: secondnom
    ...
    EndChar
...
    StartChar: derniernom
    ...
    EndChar
EndChars
EndSplineFont
```

```
StartChar: nom
Encoding: Ord., Unicode, GID
Width: chasse
...
Fore
SplineSet
    Description des contours
    du glyphe
EndSplineSet
Kerns2: ...
Position2: ...
PairPos2:...
Ligature2: ...
Substitution2: ...
AlternateSubs2: ...
MultipleSubs2: ...
LCarets2:...
EndChar
```

FIGURE 7 – Structure d'un fichier .sfd : à gauche, table des glyphes ; à droite, détail de la description de chaque glyphe




La première étape, production du fichier mafonte.sfd vient d'être décrite page 13.

**Programme PERL**

En fait, on dispose de deux fichiers perl, l'un pour analyser les substitutions de type ligature, l'autre pour les autres substitutions. C'est de façon provisoire, en attendant de faire un seul programme permettant de traiter tous les cas (pour le moment on ne traite que les substitutions « simples » et ligatures, pas par exemple le crénage dans les tables kern2 et celles citées en note 11) voire tous ensembles.
Cette distinction vient de ce que le premier type de substitution est indiqué dans le fichier .sfd par des lignes du type

```
StartChar: f_i
...
Ligature2: "'liga' Discretionary Ligatures lookup 39 subtable" f i
```

où les informations à extraire sont : `liga f i f_i`
et pour les autres par

```
StartChar: a
...
Substitution2: "'smcp' Lowercase to Small Capitals lookup 26 subtable" a.sc
```

d'où nous extrairons `smcp a a.sc`.
Ces programmes demandent à l'utilisateur le nom `mafonte` de la fonte ce qui lui permettra de lire le fichier `mafonte.sfd` et d'écrire les résultats dans `mafonte+anal.tex` (qui sera lu par le programme .tex suivant), et le nom de la substitution à étudier (ce nom a en général 4 lettres où, on peut remplacer l'une d'elle par un « . » pour signifier comme dans les expressions régulières que l'on accepte n'importe quelle lette ; par exemple « .lig » permet de trouver alig, dlig, clig, etc.).
Le langage de sortie est du .tex utilisant quelques commandes explicitées dans le § suivant.

Ces programmes sont respectivement :

**analOT+lig.pl** pour les ligatures.
Voir listing en page 41.

**analOT+subs.pl** pour les ligatures.
Voir listing en page 39.

Ces fichiers sont téléchargeables depuis `http:jacques-andre.fr/fontex/Util+OT.zip`

### Listing résultat par LuaLaTeX (`analOT+print.ltx`

Les sorties des deux programmes perl `analOT+subs.pl` et `analOT+lig.pl` se font dans un fichier `mafonte+anal.tex` qui est appelé (par `\input`) depuis un unique programme `analOT+print.tex` pour imprimer les résultats (donc dans le fichier `analOT+print.pdf`).
Les fichiers `mafonte+anal.tex` commencent par un passage de paramètres :

```
\def\JAN{mafonte}
\def\JAD{date système}
\def\JAF{substitution}
\JAGO
```

où `\JAGO` est le début d'un environnement de mise en colonnes des résultats ; ils sont formés de lignes du type

```
'sups'<zero>  \parnom{zero} \FL  \parnom{zero.superior} <zero.superior>\\
'dlig' \parnom{R} \parnom{E} \FL  \parnom{R_E} <U+E172>\\
```

où
– 'subs' ou 'dlig' est le nom de la substitution étudiée,
– \parnom l'appel à une commande définie dans notre programme analOT+print.tex et qui imprime un caractère de nom (ici) zero, R, E, R_E
– <...> indique le nom du glyphe
–<U+xxxx> indique le code hexa
– \FL un séparateur imprimant (aujourd'hui) la flèche ⇒.
Ce programme n'est a priori qu'un programme d'édition, mais il suppose une connaissance de lua. En effet, le résultat fourni par nos programmes perl fait appel à la notion d'appel d'un glyphe par son nom (matérialisé ici par la procédure \parnom) qui nous oblige donc à appeler une procédure lua (trouvée sur le site de StackExchange [12]). J'ai été obligé de la modifier en cas de non-résultat (voir la procédure \JAERR et l'utilisation de la bombe de Bovanni) mais ça pose encore quelques problèmes (voir page 20).

On trouvera un listing de ce programme 43. Il est dans un fichier téléchargeable depuis http:jacques-andre.fr/fontex/Util+OT.zip

12. https://tex.stackexchange.com/questions/502907/moving-luacode-to-lua-file-accessing-glyphs-by-name.

**Exemple**

Avec la fonte (de TeXLive) `EBGaramond-Italic.otf`, en travaillant dans un répertoir TMP où se trouve les fichiers perl et tex à utiliser.

1. On ouvre un répertoire TMP où on met la fonte `EBGaramond-Italic.otf` et les fichiers perl, lua et tex à utiliser.
2. On ouvre *fontforge* avec `EBGaramond-Italic.otf` et par la commande Save, on sort le fichier `EBGaramond-Italic.sfd` dans TMP.
3. Depuis le terminal, dans TMP, on lance la commande
   `perl anaOTsubs.pl+`
4. On lui donne les informations demandées, à savoir ici
   `EBGaramond-Italic`
   `swsh`
5. Quand le programme perl s'arrête, on reçoit le message
   `c'est fini : sortie dans EBGaramond-italic`anal.tex+
   Ce fichier commence comme suit :

```
\def\JAN{ebgaramond-italic}
\def\JAD{Mar  5 avr 2022 11:27:33 CEST }
\def\JAF{dlig}
\JAGO
'dlig' \ \parnom{Q} \ \parnom{y} \FL  \ \parnom{Q_y} <U+1100B2>\\
'dlig' \ \parnom{T} \ \parnom{h} \FL  \ \parnom{T_h} <U+1100B3>\\
```

6. On ouvre alors (p.ex. avec TexShop) le fichier `analOR+print.ltx` et en 3e ligne on écrit
   `\newcommand\myfont{EBGaramond-italic}`
   et on l'exécute.
7. On trouve le résultat dans `analOR+print.pdf` dont le début est

*«Étude de la fonte EBGaramond-italic*
*faite le «Mar 5 avr 2022 11:27:33 CEST » à partir du fichier* `EBGaramond-italic.sfd`
*Liste des glyphes substitués par la feature*
dlig *de EBGaramond-italic*

*'dlig' Q y⇒ Qy<U+1100B2>*
*'dlig' T h⇒ Th<U+1100B3>*
*'dlig' a s⇒ aſ<U+1100B4>*
*'dlig' c h⇒ ch<U+1100B5>*
*'dlig' s s⇒ ſſ<U+1100CA>*
*'dlig' ſ b⇒ ſb<U+1100CB>*
*'dlig' ſ h⇒ ſh<U+1100CC>*
*'dlig' ſ i⇒ ſi<U+1100CD>*

## Limites et problèmes

La méthode pêche par un point qui n'est d'ailleurs pas apparu de suite… : on ne travaille jamais dans cet ensemble de programmes sur la fonte `mafonte.otf`, mais d'une part sur la version transcrite par *fontforge*, `mafonte.sfd`, et, d'autre part, sur celle `mafonte.lua` transcrite par lua. Le problème vient de ce que ces deux produits, *fontforge* et lua, n'ont pas des transcriptions tout à fait identiques, les deux faisant par exemple des modifications de position (c'est-à-dire de codage des glyphes, typiquement ramener dans le BMP ce que les dessinateurs mettent souvent maintenant dans la dernière zone privée d'Unicode) ou de noms de glyphe (lua a l'air d'accepter les noms vides, *fontforge* pas, voir note 9).

Voici un exemple, avec la fonte `EBGaramond-Italic.otf`
En 64261 (0xfb05), *fontforge* affiche le glyphe *ft* de la ligature du longs avec t, mais non pas sous le nom canonique `longs_t` mais sous celui `uniFB05` (on est dans un ds cas de la note 9)
Dans `EBGaramond-Italic.sfd`, on a, pour l'entrée de longs, sous le nom de `uniFB05` donc, les lignes

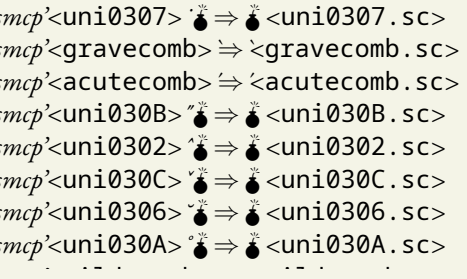
```
StartChar: uniFB05
...
Ligature2: "'dlig' Discretionary Ligatures in Latin lookup 37 subtable" longs t
```

que notre programme perl traduit donc (en inversant l'ordre) par
`'dlig' \ \parnom{longs} \ \parnom{t} \FL  \ \parnom{uniFB05} <U+FB05>`

Mais si on regarde maintenant la version lua de notre fonte, `EBGaramond-Italic.lua`, on n'y trouve pas non plus de `longs_t`, alors qu'il y a bien `"longs"`, `"longs.sc"`, `"longs_b"]`, etc. Mais on voit, à l'entrée 64261 (0xfb05) :

```
 [64261]={
   ["boundingbox"]={ -194, -290, 594, 705 },
   ["index"]=1004,
   ["unicode"]=64261,
   ["width"]=557,
  },
```

où il n'y a pas de champ `name` qui devrait être `["name"]="longs_t"`.
Bref, *fontforge* appelle le glyphe `uniFB05` et lua ne lui donne aucun nom. Or, la sortie de nore programme perl, basée sur le fichier .sfd, reprend la dénomination par nom, et demande `\parnom{uniFB05}`. Cette macro `parnom` appelle une procédure lua, qui étudie donc le fichier .lua où il n'y a pas ce nom `uniFB05` et signale donc une erreur. J'ai essayé d'aller un peu plus loin, en partant sur le fait que les noms de type `uniXXXX` indiquent les glyphes dont le code est `XXXX`. Je corrige donc l'erreur de `\parnom` et imprime alors le bon glyphe. Quant aux autres cas ("affi..." "u...") je ne sais pas trop comment faire (et n'ai d'ailleurs pas la liste des cas possibles). Et encore moins quand *fontforge* invente un nom à partir d'un nom inventé, par exemple (voir fig. ci-dessous) `uni0307.sc` ! tous les cas j'imprime une bombe (💣) pour signaler qu'il y a un soucis. D'où parfois des sorties surchargées, comme (pour EBGaramond-Italic, avec recherche de smcp) :



# 4 Autres programmes

Je donne ici des programmes qui ne sont pas directement liés aux fontes OpenType mais qui ont nécessité certaines manipulations de fontes peu courantes ce qui peut aider d'autres utilisateurs de TEX.

## 4.1 `Metrique`
### Macros pour montrer la métrique de caractères

Depuis des années, il m'est arrivé d'avoir à comparer des caractères en plomb entre eux, ou avec la trace imprimée qu'ils ont laissée, ou les comparer avec les caractères numériques de fontes dans des formats divers (notamment PostScript, TrueType, Type1 et maintenant OpenType). Si on veut faire les choses proprement et de façon portable, il faut un minim:um de connaissances sur la typographie traditionnelle et sur les formats de fontes numériques. Un article en cours [13] m'a amené à écrire ces macros que je viens (mars 2022) d'utiliser dans une note pour la *Lettre GUTenberg* 45...
Le problème est de dessiner la figure suivante et plus précisément les cadres englobant les images de caractère, en partant de la seule connaissance de ces caractères.

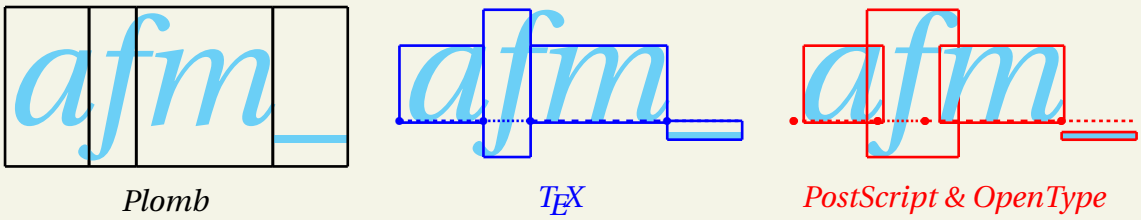



FIGURE 8 – Trois méthodes de composition.

13. Sur le concept de mesure en histoire de la typographie. Il devrait paraître, en collaboration avec William Kemp, dans le premier numéro de *Le Pot cassé* , lrevue francophone de typographie, encore en gestation...

Précisons dès à présent que les schémas de la figure 8 ne sont pas des fictions, même si parfois la réalité peut surprendre : le *f* italique en plomb (figure 8-gauche) déborde bien du « caractère » en plomb, comme le montre la photo ci-contre [14]. Il s'agit d'un crénage (au sens des fondeurs de caractère et non au sens du *kerning* c'est-à-dire de la réduction de l'approche entre lettres, même si ça revient au même dans ce cas). De même en TEX!

### Cas du plomb

Le problème du plomb est que l'on ne sait pas, au seul vu d'un caractère imprimé, où se situent les parties supérieure et inférieure du type (du caractère en plomb). Mais si on regarde la technique utilisée par les fondeurs jusqu'au xx$^{e}$ siècle, on voit que le principe est de frapper la matrice (où sera moulé le plomb) de façon que la moitié des bas-de-casse (sans hampe ni descendante) soit au centre du type. Fournier (*Manuel typographique*, 1764) précise même qu'il faut prendre le milieu du « m ». La figure 9 donne explicitement la méthode à employer : prendre le caractère m ; diviser sa hauteur (connue par `\settoheight{\mm}{m}`) par 2, ce qui détermine le point bleu. Le bord supérieur est à 1/2 em au dessus de ce point, et l'inférieur à 1/2 em en dessous. Le bord droit est défini par la chasse du caractère (ici un g).

### Cas des fontes TEX

Ici, aucun problème : la boîte du caractère (boîte au sens TEX) est la boîte cherchée. Figure 12. La seule difficulté est de ne pas croire que les boîtes TEX englobent le caractère qu'elles contiennent (voir le f de la figure 8). Le fichier `.tfm` contient pour chaque caractère les trois nombres donnant (en em) *width, depth* et *height* (fig. 12.

### Cas des fontes OpenType

En fait cette métrique existait déjà pour les fontes PostScript, et a été utilisée aussi pour Type1 puis TrueType. Finalement, OpenType a repris cette même

14. On n'y montre pas le caractère « _ » qui n'existe en général pas dans les fontes en plomb !

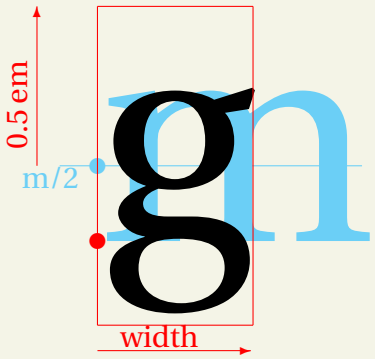



FIGURE 9 – Évaluation de la métrique des types en plomb.

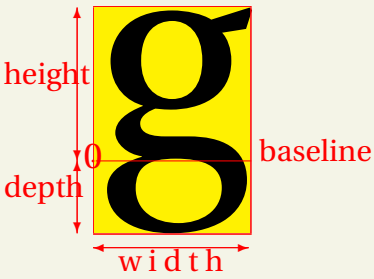



FIGURE 10 – Le calcul de la métrique des fontes TEX est immédiat.

façon de donner la métrique des caractères. En PostScript [15], elle consiste à donner la *bbox* (*bounding box*, boîte englobante ou plus petit rectangle exinscrit au caractère), caractérisée par son coin inférieur gauche *(lower left)*, de coordonnées *(llx,lly)* et son coin supérieur droit *(uper right)*, de coordonnées *(urx, ury)*. Les autres formats ont des notations légèrement différentes [16] La raison d'être de cette boîte, c'est de déterminer l'espace minimum et suffisant pour conserver en mémoire cache le bitmap du caractère une fois calculé à la première fois, pour le récupérer aux prochaines occurrences de ce même caractère dans le même contexte graphique (au lieu de tout recalculer).
Hélas, TEX seul ne permet pas de calculer, ou trouver, les coordonnées de la *bbox* à partir des seules données liées à la boîte d'un caractère (voir fig. 11). Or on sait que ces valeurs sont dans les infos d'une fonte ; pour PostScript, elles étaient dans le fichier `.afm` (voir note 15), l'équivalent des `.tfm` de TEX. Il est donc normal, puisqu'on travaille avec LuaLATEX, de faire appel à une procédure Lua pour trouver ces valeurs. Après quelques recherches, on a trouvé ce que nous voulions sur le site de StackExchange [17] qui nous a permis d'écrire de quoi importer ces valeurs *(llx,lly) (urx, ury)* (fig. 12), ce que l'on décrit dans la section qui suit.

15. Voir Jacques André et Justin Bur, Métrique des fontes PostScript, *Cahier GUTenberg* 8, 1991, p. 29-50. http://www.numdam.org/item/CG_1991___8_29_0.pdf

16. La procédure lua qu'on utilise (voir ci-dessous) appelle *(leftsidebearing, bottomsidebearing) (rightsidebearing, topsidebearing)* ce que PosScript appelle *(llx, lly) (urx, ury)*.

17. https://tex.stackexchange.com/questions/522748/lualatex-glyph-dimension-correctness-depends-on-font-extension

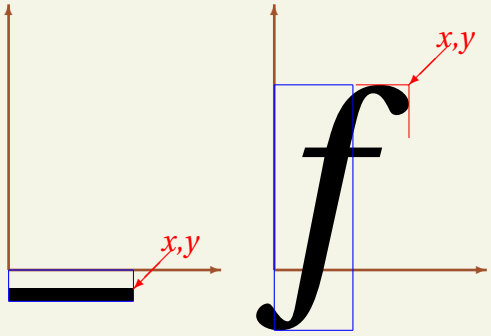



FIGURE 11 – Si TEX permet de calculer les ordonnées $y$ des deux points indiqués (elles sont données par `\settoheight`), il n'en est pas de même du $x$ de *f*.

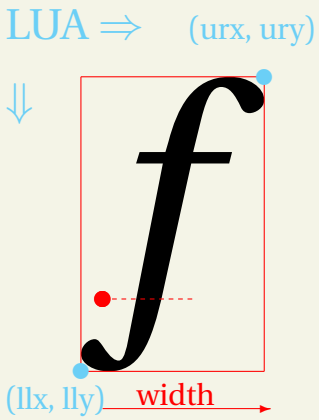



FIGURE 12 – Pour tracer en TEX la métrique OpenType, il faut une procédure extérieure (ici Lua) pour connaître les valeurs *llx, lly, urx ury* !

## Programme

Notre programme comprend en fait deux choses :

1. un programme, `psbbox.lua`, issu (après légères modifications) de StackExchange (voir note 17) ; on en trouvera la listing intégral page 45.
2. un programme d'essais `metrique.tex` (listing page 47) formé essentiellement de
   – une macro, `psbbox`, qui donne les *llx, lly, urx, ury* d'un caractère
   – une macro, `metrique`, qui dessine un caractère dans ses métriques
   – une procédure en `directlua` permettant à la macro `psbbox` permettant d'appeler le programme lua `psbbox.lua`
   – et diverses définitions de variables et deux essais.

**La macro `psbbox`** Elle est définie comme suit :

```
\newcommand{\bbox}[1]{% param #1= code hexa du caractère
\global\llx=\lsidebearing{#1}\relax%
\global\urx=\rsidebearing{#1}\relax%
\global\lly=\bsidebearing{#1}\relax%
\global\ury=\tsidebearing{#1}\relax}%
```

et se contente d'affecter à *llx, lly, urx, ury* les coordonnées de la *bounding box* du caractère dont le code hexa est passé en paramètre, lesquelles coordonnées ont été fournies par le programme extérieur `psbbox.lua`.

**La macro `metrique`** Cette macro sert à montrer les boîtes d'un caractère avec les métriques du plomb, ou de TEX, ou de PostScript-OpenType. NB. En français, le « ou » n'est pas exclusif, c'est-à-dire que cette macro peut montrer le même caractère vue par la métrique de TEX *et* vu par celle d'OpenType.
Cette macro demande 6 paramètres :
#1 : le (code du) caractère à analyser ;
#2 : la couleur avec laquelle on veut l'imprimer
#3 à #6 : quatre paramètres qui peuvent être vides.
– S'ils ne le sont pas, ils donnent respectivement :
#3 la couleur du cadre de la métrique au plomb
#4 la couleur du cadre de la métrique TEX
#5 la couleur du cadre de la métrique OpenType

#6 la couleur du point courant et de la ligne de base

– S'ils sont vides, le cadre correspondant n'est pas tracé.

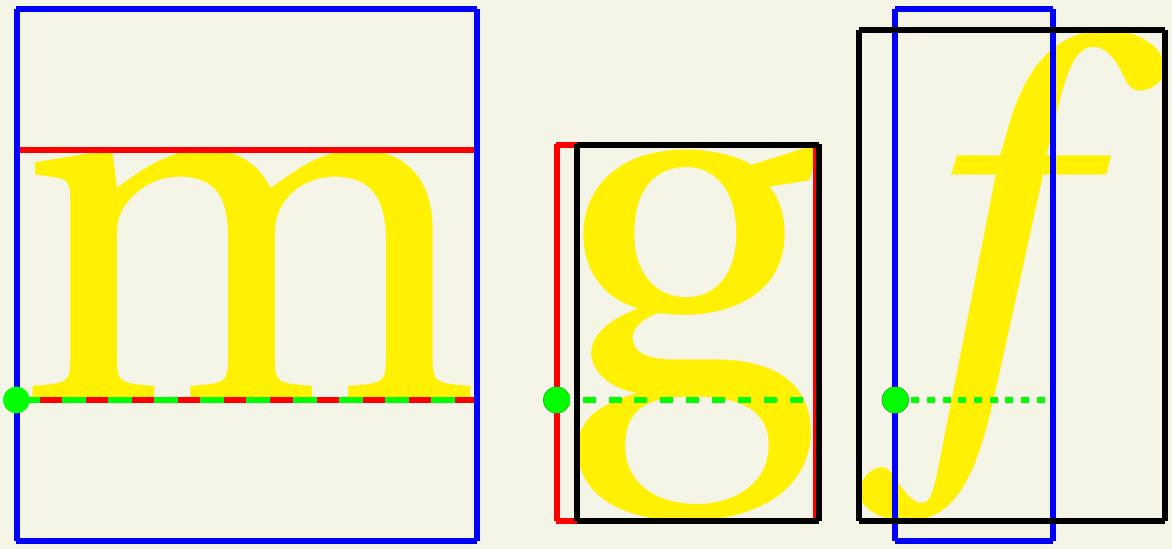
Enfin, elle utilise la valeur donnée par `\linethickness{n pt}`pour tracer les boîtes.

Elle est bien sûre modifiable « à la main » pour des besoins différents…Elle est décrite entièrement dans le programme `metrique.tex` (voir page 47). Une fois comment tracer la boîte du plomb (voir ci-dessus), et comment trouver les valeurs de *llx, lly, urx* et de *ury*, elle ne présente pas de difficulté particulière!

**Exemples d'emploi de `metrique`** Le programme `metrique.tex` donne deux exemples (voir page 47), que voici :

```
\linethickness{2pt}%
\fontsize{200}{100}\selectfont%\
\metrique{m}{yellow}{blue}{red}{}{green}% comparaison plomb et TeX
\metrique{g}{yellow}{}{red}{black}{green}% comparaison TeX et OT
\itshape\metrique{f}{yellow}{blue}{}{black}{green}% comparaison plomb et OT
```

dont la sortie est :



Le second exemple est celui que nous avons utilisé pour la figure 8 plus haut.
Le voici :

```
\begin{figure}[h]%
\setlength{\fboxsep}{0em}\linethickness{1pt}%
\centerline{\fontsize{60}{60}\selectfont\itshape%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{black}{}{}{}%
\metrique{f}{bleu}{black}{}{}{}%
\metrique{m}{bleu}{black}{}{}{}%
\metrique{\_}{bleu}{black}{}{}{}\\[.3em]
\normalsize\color{black}Plomb\end{center}\end{minipage}\hspace*{1cm}%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{}{blue}{}{blue}%
\metrique{f}{bleu}{}{blue}{}{blue}%
\metrique{m}{bleu}{}{blue}{}{blue}%
\metrique{\_}{bleu}{}{blue}{}{blue}\\[.3em]
\normalsize\color{blue}\TeX\end{center}\end{minipage}\hspace*{1cm}%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{}{}{red}{red}%
\metrique{f}{bleu}{}{}{red}{red}%
\metrique{m}{bleu}{}{}{red}{red}%
\metrique{\Uchar"5F}{bleu}{}{}{red}{red}\\[.3em]
\normalsize\color{red}PostScript \& OpenType\end{center}\end{minipage}
}
\caption{Trois méthodes de composition.}\label{afm}
\end{figure}
```

## 4.2 `Leicester` Titres en pleine justification

Quand un graphiste veut mettre un titre en pleine justification – dans la presse, notamment pour les magazines, c'est une coutume courante – il le fait souvent en augment la taille des espaces justifiantes voire l'interlettrage (c'est le *track kerning*). Il y a une autre méthode, qu'en Grande-Bretagne on appelle *Leicester*[18] et qui consiste à calculer le corps pour que le titre couvre la justification. La figure **??** compare ces méthodes.

### Méthode

Pour trouver quel corps donner à un bout de texte pour qu'il ait une chasse égale à la justification, une simple règle de trois suffit (en comparant avec par exemple un corps 100). Mais je me suis rendu compte que faire en TEX une division entre deux longueurs pour avoir leur rapport n'était pas aussi simple que ça. Heureusement, `StackExchange` [19] a, comme souvent, une solution que j'ai donc utilisée. C'est ce calcul qui m'a incité à mettre cette procédure ici.

### Programme

Voici le programme complet, avec les définitions encadrées de %%% et un exemple d'emploi dont la sortie est montrée après.

```
%!TEX encoding = UTF-8 Unicode
\documentclass{article} %démo justif à la Leicester
\usepackage{fontspec}
 \setmainfont{Erewhon}
 %%%%% Matériel pour Leicester
\usepackage{xfp}
\makeatletter%
 \newcommand{\mytextsize}{\f@size}%
```

18. Du nom de la ville où elle aurait été utilisée en premier. J'ai entendu ce nom à Londres il y a plus de quarante ans, mais aujourd'hui je ne le retrove pas dans ce contexte sur le web…J'appelle quand-même ma procédure ainsi!

19. En l'occurrence à https://tex.stackexchange.com/questions/6417/how-to-find-the-ratio-of-a-length-command-e-g-textwidth-to-a-reference-valu. Merci aux auteurs des réponses!

```
 \newcommand*{\getlength}[1]{\strip@pt#1}%
 \newcommand*{\getlengthencm}[1]{%
  \strip@pt\dimexpr0.0351510\dimexpr#1\relax\relax}%
 \newcommand{\@LCSTRcorps}{}%
 \newlength{\@LCSTR}%
\newcommand{\Leicester}[2]{{\fontsize{100}{100}\selectfont%
\settowidth{\@LCSTR}{#1}%
\renewcommand{\@LCSTRcorps}{%
\fpeval{100*\getlengthencm{#2}/\getlengthencm{\@LCSTR}}}%
{\fontsize{\@LCSTRcorps}{\@LCSTRcorps}\selectfont%
#1}}}%
\makeatother
%%%%%
\begin{document}\thispagestyle{empty}\parindent0pt

\newcommand{\textelong}{Il a beaucoup plu sur Rennes}
\newcommand{\textecourt}{Il a fait beau à Rennes}
\Huge
\rule{10cm}{1pt}

\textecourt

\Leicester{\textecourt}{10cm}

\Leicester{\textelong}{10cm}

\textelong

\rule{10cm}{1pt}
\end{document}
```

Programme dont la sortie donne

Il a fait beau à Rennes
Il a fait beau à Rennes
Il a beaucoup plu sur Rennes
Il a beaucoup plu sur Rennes

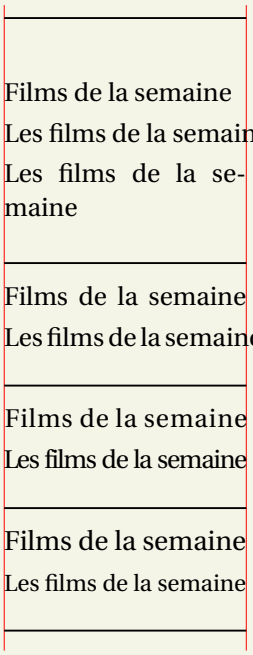
**Exemple 1 : méthodes de justification**

La figure 13 montre diverses méthodes de justification appliquées à de deux textes, l'un court et l'autre un peu plus long, définis par

```
\def\TitreC{Films de la semaine} % titre court
\def\TitreL{Les films de la semaine} % titre Long (un peu plus;-
)
```

qui sont supposés être les titres d'une rubrique, dans un colonne (explicitée ici en rouge) de justification `\majustif`. On présente les résultats dans quatre blocs, séparés par des filets :

1. Composition classique de ces textes considérés comme un paragraphe.
   — `\TitreC` : le titre court étant inférieur à la justif, la ligne est creuse.
   — `\mbox{\titreL}` : le titre L étant mis dans une boîte dépasse la justification et sort de la colonne.
   — `\TitreL` : le texte est justifié « normalement » ; mais avec de grandes espaces, faute de points de coupure et la justification étant très petite.
2. Composition de ces deux titreses de façon qu'il soient justifiés sur toute la largeur de la colonne. La méthode utilisée ici consiste à modifier les espaces-mots ; pour le texte court, elles sont agrandies mais pour le long, elles sont réduites au maximum (on pourrait modifier la taille minimale entre deux mots définie par défaut par TEX, mais les mots ne se distingueraient plus), mais ce n'est pas suffisant et ici encore le texte déborde de la colonne !



```
\setlength{\majustif}{4.5cm}
\begin{minipage}{\majustif}
\titreC
\mbox{\titreL}
\titreL % (texte justifié)

\makebox[\majustif][s]{\titreC}
\makebox[\majustif][s]{\titreL}

\textls[36]{\titreC} % 36 par tatonnement
\textls[-45]{\titreL}% -45 par tatonnement

\Leicester{\titreC}{\the\majustif}
\Leicester{\titreL}{\the\majustif}
\end{minipage}
```

FIGURE 13 – Il y a plusieurs façons de justifier un texte. À droite les programmes-source, à gauche la sortie. Bloc du haut : sorties normales, blocs suivants : par variations de l'espace inter-mots puis par crénage ; en bas, méthode de Leicester. En rouge, les limites de la colonne de justification `\majustif`.

3. Composition par crénage (on modifie les espaces entre lettres en les « crénant ». Pour cela on utilise la fonction `\textls` définie dans `microtype`[20]. Si le résultat est correct pour le titre court, les lettres du long se chevauchent un peu et on frise l'illisibilité.
   En fait cette fonction demande, en premier paramètre, l'accroissement (positif ou négatif) à donner ; ici on a trouvé les valeurs 36 et –45 e(n centièmes de points) en faisant plusieurs essais. Il faudrait donner à la macros la justification et lui laisser choisir ce crénage en divisant l'espece en trop (ou en moins) par le nombre de signes. Mais là on se heurte à la très grosse difficulté de faire ce comptage (le package xstring n'y arrive pas bien, alors…)…
4. Composition à la Leicester : on choisit le corps du titre pour qu'il ait exactement la largeur de la colonne. Dans les deux cas, on reste lisible. Mais pour des textes de longueurs différentes, on aurait des changements de corps très visibles.

**Exemple 2 : pavés de couverture**

C'est surtout pour ceci que la méthode de Leicester trouve son plein emploi : il s'agit de créer des pavés de lettres pour former par exemple un couverture de livre[21]

Le principe est simplement d'écrire le titre ligne par ligne en annonçant la justification. Par exemple, les commandes

```
\centerline{\parbox{10cm}{%
\Leicester{FONTSPEC}{10cm}\\
\Leicester{Pour les nuls}{10cm}}}
```

impriment


20. En fait on en utilise une autre définition car le triplet `fontspec, microtype` et `textls` ne feraient pas toujours bon ménage. Voir https://tex.stackexchange.com/questions/475553/microtype-texls-has-no-effect.
21. C'est la méthode que j'ai employée pour la couverture du *Cahier GUTenberg* 20.

L'exemple 2 (figure 14) est inspiré d'une couverture gravée sur bois par Louis Jou [22] dans les années 1950. On la compose ici en utilisant la fonte *Infini-romain* (voir note 5) qui est très riche en ligatures.
Avec les macros définies plus haut, le programme est le suivant :

```
\begin{center}
\fontspec[RawFeature={+liga,+dlig}]{Infini-romain.otf}
\setlength{\justif}{8cm}

\Leicester{RONSARD}{\the\justif}

\Leicester{SONNETS}{\the\justif}

\Leicester{POUR HELENE}{\the\justif}

\Leicester{LES LIVRES DE}{\the\justif}

\Leicester{LOUIS JOU}{\the\justif}
\end{center}
```

et la sortie en figure 14.

22. Loui Jou a été une sorte de Bob William français des années 1930 et suiv. ;voir ma note parue dans *Histoire de l'écriture typographique, le* XX^e *siècle* et qui se trouve ici : http://jacques-andre.fr/japublis/XX-jou.pdf.


FIGURE 14 – Couverture inspirée d'une de Louis Jou, composée ici avec la fonte *Infini* et la méthode de Leicester.
)

# 5 Annexe : listing complet des programmes mentionnés

Les programmes étudiés dans cette note sont listés ci-dessous.
Ils sont accessibles par téléchargement depuis mon site http:jacques-andre.fr/fontex/Util+OT.zip.

## Inventaire+casseau.ltx

```
%!TEX encoding = UTF-8 Unicode
% !TEX TS-program = LuaLaTeX
\documentclass[10pt]{article}
% inventaires des caractères d'une fonte casseau
% Version J.A./fontex 13/2/2022
\usepackage[a4paper,textwidth=16cm,%
textheight=23.4cm,heightrounded]{geometry}
\usepackage{fontspec}
\usepackage{ifthen,xifthen,calc}
\usepackage{multido}
\usepackage{multicol}
\usepackage{fmtcount}
\newcounter{ctr}
\setmonofont{lmmonolt10-regular.otf}
\newcommand{\mafonte}{Infini-picto}%{<mafonte>}
%%%%%%%%% Remplacer <mafonte> pari path+ nom de la fonte à examiner
\begin{document}\parindent0pt
{\Large Glyphes  du casseau \textbf{\mafonte}}\\
Notation: \texttt{code hexa = car Unicode = glyphe de \mafonte}\par
\fontsize{12}{36}\selectfont
\newfontfamily\cettefonte{\mafonte}
\multido{\ia=0+1}{255}{\setcounter{ctr}{\ia}%
\cettefonte
 \ifthenelse{
   \directlua{
     local f = font.getfont(font.current())
     local chars = f.characters
     if chars[\ia] then
       tex.print('\string\\boolean{false}') %
     else
       tex.print('\string\\boolean{true}')
     end
   }}{}{\makebox[4cm][l]{\texttt{\HEXADecimal{ctr}\mbox{ = \char\ia\ = }}%
   {\cettefonte\parbox[b][15pt]{50pt}{\strut\Huge\char\ia}}}\ }}\par
```

```
   \vfill
 \texttt{\small[Liste établie par Inventaire+casseau.ltx --  \today]}
   \end{document}
```

### Inventaire+OpenType.ltx

Inventaire des glyphes d'une fonte OpenType (voir page 8).
Téléchargeable depuis http:jacques-andre.fr/fontex/Util+OT.zip.

```
\documentclass{article}%Inventaire+OpenType.ltx
%Inventaire des glyphes d'une fonte
\newcommand\myfilename{\detokenize{infini-romain}}% <=<=<= fonte à examiner
\newcommand\myext{.otf} %<=<=<= et son extension
\newcommand\myfont{infini-romain\myext} % et encore ici...
\usepackage{fmtcount}
\usepackage{xcolor}
\usepackage{fontspec}
\setmainfont{Noto Serif}
\newfontfamily\forn{\myfilename}[
   Extension=\myext,
  UprightFont=*,
]

\input binhex
\usepackage{luacode}
\usepackage{multicol}
\setlength{\columnsep}{0.3cm} \setlength{\columnseprule}{1pt}

\begin{document}

\centerline{\Large Inventaire des glyphes de \myfont -- \today} \bigskip

\begin{luacode}
           myfont         = '\myfont'
\end{luacode}

\begin{multicols}{2}\noindent
\begin{luacode*}
```

```
local f = fontloader.open(myfont)
local glyphs = {}
for i = 0, f.glyphmax - 1 do
   local g = f.glyphs[i]
   if g then
       table.insert(glyphs, {name = g.name, unicode = g.unicode})
   end
end
table.sort(glyphs, function (a,b) return (a.unicode < b.unicode) end)
for i = 1, #glyphs do
   if (glyphs[i].unicode > 0) then
   tex.print("\\hex{")
   tex.sprint(glyphs[i].unicode  )
  tex.print("}: ")
   tex.sprint("{\\makebox[4em]{\\Huge\\color{blue}\\forn\\char" .. glyphs[i].unicode ..
   tex.print("\\ {=")
   tex.print(-2, glyphs[i].name)
   tex.sprint('}\\\\\\\\\')
   end
end
fontloader.close(f)
\end{luacode*}
\end{multicols}
\centerline{Fin}
\end{document}
```

## analOT+subs.pl

```
# # analOT+subs.pl = Analyse des substitutions  simples d'une fonte OpenType.sfd
# programme écrit par Jacques André sur base de Daniel Flipo
# Version du 5 avril 2022
# À partir d'un fichier .sfd (fourni par Fontforge), ce script fournit la liste
# des glyphes associés à une feature donnée ainsi que les glyphes résultants
# Sortie : feature, <nom> et glyphe initial, glyphe résultant <nom>
# sous forme d'un programme TeX pour être édité ensuite par analOT+print.tex
#
use POSIX;
use strict;
use warnings;

my $lafonte;#fonte étudiée
my $mafonte;#fonte en bdc
my $lafeat; #feature étudiée
my $name = "";
my $dec = "";
my $flag = 0;

use feature "unicode_strings";
BEGIN { binmode *STDOUT, ':encoding(UTF-8)'; binmode *STDERR, ':encoding(UTF-
8)';
binmode STDIN, ":encoding(UTF-8)";};
#binmode STDOUT, ":encoding(utf8)";
my $XDATE=`date`; substr($XDATE,-1,1)=" "; # suprimer end of line \n


print "Nom de la fonte (sans extension) ?";
chomp($lafonte =<STDIN>);
print "feature a etudier ?"; ################ pourquoi accents passent pas ???
chomp($lafeat =<STDIN>);
open (IN,"<:encoding(utf8)","$lafonte".".sfd")|| die "pas ouvert:"."$lafonte".".sfd";
open (OUT, ">:encoding(utf8)","$lafonte"."+anal.tex") ||
die  "$lafonte"."+anal.tex  pas ouvrable";
#################### on y va
$mafonte=lc $lafonte;
print OUT ("\\def\\JAN\{$mafonte\}\n");
print OUT ("\\def\\JAD\{$XDATE\}\n");
print OUT ("\\def\\JAF\{$lafeat\}\n");
print OUT ("\\JAGO\n");
###

while (<IN>)
```

```
  {if (/^StartChar:\s+(.+)$/)
      {$name = "$1"; $flag= 1;}
   if ($flag= 1 && /^Encoding:\s+(\d+)\s/)
      {$dec="$1";}
   if ($flag= 1 && /^Substitution2:\s+"(.+)$/)
      {my $lig = "$1";
       $lig =~ s/ .*"//; #ici tester si il y a lig dans $1
#trouver feature et car initiaux
my @ventila=split(/\s+/,$lig);
my $feat = $ventila[0];
if ($feat =~ m/$lafeat/){#
splice (@ventila, 0,1);
print OUT ($feat);my $i;
print OUT  ("<\\verb+$name+> "); ## imprime nom pour préciser
print OUT  ( " \\parnom{", $name,"}"); # et son glyphe
print OUT (" \\FL ");
foreach $i (@ventila) {print OUT (" \\parnom{",$i,"}");
    print OUT (" <\\verb+$i+>");};
print OUT  ("\\\\\n");
      }}#fin lig #fin boucle
   if (/^EndChar/)
      {$flag= 0;}
  }

## fin analfonte.pl

#################### on finit
close(IN);
close(OUT);

print "c'est fini : sortie dans $lafonte+anal.tex\n";

### Local Variables:
### mode: perl
### End:
```

## analOT+lig.pl

```
## analOT+lig.pl =  Analyse des ligatures  simples d'une fonte OpenType.sfd
# programme écrit par Jacques André sur base de Daniel Flipo + 05/04/2022
# À partir d'un fichier .sfd (Fontforge), ce script fournit la liste
# des glyphes associés à une feature  DE LIGATURE donnée
# ainsi que les glyphes résultants
# Sortie : feature, glyphes initiaux, glyphe résultant
# sous forme d'un programme TeX pour être édité ensuite.
#
use POSIX;
use strict;
use warnings;

my $lafonte;#fonte étudiée
my $mafonte;#fonte en bdc
my $lafeat; #eature étudiée
my $name = "";
my $dec = "";
my $flag = 0;
my $i;

use feature "unicode_strings";
BEGIN { binmode *STDOUT, ':encoding(UTF-8)'; binmode *STDERR, ':encoding(UTF-
8)';
binmode STDIN, ":encoding(UTF-8)";};
#binmode STDOUT, ":encoding(utf8)";
my $XDATE=`date`; substr($XDATE,-1,1)=" "; # suprimer end of line \n


print "Nom de la fonte (sans extension) ?";
chomp($lafonte =<STDIN>);
print "feature (dlig, frac...)  a etudier ?"; ################ pourquoi accents passent pas ???
chomp($lafeat =<STDIN>);
open (IN,"<:encoding(utf8)","$lafonte".".sfd")|| die "pas ouvert:"."$lafonte".".sfd";
open (OUT, ">:encoding(utf8)","$lafonte"."+anal.tex") || die  "$lafonte"."+anal.tex  pas ouvrabl
#################### on y va
$mafonte= lc $lafonte;
print OUT ("\\def\\JAN\{$mafonte\}\n");
print OUT ("\\def\\JAD\{$XDATE\}\n");
print OUT ("\\def\\JAF\{$lafeat\}\n");
print OUT ("\\JAGO\n");
###

while (<IN>)
```

```
  {if (/^StartChar:\s+(.+)$/)
      {$name = "$1"; $flag= 1;}
   if ($flag= 1 && /^Encoding:\s+(\d+)\s/)
      {$dec="$1";}
   if ($flag= 1 && /^Ligature2:\s+"(.+)$/)
      {#print("lu= ", $_ , "\n");
 my $lig = "$1";
       $lig =~ s/ .*"//; #ici tester si il y a lig dans $1
my @ventila=split(/\s+/,$lig);
#print("ventila= ",@ventila,"\n");
my $feat = $ventila[0];
if ($feat =~ m/$lafeat/){#
splice (@ventila, 0,1);
print OUT ($feat);
foreach $i (@ventila) {print OUT (" \\ \\parnom{",$i,"}")};
print OUT (" \\FL ");print OUT ( " \\ \\parnom{", $name,"} <U+");
                     printf OUT ( "% 4X",  $dec);
## en cas de ligature OK; sinon faire fonthex{name);
print OUT (">");
print OUT ("\\\\\n");
      }}#fin lig #fin boucle
   if (/^EndChar/)
      {$flag= 0;}
  }#fin while


#################### on finit
close(IN);
close(OUT);

print "c'est fini : sortie dans $lafonte+anal.tex\n";


### Local Variables:
### mode: perl
### End:
```

## analOT+print.ltx

```
\documentclass{scrartcl}% analOT+print.tex. %% version du 5/4/2022
%%Lecture des résultats d'analyse d'une fonte OpenType produits par
%%le programme PERL analOT+subs/lig.pl (J.A.)
\newcommand\myfont{GaramondPremrPro-italic}% <=<=<=<= Mettre ici nom fonte, sans extension.
\usepackage{fourier-orns}% contient def de \bomb utilisée en cas d'erreur
\usepackage{comment}%pour debugging
%
% inspiré de < https://tex.stackexchange.com/questions/502907/
%pour les appels par nom de glyphe nécessaires pour la commande \parnom
%\usepackage{comment}
\usepackage{xstring}% pour \JAERR
\newcommand{\JAERR}[1]{\StrLen{#1}[\myl]%
\IfBeginWith{#1}{uni}{\ifthenelse{\equal{\myl}{7}}%uniXXXX
{\StrRight{#1}{4}[\mynum]\symbol{\string"\mynum}\ \bomb}{\bomb}}{\bomb}}%
\usepackage{luaotfload,luacode}
\usepackage{newunicodechar}
\begin{luacode}
  documentdata = documentdata or { }
  documentdata.fontchar = function (chr)
  local xxx=chr
    local chr = luaotfload.aux.slot_of_name(font.current(), chr, false)
    if chr and type(chr) == "number" then
      tex.sprint(string.format([[\char"%X]], chr))
      else  tex.sprint('\\JAERR{') tex.sprint(string.format(xxx)) tex.sprint('}')
    end
  end
\end{luacode}
\usepackage{ifthen}

\newcommand{\parnom}[1]{\edef\mynom{#1}\directlua{documentdata.fontchar "#1"}}
\usepackage{fontspec}
\usepackage{multicol}
\usepackage{xspace}

%%%%%
\setmonofont[Scale =.95]{Noto Sans Mono}
% le fichier \myfont+anal.tex définit JAN, JAD, JAF
\newcommand{\FL}{$\Rightarrow$\xspace}%flèche
   \newcommand{\JAGO}%
{\setmainfont{\JAN}%\JAN=\myfont en bdc comme nom.lua
\addfontfeature{Numbers={Lining,Monospaced}}%
Étude de la fonte \textbf{\myfont}\\
{\small faite le «\JAD» à partir du fichier \texttt{\myfont.sfd}}\\
Liste des glyphes substitués par la \textit{feature} \\
```

```
 {\huge  \texttt{\JAF} } de {\huge \textbf{\myfont}}
\medskip
\begin{multicols}{2}\noindent}%fin JAGO
\newcommand{\JAFIN}{\end{multicols}}
%
\begin{document}\parindent0pt
\input{\myfont+anal}% fichier de sortie du programme perl
 \JAFIN
 \end{document}
```

## psbbox.lua

```
--- psbbox.lua --- from https://tex.stackexchange.com/questions/522748/
  packagedata                  = packagedata or { }
  packagedata.psbbox     = { }
  local psbbox           = packagedata.psbbox

  local utfbyte                = utf.byte
  local texsprint              = tex.sprint

  local get_psbbox = function (id, char)
    local tfmdata = font.getfont (id)

    if not (tfmdata and tfmdata.shared) then
      return 0, 0
    end

    local descriptions = tfmdata.shared.rawdata.descriptions
    local glyphdata    = descriptions [char]
    if not glyphdata then
      --- font lacks the glyph
      return 0, 0, 0, 0
    end

    local boundingbox   = glyphdata.boundingbox
    local wd            = boundingbox [3] or glyphdata.width
    local llx          = boundingbox [1] or 0
    local lly          = boundingbox [2] or 0
    local urx          = boundingbox [3] or 0
    local ury          = boundingbox [4] or 0
    local factor        =  tfmdata.size / tfmdata.units_per_em

    --- If you would like to print the values to terminal
    inspect (glyphdata)
return llx * factor, lly * factor, urx * factor, ury * factor
  end

  local psbbox = function (id, char, side)
    char = utfbyte (char)
local llx, lly, urx, ury = get_psbbox (id, char)
```

```
    if side == "llx" then
      texsprint (tostring (llx), "sp")
    elseif side == "lly" then
      texsprint (tostring (lly), "sp")
    elseif side == "urx" then
      texsprint (tostring (urx), "sp")
    elseif side == "ury" then
      texsprint (tostring (ury), "sp")
    end
  end

  packagedata.psbbox.left  = function (char)
    return psbbox (font.current (), char, "llx")
  end

  packagedata.psbbox.right = function (char)
    return psbbox (font.current (), char, "urx")
  end

  packagedata.psbbox.top = function (char)
    return psbbox (font.current (), char, "ury")
  end

  packagedata.psbbox.bottom = function (char)
    return psbbox (font.current (), char, "lly")
  end
```

## metrique.tex

NB Ce programme (voir section 4.1) utilise la procédure `psbbox.lua` listée page 45.

```
\documentclass{article}%Inventaire des glyphes d'une fonte
\newcommand\myfilename{\detokenize{infini-romain}}% <=<=<= fonte à examiner
\newcommand\myext{.otf} %<=<=<= et son extension
\newcommand\myfont{infini-romain\myext} % et encore ici...
\usepackage{fmtcount}
\usepackage{xcolor}
\usepackage{fontspec}
\setmainfont{Noto Serif}
\newfontfamily\forn{\myfilename}[
   Extension=\myext,
  UprightFont=*,
]

\input binhex
\usepackage{luacode}
\usepackage{multicol}
\setlength{\columnsep}{0.3cm} \setlength{\columnseprule}{1pt}

\begin{document}

\centerline{\Large Inventaire des glyphes de \myfont -- \today} \bigskip

\begin{luacode}
           myfont         = '\myfont'
\end{luacode}

\begin{multicols}{2}\noindent
\begin{luacode*}
local f = fontloader.open(myfont)
local glyphs = {}
for i = 0, f.glyphmax - 1 do
   local g = f.glyphs[i]
   if g then
       table.insert(glyphs, {name = g.name, unicode = g.unicode})
   end
end
table.sort(glyphs, function (a,b) return (a.unicode < b.unicode) end)
for i = 1, #glyphs do
   if (glyphs[i].unicode > 0) then
   tex.print("\\hex{")
   tex.sprint(glyphs[i].unicode  )
```

```
  tex.print("}: ")
   tex.sprint("{\\makebox[4em]{\\Huge\\color{blue}\\forn\\char" .. glyphs[i].unicode .. "}}");
   tex.print("\\ {=")
   tex.print(-2, glyphs[i].name)
   tex.sprint('}\\\\\\\\\')
   end
end
fontloader.close(f)
\end{luacode*}
\end{multicols}

\centerline{Fin}
\end{document}
```

```
%!TEX encoding = UTF-8 Unicode
% Metrique.tex = montre métrique d'un caractère (formats plomb, tfm, afm)
\documentclass[english,french,12pt]{article}
\usepackage{babel}
\usepackage{fontspec}
%
%%%%%%%%%%%%%%%  materiel pour macro Metrique
%%%%%%%% Pour dessin bbox
\newlength{\tmpl}\newlength{\tmplg}% tmp pour length
\newcommand{\putleft}[3]{\settowidth{\tmplg}{#3}%
\setlength{\tmplg}{#1-\the\tmplg}\put(\the\tmplg,#2){#3}}
\newcommand{\putmilieu}[3]{\settowidth{\tmplg}{#3}%
\setlength{\tmplg}{#1-(\the\tmplg/2)}\put(\the\tmplg,#2){#3}}
%%% pour calculs bbox d'après https://tex.stackexchange.com/questions/522748/
%%%%%%% À utiliser avec fichier psbbox.lua dans même répertoire
\directlua {require "psbbox"}
  \def \lsidebearing #1{%
    \the\dimexpr\directlua {packagedata.psbbox.left[[#1]]}\relax%
  }
  \def \rsidebearing #1{%
    \the\dimexpr\directlua {packagedata.psbbox.right[[#1]]}\relax%
  }
  \def \tsidebearing #1{%
    \the\dimexpr\directlua {packagedata.psbbox.top[[#1]]}\relax%
  }
  \def \bsidebearing #1{%
    \the\dimexpr\directlua {packagedata.psbbox.bottom[[#1]]}\relax%
  }
 %
\def\bla{}%
\def\ifpasvide#1{%macro déf par JC Charpentier%
%teste si param de macro  est vide
\ifx\relax#1\relax%
\ifx\bla#1\bla%
%vide%
\else%
%pas vide%
\fi%
```

```
\else}% ne pas oublier le /fi final après l'appel


\newlength{\llx}\newlength{\lly}\newlength{\urx}\newlength{\ury}
\newcommand{\bbox}[1]{%%%%%%%%%%%%%%%%%%%%%%%%%%%%% bbox
\global\llx=\lsidebearing{#1}\relax%
\global\urx=\rsidebearing{#1}\relax%
\global\lly=\bsidebearing{#1}\relax%
\global\ury=\tsidebearing{#1}\relax}%
\setlength{\fboxsep}{0em}
\newlength{\mm}\newlength{\ww}%
\newlength{\deltax}\newlength{\deltay}%
\newcommand{\metrique}[6]{%%%%%%%%%%%%%%%%%%%%%%%%% metrique
% paramètres: # 1=carac #2=couleur car; 3=coul corps ;
%4 = coul fbox ; 5 = coul bbox; 6 = coul ligne base
%NB utilise épaisseur des traits
 \edef\moncar{#1}%
\settowidth{\ww}{#1}%
\bbox{\moncar}%laisser ici même si #5=vide, pour avoir llx & co
\begin{picture}(\the\ww,1em)%
\put(0,0){\ifthenelse{\equal{#4}{}}{{\color{#2}#1}}% print car
{\color{#4}\begin{picture}(0,0)\put(0,0){\color{#2}#1}%
\settoheight{\mm}{#1}\settodepth{\pp}{#1}%
%cadre pour remplacer fbox pour garder linethickness
\setlength{\mm}{\the\mm+\the\pp}%
\put(0,-\the\pp){\line(0,1){\the\mm}}%
\put(\the\ww,-\the\pp){\line(0,1){\the\mm}}%
\put(0,-\the\pp){\line(1,0){\the\ww}}%
\put(0,-\the\pp+\the\mm){\line(1,0){\the\ww}}%
\end{picture}}}%
%
\ifpasvide{#3}%corps tradi
\color{#3}%
\settoheight{\mm}{m}\setlength{\mm}{\the\mm/2}%
\put(0,\the\mm+.5em){\line(1,0){\the\ww}}%
\put(0,\the\mm-.5em){\line(1,0){\the\ww}}%
\put(0,\the\mm-.5em){\line(0,1){1em}}%
\put(\the\ww,\the\mm-.5em){\line(0,1){1em}}%
```

```
\fi%%
\ifpasvide{#5}%
\color{#5}%bbox
\setlength{\deltax}{(\the\urx-\the\llx)}\setlength{\deltay}%
{(\the\ury-\the\lly)}%
\put(\the\llx,\the\lly){\line(1,0){\the\deltax}}%()
\put(\the\llx,\the\lly){\line(0,1){\the\deltay}}%
\put(\the\urx,\the\ury){\line(-1,0){\the\deltax}}%()
\put(\the\urx,\the\ury){\line(0,-1){\the\deltay}}%
\put(0,0){\circle*{.05em}}\fi%
%
\ifpasvide{#6}%ligne de base et point 0,0
\color{#6}%
\put(0,0){\circle*{.05em}}%
\multiput(0,0)(\the\ww/10,0){10}{\line(1,0){\the\ww/20}}%dashed baseline
\fi%
\end{picture}}%fin metrique

%%%%%%%%%%%%%%%%%%%%
 \setmainfont{Erewhon}[
           Extension=.otf,
           UprightFont=*-Regular,
           ItalicFont=*-Italic]
\usepackage{color}
\usepackage{picture,ifthen,calc}
\usepackage{relsize}

\begin{document}
\newlength{\myl}\setlength{\myl}{3pt}
\definecolor{bleu}{cmyk}{0.5,0,0,0}%

\linethickness{2pt}%
\fontsize{200}{100}\selectfont%\
\metrique{m}{yellow}{blue}{red}{}{green}\ \metrique{g}{yellow}{}{red}{black}{green
\itshape\ \metrique{f}{yellow}{blue}{}{black}{green}\bigskip

\begin{figure}[b]%
\setlength{\fboxsep}{0em}\linethickness{1pt}%
```

```
\centerline{\fontsize{60}{60}\selectfont\itshape%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{black}{}{}{}%
\metrique{f}{bleu}{black}{}{}{}%
\metrique{m}{bleu}{black}{}{}{}%
\metrique{\_}{bleu}{black}{}{}{}\\[.3em]
\normalsize\color{black}Plomb\end{center}\end{minipage}\hspace*{1cm}%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{}{blue}{}{blue}%
\metrique{f}{bleu}{}{blue}{}{blue}%
\metrique{m}{bleu}{}{blue}{}{blue}%
\metrique{\_}{bleu}{}{blue}{}{blue}\\[.3em]
\normalsize\color{blue}\TeX\end{center}\end{minipage}\hspace*{1cm}%
\begin{minipage}{2em}\begin{center}
\metrique{a}{bleu}{}{}{red}{red}%
\metrique{f}{bleu}{}{}{red}{red}%
\metrique{m}{bleu}{}{}{red}{red}%
\metrique{\Uchar"5F}{bleu}{}{}{red}{red}\\[.3em]
\normalsize\color{red}PostScript \& OpenType\end{center}\end{minipage}
}
\caption{Trois méthodes de composition.}\label{afm}
\end{figure}
%%%%
\end{document}
```