LA FONTE DU CE NUMÉRO : *INFINI*

Le caractère *Infini* a été créé par Sandrine Nugue dans le cadre d'une commande du Centre national des Arts plastiques. Et c'est précisément en raison de cette commande publique que nous avons décidé de l'utiliser pour ce numéro de la *Lettre* GUTenberg. Utiliser nos outils habituels de mise en page avec cette fonte nous a semblé intéressant : comment allions-nous accéder aux différents glyphes qui la composent sans bénéficier d'un package qui simplifie ce travail ? C'est ce que nous verrons en page 45 et suivantes. Bien que nous y fassions surtout une introduction au format de fontes *OpenType*, nous allons aussi y utiliser quelques mots techniques de typographie. Pour les lecteurs qui ne connaîtraient pas leur sens, nous recommandons de consulter Wikipedia. Cette encyclopédie est en effet très sérieuse et les articles liés à la typographie sont en général à la fois pédagogiques et fouillés. Mais nous allons nous efforcer de faire ici des rappels historico-techniques, quitte parfois à noyer le poisson dans l'eau.

Par ailleurs, décider d'utiliser cette fonte, c'est la mettre au banc d'essai : pour composer la *Lettre* GUTenberg, nous faisons habituellement appel à une large gamme de caractères : avec ou sans empattements, à chasse fixe, dédiés aux mathématiques etc. ; comment *Infini* allait-elle s'insérer dans notre processus de fabrication ? Et quelles solutions allions-nous trouver pour répondre aux éventuels problèmes techniques rencontrés ? Enfin, utiliser ici cette fonte, c'est la laisser se déployer sur de nombreuses pages, en apprécier le niveau de gris, les différentes graisses et corps. Cela permet de s'en faire une opinion fondée sur une vraie lecture et non sur la simple contemplation d'un exemple.

Description

Infini appartient, dans la classification des caractères dite Vox-Atypi, à la famille des incises dont les capitales possèdent des fûts évasés qui rappellent l'écriture lapidaire (celle des inscriptions antiques gravées au ciseau dans la pierre). Les larges contreformes [66] aèrent le texte, et cet effet est renforcé par le fait que les panses du B, du P et du R ne rejoignent le fût qu'à ses extrémités. Un effet similaire est obtenu avec les diagonales du K, totalement indépendantes de son fût (figure 6 page suivante). Le L et le F présentent des traverses évasées, avec un angle plus prononcé que ceux des fûts, ce qui évoque les empattements absents. La queue du Q est à cet

66. Les contreformes, originellement créées par des contre-poinçons, correspondent aux creux de certaines lettres, souvent fermés comme le *e* ou le *o*, parfois ouverts comme le haut du *f*. Nous renvoyons les lecteurs au bel ouvrage de Fred Smeijers [27] et au bref compte-rendu que nous en avons fait dans la *Lettre* précédente [28].

FIGURE 6 – Le côté lapidaire d'*Infini* est atténué par un tracé aéré.

égard plus significative encore. Il est facile d'imaginer les capitales d'*Infini* gravées à l'aide d'un ciseau ou d'un calame, et un texte entièrement composé avec les capitales d'*Infini* évoquera celles, monumentales, des bâtiments romains — à ceci près que les empattements ne seront que suggérés [67].

Mais voilà, notre écriture est bicamérale [68] : comment les bas-de-casse, qui ont une origine manuscrite, allaient se confronter à l'esthétique des capitales d'*Infini*, dont la rigueur évoque nettement l'écriture lapidaire ? Quelle apparence allait leur donner leur créatrice ? Sandrine Nugue a trouvé des solutions pertinentes et remarquables : les bas-de-casse bénéficient également d'amples contreformes, grâce auxquelles une page composée en *Infini* reste lumineuse, malgré un niveau de gris assez dense dû à un trait épais. Les fûts sont subtilement évasés, comme ceux des capitales, et de très discrets empattements, tels ceux du « p » et du « r » (figure 7 page suivante), fluidifient la lecture. L'esthétique homogène des capitales et bas-de-casse rend la fonte *Infini* appropriée pour le texte courant [69]. Une version grasse est fournie, et nous ne pouvons que regretter l'absence de son pendant maigre, qui nous aurait comblés. Nous espérons que l'évolution de cette fonte continuera et qu'elle proposera un jour support mathématique et caractères grecs.

Les italiques sont également réussis. L'évasement des fûts du romain créant déjà des obliques, ceux-ci ont été accentués pour l'italique, et l'ensemble est harmonieux. On appréciera particulièrement le « *a* » de l'italique (voir figure 14 page 54), qui évoque celui de l'écriture semi-onciale et montre à quel point la créatrice de ces caractères avait à cœur de revisiter l'histoire de l'écriture.

Le signe distinctif d'*Infini*, qui nous permet de reconnaître cette police au premier

67. On trouvera, sous la plume de Sébastien Morlighem, une page entièrement composée en capitales : la page 7 de la plaquette de présentation d'*Infini* [1].

68. En typographie, ce mot signifie en gros « formé de majuscules et minuscules ». Il a été mis à la mode par le codage Unicode pour pouvoir spécifier les écritures qui permettent de passer des unes aux autres.

69. Nous déplorons le caractère anguleux de l'arobase « @ » et de l'esperluette « & », qui hélas rompt la fluidité de la lecture et jure avec les autres caractères d'*Infini*.

FIGURE 7 – Détail des fûts du *r* et du *p* de *Infini*-romain.

FIGURE 8 – Utilisation de ligatures de *Infini* pour un titre de livre inspiré d'une couverture de Louis Jou.

regard, ce sont évidemment ses ligatures capitales. Elles sont très nombreuses et peu communes, parfois subversives quand leur créatrice place le « I » horizontalement. Elles nous renvoient aux inscriptions anciennes (monuments, pierres tombales, etc.) qui employaient force abréviations. Certaines de ces ligatures sont très réussies, comme « ME » ou « ASSE [70] » ; d'autres moins : nous avons toujours tendance à lire « O J » la ligature « OJ » alors qu'elle est formée par les lettres « O » et « U » (voir par exemple en figure 8). Mais nous saluons cette expérimentation ludique en utilisant ces ligatures pour les titres de chacun des articles de cette *Lettre*. La figure 8 donne un autre exemple d'emploi de ces ligatures pour des couvertures de livres, dans l'esprit de Louis Jou [29] qui imitait ainsi des gravures lapidaires romaines.

Enfin, une fonte casseau est proposée, dans un style complètement différent : Sandrine Nugue a créé 26 capitales qui sont autant de pictogrammes. Ils ne sont toutefois pas sans nous faire penser à un alphabet de Tory au XVI^e siècle (figure 9 page ci-contre). Ainsi, la lettre B forme les voiles d'un bateau tandis que le G se révèle être le corps d'une guitare, dont le manche horizontal est la traverse de la lettre, exagérément allongée ! L'étude cette fonte (voir page 73) montre qu'en fait le *glyphname* d'une lettre commence par la lettre et est le nom de l'objet représenté par son glyphe. Ainsi a-t-on *arrosoir* = A = A, *bateau* = B = B, *crabe* = C = C… Bel exemple de système

70. Nul doute que cette dernière est d'ores et déjà adoptée par les *supporters* de l'ancien club de Dominique Rocheteau.

FIGURE 9 – Comparaison de l'*Alphabet fantastique* de Tory, 1520 [30] et de *Infini-picto*

« acrophonique » comme les abécédaires de nos écoles maternelles où, comme ici, M représente une montagne et S un serpent. Nous laissons cette expérimentation à l'appréciation du lecteur et ne l'utilisons que pour les capitales initiales des titres de section de notre dernière page [71].

Composition de la *Lettre*

Nous sommes habitués à utiliser des polices qui proposent des caractères avec et sans empattements. Il est fréquent que nous changions de famille, y compris au sein d'une même phrase, par exemple pour mettre en valeur le nom d'un package. Cet usage est évidemment impossible avec *Infini* et nous nous en sommes accommodés ; nous avons parfois utilisé ses petites capitales à la place. En revanche, nous avons souffert de l'absence d'une police à chasse fixe. Après divers essais, nous avons choisi d'utiliser la police *Noto Sans Mono* dans sa graisse medium, dont la juxtaposition avec *Infini* ne nous a pas semblé choquante (signalons que cette fonte fait partie de TEX Live [72]) ; on verra en page 53 comment faire cohabiter ces fontes.

Comment analyser et utiliser les substitutions d'une fonte *OpenType* ?

Infini est une fonte *OpenType* ; c'est pour nous l'occasion de montrer (rapidement [73]) comment connaître les possibilités d'une telle fonte et donc de savoir comment l'utiliser avec le package fontspec et le moteur LuaLATEX [74]. Nous nous plaçons ici avec

71. L'attentif lectorat y remarquera que nous avons gardé notre feuille aldine habituelle, qui nous est fournie par le package fourier-orns, de Michel Bovani ; il va sans dire qu'elle ne fait pas partie d'*Infini*. Mais nous serions très intéressés par des feuilles aldines et des manicules dessinées par Madame Nugue pour *Infini* !

72. Le lectorat anglophone aura plaisir à lire la communication de Steven Matteson, l'auteur de Noto, prononcée lors de la conférence TUG (*TEX User Group* [anglais] : groupe (international) d'utilisateurs de TEX) 2020 et parue dans le n° 41:2 du *TUGboat*.

73. Le meilleur ouvrage de synthèse sur ces divers formats de fonte (TEX, *TrueType*, *Type1*, *OpenType*, etc.) reste le livre *Fontes & Codages* de Yannis Haralambous [9].

74. On parlera ici du moteur LuaLATEX [2] pour parler du moteur LuaTEX avec le format LATEX chargé.

l'optique d'un utilisateur qui veut principalement savoir quels sont les caractères d'une fonte et comment les utiliser.

D'abord quelques rappels historiques…

Que contient une fonte ?

Du temps du plomb, la composition consistait à placer côte à côte des types [75] dans un composteur, boîte où l'on met ce qui fera une ligne à imprimer. Les compositeurs n'avaient aucun besoin de connaître la valeur de la chasse des caractères (mais savaient évaluer à l'œil nu quelles espaces rajouter et comment couper les mots pour justifier une ligne). Les fontes étaient donc des ensembles de caractères en métal, sans autre information [76]. Mais la richesse du fond typographique a fait que les « plombs » servent encore de modèles aux fontes numériques. Aussi montrerons-nous souvent des caractères anciens (notamment dans leur contexte métrique).

C'est avec la composition (photo-)mécanique, et notamment la Monotype (fin du XIX^e siècle) puis la photocomposition (milieu du XX^e siècle), que sont apparues les tables de chasses pour que « la machine » puisse calculer l'occupation des caractères dans les lignes. Probablement parmi les toutes premières fontes numériques [77], celles de Hershey, vers 1967, dessinaient des caractères avec des traits sur des tables traçantes. Chaque signe était défini par la succession des coordonnées des extrémités des « vecteurs » et par la chasse du caractère ce qui permettait de calculer le déplacement à donner à la plume pour dessiner le caractère suivant. C'est le même principe que l'on va retrouver avec les fontes pour les images tramées des imprimantes graphiques (cette fois, on procède par remplissage de surfaces définies par leurs contours à l'aide, par exemple, de courbes de Bézier). Ce sont notamment, de 1975 et 1980, les fontes de URW, TEX, `troff` et `PostScript`.

Les imprimantes sont des automates qui reçoivent des instructions de dessin et qui noircissent du papier, pixel par pixel, en fonction de ces instructions. Par exemple, pour tracer les lettres « *afm* » de la figure 10 page ci-contre, une imprimante recevra (dans un langage comme DVI [78] en TEX, ou `PostScript`, ici on utilise quelque chose de simpliste) :

Exemple 19

```
fonte= EBGaramond-italic
corps courant=...
point courant: x=... y=...
```

75. Ce sont les caractères en plomb, des parallélépipèdes dont, vu d'en haut, la largeur s'appelle la *chasse*, la hauteur le *corps* et dont la troisième dimension est la hauteur en papier, une constante. Voir figure 10 page suivante.

76. En fait si : quand un imprimeur achetait « une fonte », la fonderie lui livrait des petits paquets de types et un bordereau donnant la liste des types présents, par exemple 200 « a », 150 « b », etc. Cette liste s'appelait la *police* de la fonte (ce mot police est celui que l'on retrouve dans police d'assurance). C'est probablement au moment de la photocomposition, vers 1970, où ce concept de nombre de types n'avait plus de sens, qu'on a appelé « police » non plus la liste des caractères, mais l'ensembles des images de caractères. C'est comme si on disait que c'est le mot chien qui aboie…Mais on l'utilise encore pour « fonte » : nous garderons ici le mot *fonte* pour désigner un ensemble de caractères avec les mêmes propriétés, par exemple du « Garamond dessiné par Untel, italique » — au XIX^e siècle on aurait dit aussi en corps 16, etc. » — et l'expression *famille de fontes* pour désigner l'ensemble des fontes correspondant à la seule première propriété (par exemple au Garamond de Untel).

77. On trouvera dans [31, chap. 5] plus de détails, et la bibliographie, sur cette histoire des fontes numériques.

78. *DeVice-Independent* [anglais] : indépendant du type de périphérique.

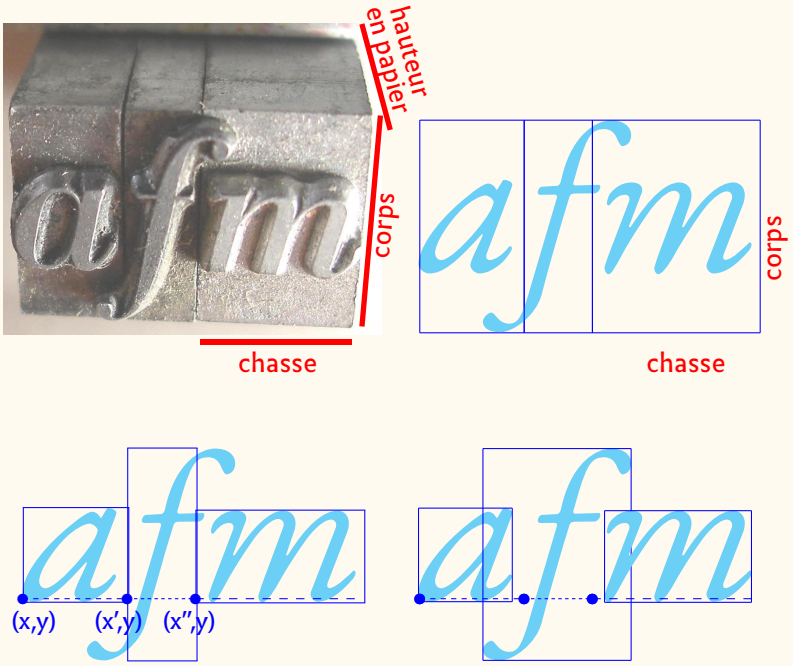


FIGURE 10 – Composition. En haut, au plomb (à gauche, photo de types retournée horizontalement ; à droite schéma). En bas, métrique vue par TEX (à gauche) et par `PostScript` et *Opentype* (à droite).
Notons que la crosse du « f » déborde sur le socle du « m ». (et sort de la boîte TEX). C'est ce que les fondeurs appellent « crénage ».
Notons aussi que le corps n'est pas identifiable sur les schémas des fontes numériques.

```
placer caract(a) en (x,y)
placer caract(f)
placer caract(m)
...
imprimer page
```

Dans ce pseudo-code, les variables `a`, `f`, `m` sont en fait les numéros de ces glyphes dans un codage donné (par exemple ASCII [79]). Le travail de l'imprimante est alors le suivant [80] :

1. ouverture du fichier `fonte` qui est dans un format défini (p. ex. fichier `.pfb` pour TEX, `.ps` pour Type1, le format d'Adobe) ;
2. récupération de la définition du glyphe de n° `a` ; il s'agit d'une série de commandes décrivant les contours du glyphe ;
3. construction d'une image de ce glyphe à l'échelle donnée par *corps* et remplissage des contours ;
4. placement de cette image dans la page en cours de construction au point de coordonnées (x, y) ;
5. ajout à (x) de la chasse de a (ou à (y) si l'on est dans un système d'écriture verticale ;
6. récupération de la définition du glyphe de n° `f`, etc.
7. ajout à (x) de la chasse de f ;
8. récupération de la définition du glyphe de n° `m`, etc. ;
9. ajout à (x) de la chasse de m ;
10. ...

79. *American Standard Code for Information Interchange* [anglais] : code américain normalisé pour l'échange d'information.
80. On trouvera dans [10] plus de détails !

FIGURE 11 – Extrait d'un fichier `.tfm`
(les commentaires sont des auteurs de cet article)

```
(CHARACTER C f % caractère f
   (CHARWD R 0.305557) % chasse de f (en em)
   (CHARHT R 0.694445) % hauteur de f
   (CHARIC R 0.077779) % profondeur de f
   (COMMENT
      (LIG C i O 14) % si f est suivi d'un i prendre LIGature n° 14 (fi)
      (LIG C f O 13) % si f est suivi d'un f prendre LIGature n° 13 (ff)
      (LIG C l O 15) % si f est suivi d'un i prendre LIGature n° 15 (fl)
      (KRN O 47 R 0.077779)  % indications de crénage (KeRNing)
      (KRN O 77 R 0.077779).
      (KRN O 41 R 0.077779)
      (KRN O 51 R 0.077779)
      (KRN O 135 R 0.077779)
      )
   )
```

Le problème est alors « comment l'imprimante connaît-elle les chasses de `a`, `f`, `m` ? » Le principe retenu est qu'à chaque fichier de fonte (donnant les instructions pour le tracé de tous les glyphes, de n° `n`) est associé un second fichier « de métriques » (ce sont les fichiers `.tfm`, *tex font metric* pour TEX, `.afm`, *adobe font metric* pour `PostScript`) où chaque case n° `n` donne la chasse du glyphe n° `n`. La figure 11 montre l'entrée [81] pour le caractère `f` d'une fonte où la seconde ligne donne la chasse `CHARWD` (*CHARacter WiDth*) mesurée en `em` pour être indépendante du corps actuel [82].

Ce principe, qui pourrait suffire, a été amélioré sur plusieurs points.

— Les imprimantes sont des automates de dessin, sans connaissance linguistique, par exemple pour diviser les mots en fin de ligne. Ce travail est donc laissé au logiciel de traitement de texte en amont, pour nous TEX, lequel du coup va gérer aussi la justification. C'est donc TEX qui va calculer la taille des espaces pour justifier une ligne. Ce qui, dans le langage de commande pour l'imprimante se traduira simplement par des modifications de l'abscisse du point courant *x* = … (en conservant les espaces qui ont une chasse fixe mais pas d'image, des exemples étant les cadratins, demi-cadratins et espaces fines). Il faut donc que TEX ait aussi [83] accès aux fichiers de métriques des fontes utilisées.

— Avant même que l'imprimé n'existe, les auteurs de manuscrits utilisèrent des ligatures dont nombre sont restées dans les imprimés, surtout celles techniques [84]. On ne peut pas demander aux auteurs de textes de les écrire en spécifiant les ligatures. Ni l'imprimante, ni le logiciel, comme TEX, ne sait quelles sont les ligatures présentes dans la fonte. On a alors imaginé que ce serait la fonte qui le signalerait, l'idée étant que cette information fasse partie des fichiers de métriques. Ainsi on voit en figure 11 que la fonte décrite signale qu'après la lettre `f`, si on a un `i` alors on pourra prendre le glyphe n° 14 qui représente `fi`, et de même pour `ff` et `fl`, ces glyphes 14, 15 et 16 ayant à leur tour une entrée dans le fichier des métriques. En particulier, le n° 15 pour `ff` annonce l'existence d'un glyphe `ffi`. On verra que la fonte *Infini* offre des ligatures de 4 lettres (comme ÆSŒ). Pour accélérer la recherche des ligatures

81. Les fichiers `.tfm` sont en binaire. On montre ici une version en ASCII lisible, obtenue dans le *shell* avec `fmtopl`.

82. 1 *em*, en français 1 cadratin, est une *unité relative* au corps : 1 *em*=la valeur corps, 1/2 *em*=la moitié de la valeur du corps. C'est donc une façon de donner les chasses indépendamment de la valeur exacte du corps.

83. Et même, il n'a besoin *que* de ce fichier, le contenu des boîtes ne lui servant à rien !

84. Les ligatures ont fait l'objet d'un *Cahier* GUTenberg entier [11]. On y appelle ligature technique celles nécessaires pour que deux caractères ne se touchent pas et esthétiques celles purement… esthétiques, donc non indispensables. Voir figure 12 page ci-contre.

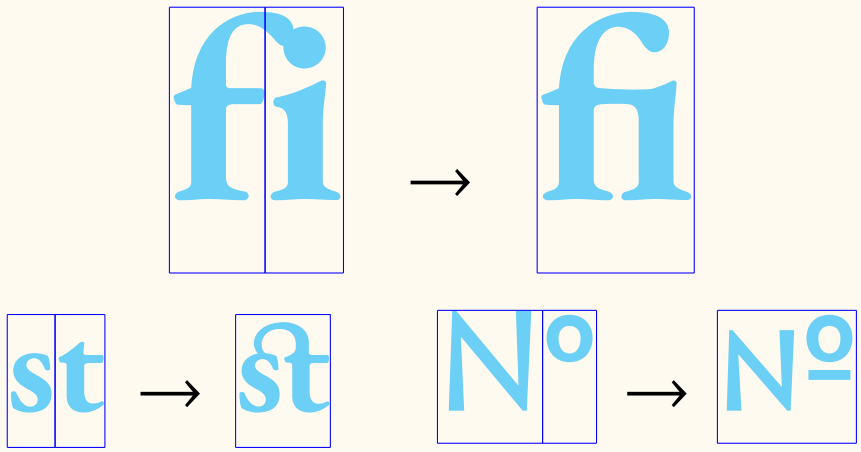


FIGURE 12 – Ligatures. En haut, au plomb la juxtaposition de « f » et « i » serait impossible car la crosse du « f » toucherait le point du « i ». On gravait alors le caractère spécial « fi ». En PAO, ça ne pose pas problème, sauf esthétique et on préfère là aussi avoir un dessin spécifique.
En bas, deux exemples de ligatures esthétiques.

a. Publication Assistée par Ordinateur.

possibles, plusieurs sous-tables sont construites, mais nous n'en dirons pas plus ici. Retenons donc que cette table de métriques permet de *substituer* à plusieurs glyphes un glyphe unique.

— Toujours dans ce souci de qualité, les bons typographes sont très sensibles à l'espacement entre les lettres (ou mots) et préfèrent composer « LAVE » à « LAVE », en faisant un crénage (en anglais *kerning* – ici crénage n'a pas le même sens que celui des fondeurs, figure 10 page 47), c'est-à-dire en resserrant le V et le A. Puisque cette connaissance appartient au dessinateur de la fonte, c'est à lui de donner les valeurs raisonnables des crénages possibles entre telle et telle lettre.Et là encore, c'est le fichier de métriques qui reçoit ces informations. La figure 11 page précédente montre ainsi des valeurs de crénage (lignes avec KRN). Pour ne pas trop charger cet article, nous avons préféré ne pas parler du tout de ces possibilités de *positionnement* (elles méritent de par leur importance un article à elles seules).

En résumé, dans les années 1980, une fonte numérique est constituée de deux fichiers :

— un fichier de description (mathématique) des glyphes ;
— un fichier, dit de métriques, contenant
  — la chasse de chaque caractère
  — des informations sur les possibilités de
    — substitution de glyphes
    — positionnement de glyphes.

Fontes numériques : un problème de communication

Une des problématiques fondamentales des fontes numériques peut se résumer en cette question : « Que doit taper sur son clavier d'ordinateur un utilisateur pour que son imprimante produise l'encadré de la figure 13 page suivante ? »

Comme Unicode (voir page 51), appelons *caractère* l'unité linguistique (ici la LETTRE MAJUSCULE LATINE Q) et *glyphe* une représentation possible d'un caractère. Ici à Q on associe les 11 glyphes de l'encadré et aux deux caractères QY le seul glyphe Qy, mais

FIGURE 13 – La fonte *OpenType EBGaramond* offre plusieurs variantes du caractère « Q », toutes au même corps.

Variantes de forme (de gauche à droite : styles romain, machine à écrire, gras, italique, avec paraphe, à queue allongée, orné pour titrage,...) ou de taille ou position (de gauche à droite, petites capitales, avec ou sans queue, en position supérieure ou inférieure) ; ligature « Qy ».

si l'on tient compte du corps, de la graisse, des éventuelles rotations, etc., ça ferait beaucoup plus.

Le principe des fontes numériques est donc qu'à chaque glyphe corresponde un programme de dessin occupant chacun une case d'un tableau, par exemple `T[i]` où `i` est le numéro du glyphe.

Un problème d'adressage se pose donc : une famille de fontes peut souvent contenir plusieurs milliers de glyphes. Or à l'époque, on travaillait essentiellement avec des octets (de 8 bit permettant d'y écrire les nombres de 0 à 255), on n'utilisait en général que des fichiers de 256 positions. Il a donc fallu découper la « fonte » T en de nombreux fichiers. On a bien sûr séparé les romain, gras, italique, etc. Mais ce n'était pas suffisant dès que l'on avait beaucoup de lettres accentuées ou de glyphes spéciaux : on a donc multiplié les fichiers. Cette ventilation peut se faire de nombreuses façons. Pour TEX, on a ainsi eu les systèmes comme NFSS [85] [3, 12] ou Omega [9]. Ce premier système est fondé sur un mécanisme de distinction et de description du contenu des fichiers par leur nom. De façon simplifiée, un utilisateur appellera une fonte par une commande de la forme `\usefont{`⟨`fam`⟩`}{`⟨`forme`⟩`}{`⟨`gr`⟩`}` où ⟨`fam`⟩ indique la famille (p .ex. `cm` pour la fonte de base de TEX *Computer Modern*, ⟨`forme`⟩ indique le style (i=italique, n=normal, b=gras, s=*slanted*, p=petites caps, etc.) et enfin ⟨`gr`⟩ indique la graisse (n=normal, g=gras, m=maigre). On a donc une kyrielle de sous-fontes par exemple `cmrn` pour le romain normal, `cmin` pour l'italique, `cmpg` pour de petites capitales grasses, etc. Ces sous-fontes sont alors déclarées dans des fichiers `.fd` (*font description*) comme celui-ci (présenté dans une version épurée) :

Exemple 20

```
\DeclareFontFamily{TSI}{mf11}{}
...
\DeclareFontShape{TSI}{mf11}{m}{sc}{<-> mf1r9c}
\DeclareFontShape{TSI}{mf11}{m}{it}{<-> mf1ri9c}
\DeclareFontShape{TSI}{mf11}{m}{sl}{<-> mf1ro9c}
...
```

Ce fichier signifie, en gros, que la « famille de fontes » imaginaire que nous appelons `TSI` est formée d'une forme (*shape*) de petites-capitales dont le fichier est `mf1r9c.pfb` (et le fichier de métriques `mf1r9c.tfm`), d'une forme italique (fichier `mf1ri9c`) et d'une forme penchée *(slanted)* (fichier `mf1ro9c`).

À cette base de données formée des divers fichiers, NFSS ajoute encore quelques fichiers pour une gestion plus efficace, dont les fichiers `.map`.

85. *New Font Selection Scheme* [anglais] : nouveau schéma de sélection de fonte.

Évolution des formats et codages

Très vite après qu'Adobe et Apple ont pris une part importante du marché des fontes, Microsoft lance son format de fonte TrueType pour attaquer ce marché. Fondé sur celui d'URW (notamment pour l'emploi de quadratiques pour dessiner les courbes), il se distingue des autres (TEX, `PostScript`/*Type1*, URW même) en ce sens qu'il n'y a plus qu'un seul fichier, agglutinant les deux des autres. C'est alors qu'a lieu la « guerre des polices » (en anglais *fontwar*, voir [13] pour son histoire détaillée) dont les principaux belligérants sont Apple, Microsoft et Adobe. Finalement il n'y aura pas de vrai vainqueur, sauf peut-être le format de fonte *OpenType* dont Microsoft est quand même le leader [86], mais où Adobe a apporté beaucoup (ainsi d'ailleurs que de nombreux organismes ou personnes à titre personnel). Nous allons revenir bien sûr sur ce format.

Il est important de signaler que depuis 1975 environ, Xerox, qui cherchait à attaquer le marché des imprimantes en Asie, commença à étudier un codage multilingue pour le monde entier (à l'époque l'ISO avait du mal à définir ses codages *Latin-N* pour les langues européennes) et s'associa très vite avec des partenaires informatiques comme Sun, IBM, Apple, etc. Ainsi fut créé le consortium Unicode et le codage du même nom [87]. En deux mots, disons que le codage Unicode :

- tente de coder toutes les langues du monde [88], passées ou actuelles ; la liste des caractères est ouverte et s'allonge de version en version ;
- traite de caractères linguistiques et non de glyphes graphiques ;
- utilise un système de numération hexadécimal (sur 16 bit donc) ;
- a limité le nombre de caractères possibles à 65 535 (FFFF en hexa) jusque 2012 puis à 1 114 111 (10FFFF en hexa). On note souvent, et on adapte ici cette notation, par U+004F le code hexadécimal 4F. À chaque nombre correspond
  - soit un « caractère », par exemple à U+004F correspond le caractère « O » et à U+1D11E (soit 119 070 en décimal) correspond « 𝄞 »,
  - soit un emplacement de caractère dans une zone privée (certaines de ces zones sont là où les fontes mettent les glyphes qui ne sont pas des glyphes de caractères Unicode),
  - soit à rien (il n'y a pas de caractère Unicode avec ce code) ; dans ce cas les fontes donnent un dessin spécial pour signaler l'erreur. Avec *Infini*, si on appelle le caractère U+0378 (qui n'existe pas) on récupère le signe ☒ ;
- propose une écriture condensée des caractères : théoriquement les caractères sont codés avec 16 bit, alors qu'en ASCII ou Latin-1, il suffit d'un mot de, respectivement 7 bit et 8 bit ; pour gagner de la place, Unicode a prévu un moyen de condenser ces valeurs Unicode ; l'une des méthodes employées (notamment pour les langues occidentales) s'appelle UTF [89]-8 ; c'est celle que nous utilisons avec LATEX depuis quelques années ; mais ça ne change rien sur les valeurs des codes !

86. *OpenType* est défini dans un document Microsoft [4] et le nom est une marque déposée de Microsoft ! Suite à divers accords, il y en a une version ISO (*International Organization for Standardization* [anglais] : organisation internationale de normalisation), OFF (*Open Font Format* [anglais] : format de fonte ouvert) synchronisée avec *OpenType*. Le détail de ceci peut se consulter sur `https://france2.wiki/wiki/OpenType`.

87. La version 1 date d'octobre 1991, même si la première base de données date de 2004. On en est aujourd'hui à la version 14 [5]. L'essentiel sur ce codage est décrit en français dans [14].

88. et même plus, notamment avec les émojis dont la liste ne cesse de s'alonger.

89. *Universal (Character Set) Transformation Format* [anglais] : format de transformation (du jeu) universel (de caractères codés).

Enfin, c'est vers ces dates 1980-1990 que sont apparus les premiers systèmes WYSIWYG [90] [91]. Si l'ergonomie de la saisie est très différente du codage TEX (on clique sur des menus au lieu d'insérer des commandes), le principe est le même : `Word` par exemple travaille de façon interne avec un langage balisé, dont on se rend compte en exportant un fichier `Word` [92] sous forme RTF [93].

OpenType en deux mots

OpenType est un format de fonte pour lequel il existe de nombreuses introductions, dont certaines plus orientées pour les utilisateurs de TEX [9, 15, 16].

— Comme *TrueType* dont il est issu, une fonte est formée d'un seul fichier, en fait une base de données contenant d'une part les description des glyphes en terme de quadratiques, et d'autre part l'équivalent des tables étendant considérablement ce qu'on avait dans les fichiers `.tfm`.

— Les glyphes sont numérotés en hexadécimal et respectent le codage Unicode pour les glyphes de caractères dans Unicode. Actuellement une fonte *OpenType* peut avoir jusque 1 114 111 caractères (valeur théorique jamais atteinte évidemment).

— Deux tables principales gèrent les possibilités d'accès aux glyphes : GSUB qui pour les *SUBstitutions* de glyphes, et GPOS pour le *POSitionnement* des glyphes (particulièrement utile pour les langues non-latines). Nous ne dirons rien de cette dernière. Depuis quelques années, une nouvelle grande table a été définie, spécialement pour les *MATHématiques*, dont nous ne dirons rien non plus ici (voir [17]).

— Ces tables, et notamment GSUB qui est donc la seule que nous considérons ici, peut être utilisée par les utilisateurs (notamment les systèmes de traitement de texte) via des *features* (fonctionnalités). Alors que TEX, par exemple, ne faisait guère de substitutions que pour les ligatures, *OpenType* utilise ce mécanisme pour gérer de nombreux cas, par exemple ceux permettant de traiter tous les « Q » de la figure 13 page 50. Ce sont ces substitutions que nous allons principalement étudier ici.

— Ces tables sont elles-mêmes divisées en sous-tables et accompagnées d'autres pour accélérer le travail de reconnaissance et de traitement lié à un glyphe, compte tenu des fonctionnalités de son contexte.

— Depuis quelques années seulement, *OpenType* donne de nouvelles possibilités comme le traitement de la couleur ou de fontes « dynamiques », beaux sujets pour des *Lettres* à venir !

OpenType et LuaLATEX & XELATEX

TEX a utilisé longtemps ses propres fontes, puis quand les fontes de Type1 ont envahi le marché, TEX a été adapté à ces dernières. En 2004, Jonathan Kew a écrit sa première version de XETEX qui fonctionnait avec les fontes AAT (*Apple Advanced Typography* : un

90. *What You See Is What You Get* [anglais] : ce que vous voyez est ce que vous obtenez.

91. Ce qui signifiait interactif. Typiquement en `Word`, on voit le texte composé sur écran à mesure qu'on le saisit.

92. Le format le plus récent de `Microsoft Word` est le `docx` qui est un fichier compressé au format `zip` contenant un fichier XML (*eXtensible Markup Language* [anglais] : langage de balisage extensible) pour le document. Même depuis cette forme `docx` on peut exporter un fichier en XML « lisible par l'homme ».

93. *Rich Text Format* [anglais] : format de texte enrichi.

intermédiaire entre Type1 et *OpenType*, voir [9]). Les fontes *OpenType* prenant à leur tour de l'ampleur, il est normal que XeTeX s'y mette dès la version de TeX Live 2007. C'est alors que Khaled Hosny et Will Robertson ont démarré le package fontspec permettant l'emploi de fontes *OpenType* tant pour XeLaTeX que pour le nouveau LuaLaTeX où *lua* est le nom d'un langage de programmation adapté à ce travail sur les fontes. Toute la machinerie des fontes est désormais dirigée par ce package fontspec tant pour gérer globalement les fontes (instructions `\setmainfont{...}` p. ex.), que le traitement particulier des fonctionnalités (`\addfontfeature{...}` p. ex.). Le manuel de référence [6] donne bien sûr tous les détails mais, pour une première approche, on recommande de consulter aussi des tutoriels comme [18], [19] et le *Cahier* GUTenberg consacré à LuaTeX [17].

Chargement de la fonte *Infini*

Tout d'abord : où la trouver ? Sur le site du CNAP [94] où elle est utilisable sous licence libre [95].

Une fois téléchargée, la fonte *Infini*, comprend quatre fichiers :

- `infini-romain.otf`
- `infini-italique.otf`
- `infini-gras.otf`
- `infini-picto.otf`

Les trois premiers constituent la fonte *Infini sensu stricto* ; le quatrième fichier, `infini-picto.otf`, est en fait celui d'une fonte casseau sur lequel on reviendra en page 73.

LuaLaTeX et XeLaTeX ne se comportent pas tout à fait de la même façon pour l'appel des fontes *OpenType*.

Il y a deux façons d'appeler une fonte, soit par le nom de la fonte (*fontname*), soit par le nom des fichiers (*filename*). En général c'est pratiquement la même chose, mais *Infini* se distingue car Sandrine Nugue a choisi de donner à la fonte les noms anglais traditionnels (p .ex. *Infini-Regular* et *Infini-Bold*), mais de donner aux fichiers des noms français (`infini-romain.otf` et `infini-gras.otf`), et pour corser le tout d'écrire les fontes avec une capitale initiale *(Infini)* et les fichiers avec une minuscule *(infini)*. Mais ce qui suit est compatible (et a été vérifié) pour les deux moteurs.

Pour accéder à la fonte *Infini*, on écrit simplement

```
\setmainfont{Infini}
```

et on aura directement accès aux fontes (avec l'usage éventuel de `\emph{...}` ou `\textit{...}` et de `\textbf{...}`).

On remarque de suite que la fonte n'a pas de *bolditalic*. Comme on aimerait mettre le mot *Infini* dans les titres de section de cette *Lettre*, lesquels sont en gras, on simule cette fonte manquante en prenant le gras et en le penchant [96], ce qui se fait avec la fonctionnalité [97] `FakeSlant` [6]. Cet adjectif *fake* (trop connu depuis que l'on parle des *fake news*...) dit bien que l'on ne crée pas un vrai italique : la figure 14 page suivante montre la différence.

94. https://www.cnap.fr/sites/infini/.
95. *Infini* est plus exactement utilisable sous licence libre Creative Commons CC-BY-ND
96. L'italique d'*Infini* a un « angle d'italique » de 5° (on le voit dans les propriétés de la fonte, par exemple avec `fontforge`). On donne donc comme paramètre à `FakeSlant` la valeur de `tg5=0,087`.
97. Pour être en accord avec Haralambous [9], on traduit ici *feature* par « fonctionnalité ».

FIGURE 14 – Comparaison des 3 fontes d'*Infini*. De gauche à droite : romain, italique et gras. À droite, un gras-italique qui est en fait un gras penché par nous utilisateurs, et non un vrai italique (regardez les « a » !).

Par ailleurs, on voit aussi que la fonte *Infini* n'offre pas de caractères « machine à écrire » dont nous, informaticiens, raffolons. Nous lui adjoignons donc une fonte, choisie par tâtonnements pour que le tout aille ensemble point de vue allure, graisse, etc. Notre choix s'est fixé sur *Noto Sans Mono* qui est distribuée avec TEX Live. Toutefois, son œil étant un peu plus gros que celui d'*Infini*, nous la réduisons un peu (de 5%) et choisissons la graisse `Medium`. Enfin, on intègre à cette déclaration de fonte celle du fichier `infini-picto.otf` sur laquelle nous revenons en page 73.

Finalement, la déclaration de fonte utilisée pour cette *Lettre* GUTenberg est la suivante :

Exemple 21

```
\setmainfont{Infini}[
     BoldItalicFont = *-Bold,
             BoldItalicFeatures={FakeSlant=0.087}]
     \newfontface\Picto{infini-picto}
\setsansfont{Infini}
\setmonofont[Scale =.95]{Noto Sans Mono Medium}
```

Cette déclaration, mise dans le préambule, a pour portée tout le document. Mais une telle déclaration peut aussi se mettre dans un groupe (par exemple dans un exemple ou une figure), sa portée étant alors réduite à ce groupe.

Présence des glyphes dans une fonte

Par le style associé

De nombreux packages (et leurs fichiers `.sty`) ont été écrits pour utiliser telle ou telle fonte *OpenType*, lesquels donnent en général accès à ses glyphes par des commandes spécifiques. Il suffit alors d'étudier la documentation associée. On se place donc ici dans le cas où ce package n'existe pas, ce qui est le cas pour *Infini*.

À l'aide d'un spécimen

En général, les fontes sont distribuées avec un spécimen, ou un catalogue. Ainsi *Infini* est accompagnée d'un spécimen [1]. Cela donne en général de grands services et une idée assez claire des caractères disponibles, même si souvent on ne voit pas comment les utiliser.

À l'aide de logiciels de gestion de fontes

Par logiciel de gestion de fontes nous entendons ces applications qui permettent de créer et éditer des fontes, comme `FontLab`, `Glyphs`, etc. (voir [9]). Nous utilisons

FIGURE 15 – Exemple de liste d'affichage des glyphes d'une fonte *OpenType*. Ici la fin d'*Infini* [21].

ici essentiellement `fontforge`, car il s'agit d'un logiciel libre — mais aussi pour ses qualités.

Une fois une fonte ouverte, `fontforge` affiche une fenêtre avec un tableau dont chaque case numérotée montre un glyphe, s'il existe dans la fonte avec ce numéro, et sinon une case vide. Ces glyphes sont normalement classés dans l'ordre des code-points d'Unicode (mais on peut choisir d'autres ordres, comme celui interne à la fonte). En cliquant sur un glyphe, on voit son nom, son numéro, etc. et en cliquant sur `view` on voit ses fonctionnalités métriques, ou les informations qui sont dans la table des fonctionnalités (substitutions, ligatures, etc.).

La fonte en cours de lecture peut aussi être sauvegardée sous forme d'un fichier, `fonte.sfd`, avec toutes les tables *OpenType* dans un format lisible.

Par programme

Les fontes *OpenType* sont des fichiers binaires, mais comme on vient de le voir, on peut facilement en avoir une version « lisible par l'homme », par exemple les fichiers `.sfd` de `fontforge`.

De son côté, la machinerie Lua/fontspec ne travaille pas directement avec le format OT mais convertit les fontes dans un format lisible en Lua [98].

Il est donc facile d'écrire un programme qui analyse le contenu de ces fichiers et qui, par exemple, sorte la liste des tous ces glyphes — comme le fait NFSS pour les 256 caractères d'une fonte TEX. Le site StackExchange en publie de temps en temps. La *Lettre* GUTenberg en a publié un dès 2015 [20] ; on trouvera un exemple de sortie d'un autre en figure 15 où on voit que chaque glyphe est accompagné de son nom et de son numéro Unicode.

Toutes ces méthodes montrent les glyphes présents, mais en général pas la façon d'y accéder. Nous allons donc voir comment faire en distinguant :

1. Les caractères Unicode d'accès direct.
2. Les glyphes qui s'appellent par le biais de commandes (en général par les fonctionnalités *OpenType*, utilisées souvent dans des commandes de plus haut niveau en LATEX). La simple juxtaposition de deux caractères pour en faire une ligature étant considérée comme une commande *OpenType* !

Accès aux caractères Unicode d'une fonte *OpenType*

Si l'on cherche à utiliser un caractère précis d'Unicode, il suffit d'essayer... En connaissant son nom (par exemple NUMERO ou OMEGA), on trouve facilement (ne serait-ce que

98. Ces fichiers sont conservés dans le dossier `/usr/local/texlive/⟨année⟩/texmf-var/luatex-cache/generic/fonts/otl/`.

sur le Web) son numéro Unicode (ici `2116` et `03A9`, en hexa) et en faisant `\char"2116` on obtient « № » ; avec `\char"03A9` on obtient « ⍰ » car ce glyphe n'existe pas dans *Infini*[99] ; avec *lmr* (*Latin Modern Roman*), on aurait « Ω ».

Une fonte contient toujours des glyphes correspondant « exactement » à des caractères Unicode ; ce sont les lettres et chiffres usuels tels que A, b, Đ, ž des langues latines, des signes divers + - ↖, des signes monétaires, des caractères anciens comme №, ¶, etc. et, selon les fontes, des caractères de langues non-latines (comme le grec, le coréen, les hiéroglyphes, etc.) mais aussi des caractères mathématiques. Disons-le tout de suite, nous ne nous intéressons dans cet article ni aux mathématiques ni à ces langues non-latines, d'autant que *Infini* ne les traite pas !

Comme pour les fontes TEX, l'accès à un caractère peut se faire soit par une touche du clavier, soit par une combinaison de touches (soit effectuée au clavier, au moment de la saisie, soit par une commande TEX figurant dans le code), soit enfin par un numéro.

Nous supposons être dans un état graphique où l'on a déjà choisi le style de la fonte (romain, gras, etc.), la force du corps, l'angle de rotation des caractères, voire leur couleur). Ici, ce qui nous intéresse, c'est comment choisir le signe §, pas son état graphique (qui pourrait donc être celui donnant « § » ou tout autre).

Entrée par numéro

En fait c'est la méthode de base… Pour cela on utilise, en TEX, la notation `\char"21A` pour avoir le caractère Unicode de code hexadécimal 21A (noté souvent U+021A, c'est la LETTRE MAJUSCULE LATINE T VIRGULE SOUSCRITE) dont *Infini* propose le glyphe « Ț ».

On utilise cette méthode lorsqu'aucune autre ne marche, par exemple pour écrire `\char"2116` qui donne « № ». Par ailleurs, si on appelle un caractère Unicode qui n'est pas dessiné dans une fonte, en général, la fonte lui substitue un glyphe signalant son inexistence. N'ayant pas les caractères grecs, hébreux, cyrilliques, etc., *Infini* imprime ⍰ (caractère inexistant) si on lui demande le caractère U+0416, LETTRE MAJUSCULE CYRILLIQUE JÉ dont un glyphe recommandé est, ici en *EBGaramond*, Ж

Par touches du clavier

Tout d'abord un petit rappel historique : en informatique lourde, les entrées-sorties entre un ordinateur et ses organes périphériques (lecteur/perforateur de cartes perforées ou de rubans, imprimantes, bandes magnétiques, etc.) étaient complexes jusqu'à l'arrivée, vers 1965, du concept de *driver* qui assure le dialogue entre l'OS[100] et ces périphériques pour manipuler chacun comme un processeur indépendant et échangeant des données « textuelles » normalisées (TTS, ASCII, EBCDIC, etc., aujourd'hui Unicode). D'autre part, depuis l'invention de la machine à écrire, vers 1890 [31, chap. 3], celle-ci était équipée de touches « bicamérales » (position haute ou basse pour les majuscules ou minuscules), de touches mortes permettant la saisie de plusieurs caractères en un seul, comme ^+e pour simuler ê. Les machines à écrire connectées à un ordinateur remontent essentiellement au système 360 d'IBM connecté à une machine Selectric, dotée de touches bicamérales. Cette écriture a bien sûr été adoptée par l'OS360, et continue à l'être par nos systèmes, Linux,

99. À la relecture de cet article, nous constatons qu'Infini possède un omega : le symbole de l'ohm, l'unité de résistance électrique : U+2126 OMEGA OHM SIGN, qui donne « Ω ». On admirera l'élégance de ce glyphe… tout en voyant confirmée l'information selon laquelle Unicode manipule des *concepts* et non des *formes*, à tel point que deux concepts différents, ici la lettre grecque et l'unité de mesure, qui ont la même apparence, ne donnent pas le même rendu : dans un cas, le glyphe désiré, dans l'autre, celui indiquant que ledit caractère est manquant.

100. *Operating System* [anglais] : système d'exploitation.

macOS ou Windows mais aussi Android par exemple. Disons que ce sont ces OS qui gardent la main, mais en offrant des possibilités de modifier, voire programmer, les « préférences » des claviers.

L'accès à un caractère depuis un clavier peut se faire directement en enfonçant, quand elle existe, la touche d'un clavier (A entre le caractère « A »). Puisqu'on travaille désormais souvent en UTF-8, nos systèmes d'exploitation (Linux, macOS, Windows, etc.) sont adaptés à ce codage et le seul fait de taper l'une des touches à, §, ou £ donne directement l'un des caractères *à*, *§* ou £. L'accès au caractère souhaité peut aussi se faire en respectant les contraintes de TEX de coder les caractères spéciaux et d'entrer donc, par exemple, \ & pour avoir &.

Tous les systèmes d'exploitation (associés à un type de clavier) proposent donc la possibilité de combiner plusieurs touches successives pour entrer un caractère. Tant que le caractère obtenu est bien un caractère Unicode, on peut utiliser ces combinaisons. Par exemple – sur un Mac où ⌥ indique la touche « option », ^ la touche « accent circonflexe » et ⇧ celle « majuscule » – on tape concurremment ^+o, ⌥+a ou ⌥+⇧+o pour avoir respectivement *ô*, *œ* ou Œ.

Dans cette catégorie, citons aussi la possibilité de faire un « copier » de caractères depuis un document (par exemple en PDF [101]) et un « coller » vers le texte courant, ce qui nous permet d'écrire ici en tchèque « Příliš žluťoučký kůň úpěl ďábelské ódy. »

Accents combinatoires

La grande force d'Unicode est la possibilité de combiner des caractères entre eux. Formellement, les caractères avec diacritiques (comme la LETTRE MINUSCULE LATINE E ACCENT CIRCONFLEXE U+00EA « ê ») sont définis par la combinaison d'un caractère (ici LETTRE MINUSCULE LATINE E U+0065 « e ») et d'un diacritique combinatoire (sans chasse, ici DIACRITIQUE ACCENT CIRCONFLEXE U+0302 «◌̂»), tout comme en TEX on peut écrire `\^e` pour avoir un « ê ».

Hélas, si *Infini* offre bien les lettres accentuées qui ont un numéro de code spécifique dans Unicode (ce qui couvre quand même toutes les lettres européennes) cette fonte n'a pas implémenté ces diacritiques flottants : on ne peut donc pas mettre d'accent circonflexe sur la lettre latine « æ » (caractère effectivement gravé au XVI^e siècle par Garamont) avec *Infini* alors qu'on le peut normalement (comme justement avec *EBGaramond*) :

Exemple 22

code

```
1 \setmainfont{eBGaramond}  æ  \^æ  æ\char"0302 \\
2 \setmainfont{Infini} æ  \^æ  æ\char"0302
```

résultat

æ æ̂ æ̂
æ æ□ æ□

Où l'on voit que ça ne fonctionne pas plus avec les commandes TEX, car elles font appel à un caractère flottant inexistant [102] ! Notons au passage que le positionnement

101. *Portable Document Format* [anglais] : format de document portable.
102. Il n'aura pas échappé au lecteur que c'est la police *Noto Sans Mono Medium* que nous avons utilisée en page 29 pour composer le nom de Hàn Thế Thành, faute de diacritique flottants permettant

(en hauteur, centrage, etc.) des diacritiques n'est pas facile à gérer automatiquement, c'est l'une des raisons de l'existence des fonctionnalités d'*OpenType* utilisant les tables GPOS ; mais nous n'en parlerons pas ici [103].

Par commandes TEX

Les commandes natives de TEX (prévues pour le codage minimal ASCII) sont toujours utilisables (`\'a\`e\k A\ss` imprime « áèĄß » ; de même que `\dag\S\P` donne toujours « †§¶ »).

Mais de très nombreuses autres commandes ont été définies, notamment dans des packages. Si les caractères utilisés sont présents dans la fonte courante, ils seront utilisés.Par exemple, si dans le préambule d'un document dont la fonte courante est *Infini*, on déclare un package CarAnciens (via un simple `\usepackage{CarAnciens}`) dans lequel ont été définies les commandes suivantes :

Exemple 23

```
\newcommand*{\VxParagraphe}{\char"00A7}
\newcommand*{\VxPdMouche}{\char"00B6}
```

... alors `\VxParagraphe` donnera bien « § » tout comme `\VxPdMouche` donnera « ¶ », et ce sans qu'il soir nécessaire de les redéfinir.

Mais en général, ces définitions utilisent leur propre fonte, et il est intéressant de voir comment. Le package fourier-orn, par exemple, définit `\FourierOrns` à utiliser dans un groupe comme suit [104] :

Exemple 24

```
\newcommand*{\warning}{{\FourierOrns 1}}
\newcommand*{\bomb}{{\FourierOrns 9}}
```

(ici les numéros de caractères sont en décimal) ce qui permet à un utilisateur d'écrire :

Exemple 25

code

```
% préambule \usepackage{fourier-orns}
\warning \bomb
```

résultat

C'est bien sûr ce que nous faisons avec les fontes *OpenType* (et de façon plus simple

de composer le chữ quốc ngữ, l'orthographe officielle latine vietnamienne, et donc le caractère ế, autrement dit la LETTRE MINUSCULE LATINE E ACCENT CIRCONFLEXE ET ACCENT AIGU, ayant pour code U+1EBF.

103. Nous ne parlons pas des tables GPOS car cet article, et donc cette *Lettre*, sont déjà beaucoup trop longs. Mais ce n'est que partie remise !

104. La commande `\FourierOrns` est faite pour fonctionner aussi bien avec LualaTEX/XeTEX qu'avec pdfTEX et il n'est plus fait appel au codage mais à des chiffres ou lettres ASCII. Les définitions intégrées dans fourier-orns sont un peu plus complexes afin d'être compatibles avec les signets du package hyperref.

puisqu'on veut justement accéder à un caractère qui est dans la fonte considérée). Si on doit écrire un package `Infini.sty`, on y mettra, par exemple,

Exemple 26

```
\newcommand*{\NUMERO}{{\setmainfont{Infini}\char"2116}}
```

ce qui permettra d'écrire « `J'habite au \NUMERO~17` » pour imprimer « J'habite au № 17 » si on préfère ce glyphe à celui de la macro `\No{17}` du package babel-french, laquelle donne « N° 17 » (voir page 69).

Accès aux glyphes d'*Infini*

Fonte *OpenType*, *Infini* permet d'accéder à des glyphes inconnus d'Unicode qui correspondent :

1. soit à des variantes de caractères Unicode (petites capitales, supérieures, lettres à panache, chevrons de grande taille, etc.),
2. soit à des caractères qui ne sont pas connus d'Unicode, par exemple les ligatures comme *ffi* mais aussi *ſt*, les signes spéciaux comme le zéro barré ø…

Dans ces deux cas, on « substitue » à un caractère un autre signe par une commande de fonctionnalité ; par exemple on substitue à « a » :

— un *a* en petite capitale « ᴀ », grâce à la fonctionnalité `smcp`,

— mais avec la fonctionnalité `sups`, ce sera un *a* supérieur « ᵃ »,

— tandis qu'avec `ss01` (fonctionnalité qui permet des actions spécifiques à chaque fonte) on obtient le grand chevron à gauche « ⟨ » (spécifique donc à *Infini*) ;

Nous reviendrons dans les sections suivantes sur ces fonctionnalités, leurs rôles et comment les utiliser.

Rappelons qu'une seconde série de manipulations que l'on peut faire avec des fontes *OpenType* permet des « micro-positionnements » de glyphes (crénage, position des diacritiques, etc.) ; ça ne concerne pas normalement l'utilisateur s'il fait confiance au dessinateur de la fonte [105] !

Principales fonctionnalités

Détection des fonctionnalités d'une fonte

Pour connaître les fonctionnalités d'une fonte OT, le mieux est d'utiliser un programme qui analyse cette fonte et en extrait lesdites fonctionnalités. La méthode la plus simple est la suivante.

Dans un *terminal* de votre système d'exploitation préféré, lancer la commande (où ⟨`mafonte`⟩ est le nom — avec suffixe — de la fonte) :

```
$ otfinfo -f ⟨mafonte⟩ > prop.dat
```

… et imprimer le fichier `prop.dat`. Dans le cas où ⟨`mafonte`⟩ est `infini-romain.otf`, on obtient la sortie suivante :

105. En revanche, la maîtrise de ces micro-positionnements de glyphes montre les compétences techniques indispensables à la création de caractères *OpenType*.

```
aalt Access All Alternates
c2sc Small Capitals From Capitals
case Case-Sensitive Forms
dlig Discretionary Ligatures
frac Fractions
hist Historical Forms
kern Kerning
liga Standard Ligatures
lnum Lining Figures
onum Oldstyle Figures
ordn Ordinals
ornm Ornaments
pnum Proportional Figures
sinf Scientific Inferiors
smcp Small Capitals
ss01 Stylistic Set 1
sups Superscript
tnum Tabular Figures
zero Slashed Zero:
```

On obtient le même résultat avec `infini-italique.otf` ; avec `infini-gras.otf`, on a en plus les lignes suivantes :

```
ss01 Stylistic Set 1
ss02 Stylistic Set 2
ss03 Stylistic Set 3
ss04 Stylistic Set 4
```

Nous reviendrons plus bas sur ces quatre lignes de *stylistic sets*, en pages 72–73.

Avec d'autres fontes latines, on trouverait aussi des choses comme :

```
cpsp   Capital Spacing
cswh   Contextual Swash
dnom   Denominators
fina   Terminal Forms
size   Optical Size
swsh   Swash
```

sur lesquelles nous dirons juste un mot en page 76.

Les fonctionnalités affichées par `otfinfo` le sont par ordre alphabétique, ce qui a pour défaut de cacher le fait qu'elles appartiennent à diverses classes de fonctionnalités [106].

Nous distinguons personnellement, et en tant qu'utilisateurs :

— les fonctionnalités permettant de choisir un glyphe différent en taille et position :

  `smcp` pour avoir des petites capitales et

106. On trouvera une liste quasi-exhaustive des fonctionnalités ainsi que des recommandations pour leur implémentation dans [7] et [22].

`c2sc` pour forcer les capitales à être « petites » ;

`sups` pour avoir des lettres minuscules et des chiffres supérieurs ;

`sinf` pour les chiffres inférieurs ;

`ordn` pour avoir des formes spéciales d'ordinaux ;

— les fonctionnalités spécifiques à l'allure des chiffres :

`lnum` pour choisir des chiffres alignés :

`onum` elzéviriens,

`pnum` à chasse variable,

`tnum` à chasse fixe,

`zero` pour avoir des zéros barrés ;

— pour accéder à des caractères spéciaux :

`ss01` **à** `ss20` pour des caractères propres à chaque fonte ;

— pour choisir des variantes de forme historiques :

`hist` pour avoir des formes anciennes (comme le *s* long ou le *I* en lieu et place du *J*),

`swsh` pour avoir des lettres à panache,

`case` pour le *i* sans point en petite capitales, le ß allemand, …

`fina` pour des lettres terminales,

`cswh` pour des panaches contextuels ;

— pour activer ou désactiver l'utilisation automatique de ligatures :

`hlig` classiques, autrement dit : historiques,

`liga` standard,

`dlig` spécifiques à cette fonte,

`frac` pour les fractions ;

— des commandes techniques de micro-typographie :

`kern` pour régler le crénage,

`size` pour faire de l'ajustement optique,

`aalt` qui sert à regrouper des accès aux tables OT.

Nous ne parlons pas ici de toutes les fonctionnalités pour les langues non-latines ni, comme nous l'avons déjà dit, des commandes de micro-typographie.

Utilisation des fonctionnalités avec fontspec

Options de fonctionnalités par défaut

Signalons que LuaLaTeX (ou XeLaTeX) utilise fontspec avec des options par défaut, de façon à ce que les utilisateurs de TeX retrouvent les conventions d'écriture (saisie du double tiret « -- » pour obtenir « – », saisie de la vraie apostrophe « ' » par la touche ' etc. [107]). En gros :

```
\setmainfont{Infini} = \setmainfont{Infini}[Ligatures=TeX]
```

Insistons sur le fait que cette option par défaut ne marche qu'avec `\setmainfont` :

107. Les ligatures traditionnelles ff, fi, fl relèvent d'un autre principe. Voir page 74.

Exemple 27

code

```
{\setmainfont{Infini}
    --  L'effet affligeant d'un  film ``flou'' !
}

{\fontspec{Infini}
    --  L'effet affligeant d'un  film ``flou'' !
}
```

résultat

– L'effet affligeant d'un film "flou" !
-- L'effet affligeant d'un film ``flou'' !

Nota : dans l'exemple ci-dessus, le travail sur l'apostrophe n'est pas visible : en effet, *Infini* a utilisé le même glyphe pour l'apostrophe (') et pour la *single quote* (').

Modification des fonctionnalités de la fonte courante

Les exemples que nous donnons ci-dessous considèrent qu'on est dans le programme-source de cette *Lettre* pour laquelle on a indiqué dans le prologue les spécifications suivantes :

Exemple 28

```
\setmainfont{Infini}[
     BoldItalicFont = *-Bold,
             BoldItalicFeatures={FakeSlant=0.087}]
```

Le principe est d'ajouter localement les fonctionnalités désirées à l'aide de la commande :

Exemple 29

```
\addfontfeature{RawFeature={xxx,yyy,...}} zzzz}
```

… où `xxx,yyy,...` indiquent les indicateurs (*tags*) des fonctionnalités choisies et où `zzzz` indique le source où ces fonctionnalités seront appliquées. La portée de ces fonctionnalités est le groupe où elles sont définies (ici le groupe est marqué par la paire `{...}`). Le fait d'écrire `-xxxx` revient à désactiver la fonctionnalité correspondante (si elle était active).

Le package fontspec propose dans certains cas une notation d'un peu plus haut niveau, comme `\addfontfeature{Numbers={OldStyle,Proportional}}`, que nous citerons à l'occasion.

La portée d'un `\addfontfeature` va jusqu'à la fin du bloc où est défini `\setmainfont`.

Si on veut que les fonctionnalités choisies s'appliquent à tout un bloc, on peut utiliser la notation suivante :

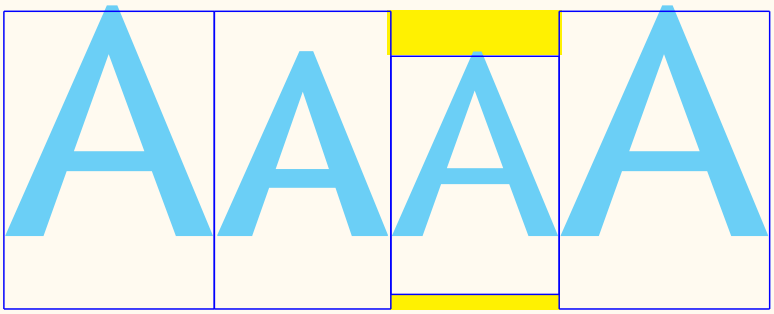


FIGURE 16 – De gauche à droite, « A » capitale, « a » petite capitale, « a » fausse petite capitale, « A » capitale. Toutes au corps 80, sauf la fausse petite capitale qui est en 64.

Exemple 30

```
\bgroup \setmainfont{Infini}[RawFeature={xxx,yyy,...}]
\egroup
```

… et si on veut que ce soit tout le document, il faut mettre cette commande, sans `\bgroup...\egroup`, dans le préambule.

Attention ! Le package fontspec a mis des pièges dans ses définitions !

`\addfontfeature` peut s'écrire aussi avec un `s` final : `\addfontfeatures` est également correct (sans doute pour distinguer « grammaticalement » les cas où il y a une ou plusieurs fonctionnalités). Nous avons évité cette écriture ici.

`RawFeature` au contraire ne prend jamais de `s` même s'il y a plusieurs fonctionnalités ; par ailleurs, ne pas oublier le `F` capital sous peine d'erreur à la compilation ; mais ne pas en mettre à `\addfontfeature`…

Nous allons voir une par une les principales fonctionnalités possibles avec *Infini*.

Variantes petites capitales

a $\xrightarrow{smcp}$ A

Les petites capitales sont apparues très tôt en imprimerie, où elles ont longtemps joué le rôle du gras. Elles sont systématiquement présentes dans de nombreuses casses [32]. Il est donc normal de les retrouver dans les fontes numériques, alors qu'Unicode ne les considère pas comme des caractères [108]. Lorsqu'un imprimeur n'avait pas de petites capitales, il prenait une capitale de plus petit corps (c'est ce qu'on appelle des « fausses petites capitales »). Mais la différence de graisse se voyait (voir figure 16) et il fallait « parangonner » c'est-à-dire rajouter des blancs (marqués en jaune sur la figure 16) en haut et en bas du type pour qu'il n'y ait pas de trous dans la composition. Ces trous ne représentant pas une gêne pour les fontes numériques, ces fausses petites capitales ont été beaucoup (trop) utilisées ces dernières années.

La fonctionnalité `smcp` (*small capitals*) donne accès aux petites capitales. Mais, puisqu'aujourd'hui pratiquement toutes les fontes en disposent (et ont donc cet attribut `smcp`), LuaLaTeX ou XeLaTeX sont capables de les détecter automatiquement et permettent d'utiliser la commande `\textsc` systématiquement. Avec *Infini* on a donc :

108. Théoriquement, Unicode ignore les petites capitales, considérant que ce sont des variantes glyphiques de lettres. Toutefois, pour des besoins spécifiques (notamment pour les alphabets phonétiques) beaucoup de petites capitales (toutes les lettres latines, sauf le x) y ont été introduites. D'ailleurs elles ne sont pas regroupées mais réparties un peu partout dans le BMP (le plan multilingue de base d'Unicode, qui correspond aux informations encodées sur ses 16 premiers bits et qui contient la plupart des écritures actuellement utilisées de par le monde).

Avec `\newcommand{\testSC}{AabC@}`

Exemple 31

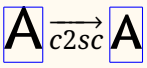
code

```
\testSC =>  \textsc{\testSC} \\
{\itshape  \testSC =>  \textsc{\testSC}}\\
 {\bfseries \testSC =>  \textsc{\testSC}}\\
{\bfseries \itshape \testSC =>
                \textsc{\testSC}}
```

résultat

AabC@=> AABC@
AabC@=> AABC@
AabC@=> AABC@
AabC@=> AABC@

où l'on remarque d'une part que le signe « @ » (qui a, dans *Infini*, la forme carrée « @ »), est considéré comme une lettre (dont il existe une variante « petite capitale », alignée sur la ligne de base) et d'autre part, puisqu'il n'y a pas de fonte « infini-italique-gras », LualaTeX (ou XeLaTeX) n'en invente pas et utilise le gras à la place (en le signalant toutefois dans le `.log`).



La fonctionnalité `c2sc` (*capital 2 (to) sc (small capitals)*) demande que les capitales mises en petites capitales soient elles aussi de la même taille (et de la même graisse) que les autres petites capitales, permettant de retrouver les petites capitales historiques [109]. La fonctionnalité `c2sc` permet d'écrire LOUIS XIV. Mais les auteurs de fontes interprètent différemment la fonctionnalité *Capital to small-capital* selon qu'ils la considèrent de façon exclusive (seules les capitales prennent la taille voulue) ou pas (les lettres, y compris les capitales, prennent la taille voulue). Par exemple :

Exemple 32

code

```
{\setmainfont{ebgaramond}EBGaramond :
 A{\addfontfeature{RawFeature=c2sc}AabB}B}

{\setmainfont{infini}infini :
 A{\addfontfeature{RawFeature=c2sc}AabB}B}
```

résultat

EBGaramond : AAabBB
infini : AAABBB

Si on veut définir une macro traitant le cas où tout est en petites capitales, et qui

109. Si on regarde les spécimens de caractères, les casses et surtout les textes imprimés avant les années 1950, on voit qu'en général toutes les petites capitales sont de la même hauteur, les majuscules ayant donc la même taille que les minuscules, c'est-à-dire qu'on écrivait LOUIS XIV. Les fontes numériques ont permis de mélanger des majuscules capitales avec des minuscules en petites capitales et d'écrire LOUIS XIV. La confusion est encore plus marquée quand on compare la hauteur d'œil des x et celles des petites capitales. Frey, au milieu du XIX^e^ siècle, avait inventé le terme de médiuscules et Microsoft parle, en anglais, de *petitecaps* comme on va le voir ! Enfin, rappelons qu'on appelle parfois « vraies petites capitales » celles de bonne graisse, en opposition aux *fake* obtenues par réduction du corps (figure 16 page précédente).

fonctionne quelle que soit l'interprétation de `c2sc`, on pourra écrire ceci :

Exemple 33

```
\newcommand{\textAllsc}[1]{{
  \addfontfeature{Rawfeature={scmp,c2sc}}#1}}
```

… et donc différencier les deux types de petites capitales :

	`\textsc{AabB}`	`\textAllsc{AabB}`
`\setmainfont{EBGaramond}`	AabB	AABB
`\setmainfont{infini}`	AabB	AABB

Petitecaps (en un mot)

C'est avec la revue *Emigre*[110] qu'est apparue dans les années 1980 une troisième taille de petites capitales : puisqu'on considère que les majuscules peuvent avoir la taille de petites capitales, *Emigre* a considéré que les minuscules devaient être plus petites et avoir la taille des minuscules ordinaires et a créé ainsi des *petitescap* (terme anglais, repris notamment par Microsoft [4, features]). *OpenType* lui associe la fonctionnalité pcap. *Infini* ne l'a pas implémentée, mais on la trouve dans *EBGaramond*.

```
\setmainfont{EBGaramond-Regular}
A
{\addfontfeature{RawFeature=smcp}a}
{\addfontfeature{RawFeature=pcap}a}
a
```

Désormais, on considère normal d'accentuer aussi les petites capitales ; certains auteurs de caractères étendent ce concept à des caractères autres que les lettres (dont des ligatures, @ et & font partie parfois) et notamment aux crochets, parenthèses, etc. C'est le cas de Sandrine Nugue dont la fonte *Infini* offre la possibilité d'écrire les paires de caractères suivants (normaux et en petites capitales) :

AA ÊÊ ŸŸ ((ŒŒ && @@]]

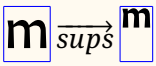
Lettres supérieures et chiffres supérieurs

Dès la fin du XV^e^ siècle, des imprimés français montrent des chiffres supérieurs (romains, notamment des x) dans des notations de nombres, comme iiii^xx^viii qui se lit « quatre vingt huit ». Abréger des mots est une habitude courante en écriture manuscrite [33] et se fait, entre autres, en utilisant des petites lettres en position supérieure. On trouve de telles « supérieures » dans les casses d'imprimeurs dès le XVII^e^ siècle. Par ailleurs, dès le début du XVI^e^ siècle, les appels de note sont marqués par des signes supérieurs comme « * », « ª » « † », etc. Vers 1750, Fournier propose 4 (vraies) supérieures (^aers^) ; la casse parisienne, qui a duré en gros de 1850 à 1950, en comptait 8, appelées ^roselmit^ ou ^eilmorst^ selon l'ordre de rangement dans les casses [32] ; en 1934, Brossard en énumère 16 différents (a c d e f g h i k l m n o r s t) dans une police standard – elles suffisaient pour les abréviations courantes.

La composition des formules mathématiques a utilisé des exposants (et indices) que les imprimeurs réalisaient dans des corps différents avec tout un travail de parangonnage (disons de calage) de ces bouts de métal dans le plan. Travail particulièrement délicat et que peu d'imprimeurs étaient à même de pratiquer correctement. On

110. *Emigre* était une revue américaine de recherche d'une nouvelle typographie basée essentiellement sur les possibilités juste inventées alors par `PostScript`.

sait que Donald Knuth créa justement TEX pour pallier cette incompétence. Mais en attendant TEX, les premiers systèmes de PAO avaient du mal à composer les formules [111]. C'est beaucoup pour la PAO que les premiers systèmes de fontes offrirent des lettres et chiffres « en exposant ». Le codage Latin1 proposa trois chiffres (¹ ² ³) et deux lettres (ª º, on verra plus bas que ce sont en fait des indicateurs ordinaux). Unicode à son tour reprit ces signes et les compléta par quelques signes (⁰ ⁴ ⁵ ⁶ ⁷ ⁸ ⁹ ⁺ ⁻ ⁼ ⁽ ⁾ ⁿ ⁱ) et en gros les mêmes en indice. Ce qui permet d'écrire quelques formules de chimie ou de physique sans recourir à des langages comme TEX.



OpenType propose donc des supérieures [112] sans préciser qui peut être en position supérieure (quid des signes Unicode que nous venons de citer ?) et notamment si les lettres latines pouvaient être accentuées ou pas ; on trouve donc des supérieures très diverses selon les fontes et l'interprétation que l'on donne à « supérieures » (exposants ou lettres abréviatives ?). Mais, par cohérence, si il existe un « y » supérieur « ʸ » (qui n'existe dans aucune abréviation), pourquoi refuser un « è » sous prétexte qu'il n'entre dans aucune abréviation correcte [113] ? L'avenir le dira, mais il semble bien que l'on va considérer bientôt que tout peut être mis en supérieures. Mais en attendant, il faut se dire que les supérieures ne sont pas réservées aux abréviations conventionnelles, même si nous nous ne savons pas à quoi ça sert…

C'est la fonctionnalité sups qui permet de choisir des lettres supérieures, comme le montre l'exemple suivant :

Exemple 34

code

```
M{\addfontfeature{RawFeature=sups}gr}
```

résultat

M^{gr}

Selon les fontes, répétons-le, on a plus ou moins de lettres supérieures, ce qui est illustré par l'exemple ci-après :

Exemple 35

code

```
\newcommand{\macasse}{fi PG æ œ \& è @ [(m+e)*=\$}
\begin{tabular}{rl}
avec \fnt{EBGaramond}&\setmainfont{ebgaramond}
 \Large \_{\addfontfeature{RawFeature=sups} \macasse}\_\\ \\
avec \Infini&\setmainfont{infini}
 \Large \_{\addfontfeature{RawFeature=sups} \macasse}\_\\
\end{tabular}
```

111. On lira l'expérience d'une célèbre imprimerie spécialisée dans ce domaine en France dans [34].

112. La définition d'*OpenType* [4] les dénomme *superscripts* et dit qu'ils ont pour fonction : « Replaces lining or oldstyle figures with superior figures (primarily for footnote indication), and replaces lowercase letters with superior letters (primarily for abbreviated French titles) »

113. Le mot « deuxième » s'abrège 2e, et la bonne typographie française refuse 2ème ; or cette forme est suffisamment populaire pour être tolérée au moins dans des écrits non-académiques ; de toutes façons, elle est correcte en francoprovençal… Signalons que nous avons contourné ici le problème en écrivant `\fontspec{ebgaramond}2\up{ème}}}` avec le package realscripts.

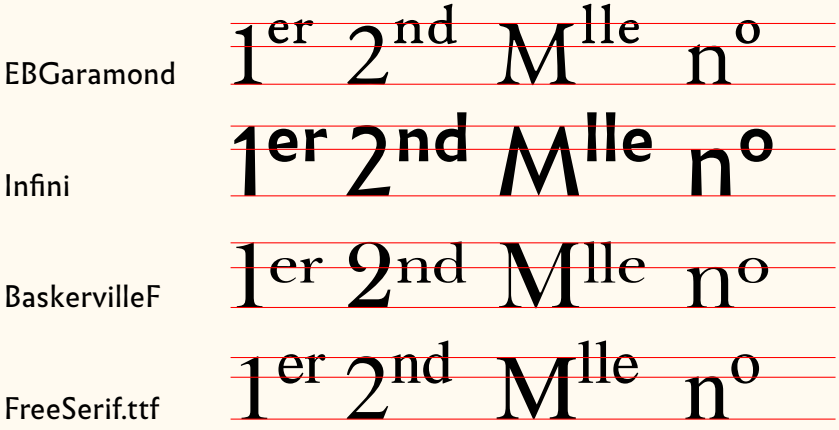


FIGURE 17 – La position des supérieures dépend de la fonte !

résultat (suite)

avec *EBGaramond* _fi PG æ œ & è @ [(m+e)*=$_

avec *Infini* _fi PG æ œ & è @ [(m+e)*=$_

Les seules lettres supérieures offertes par *Infini* sont donc des bas de casse.

Lettres inférieures

C'est peut-être par souci de cohérence que certains dessinateurs de fontes ont été amenés à concevoir des « lettres inférieures » qui, comme les supérieures, sont d'œil plus petit et sont positionnées en dessous (et non au-dessus) de la ligne de base. Fort peu utiles, elles ne sont que rarement implémentées dans les fontes *OpenType*, ce format prévoyant toutefois la fonctionnalité subs. *Infini* n'en a pas.

Chiffres supérieurs

Ils font aussi partie des lettres supérieures, au même titre que certains signes arithmétiques et parenthèses, comme le montre l'exemple suivant :

Exemple 36

code

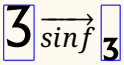
```
1 12{\addfontfeature{RawFeature=sups}34}56
```

résultat

$12^{34}56$

Les utilisateurs du package babel-french savent que la commande `\up{...}` permet de rendre supérieurs des signes, en positionnant plus haut que la ligne de base ces signes dans un corps plus petit. Toutefois, en utilisant le package realscripts, la commande `\up{...}` fait dans la mesure du possible appel à la fonctionnalité sups de la fonte pour utiliser les vraies supérieures. Mais ici `\_\up{\macasse}\_` donne _fi pg æ œ & è @ [(m+e)*=$_ : cela ne fonctionne pas.

Dans un style pour *Infini*, il faudrait donc redéclarer la commande `\up{...}`, pour qu'elle utilise directement la fonctionnalité sups.



Chiffres inférieurs

Les chiffres inférieurs existaient dans les fontes en matière au XIX^e siècle et ont été utilisés avec les premiers systèmes de traitement de texte (informatisés) pour faire quelques expressions mathématiques simples. *OpenType* en offre donc la possibilité par la fonctionnalité `sinf` (*scientific inferior*), mais ce ne sont pas les caractères utilisés pour faire des bonnes mathématiques (comme avec TEX) ! On observera l'exemple suivant :

Exemple 37

code

```
12{\addfontfeature{RawFeature=sinf}34}56
```

résultat

$12_{34}56$

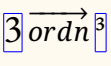
Notons qu'*OpenType* fournit des caractères comme « ² » et « ₃ », mais ce n'est pas à la fonte de composer X_3^2.

Un problème graphique ?

Autant la définition de « lettres supérieures » est floue, autant leur allure graphique est floue… Du temps du plomb, les caractères supérieurs avaient leur œil touchant le haut du type. La figure 17 page précédente montre que chaque dessinateur a sa propre interprétation. Si fondamentalement ça n'a pas une très grande importance, il faut quand même avouer que nous trouvons regrettable que les supérieures soient trop hautes dans certaines fontes (dans le cas de *Infini*, elles dépassent nettement le haut du type).

Ordinaux

À la demande de pays latins (Italie, Espagne, Portugal, etc.) le codage Latin1 a prévu deux positions pour des « nombres ordinaux », appelés respectivement féminin et masculin (même si les usages en sont différents selon les pays) qui jouent approximativement le rôle du ^e dans 3^e. Ces deux signes sont restés dans Unicode. Il s'agit des caractères U+00AA INDICATEUR ORDINAL FÉMININ et U+00BA INDICATEUR ORDINAL MASCULIN. Ces deux signes sont implémentés dans *Infini*, mais nous n'avons vu aucune fonctionnalité [114] pour les appeler autrement que par leur numéro : `\char"AA` et `\char"AB` ce qui donne « ª » et « º ». On remarque que, selon la tradition [115] (remontant peut être aux temps du plomb dans ces pays), les petits a et o sont soulignés.



Fonctionnalité `ordn`

Par ailleurs, si on regarde d'autres fontes, comme *EBGaramond*, on voit que ces signes d'ordinaux sont plus bas que les supérieurs équivalents (figure 18 page ci-contre). C'est sans doute pour imiter ceci que on a inventé le concept d'ordinaux [116] qui sont

114. Toutefois les *glyph info* de « A » et « O » font références à `aalt` et aux ordinaux.

115. Il n'y pas très longtemps encore, en France, quand on apprenait à écrire à l'école primaire on nous faisait souligner les abréviations, comme « M^me » ou « D^r », ce qui était banni en imprimerie dès la fin du XIX^e siècle.

116. La plus ancienne allusion à ces ordinaux plus bas que les supérieures que nous ayons trouvée ne remonte qu'aux années 1990 dans un document de Microsoft pour les dessinateurs de fontes *TrueType* qui a été adapté depuis pour *OpenType* [23].

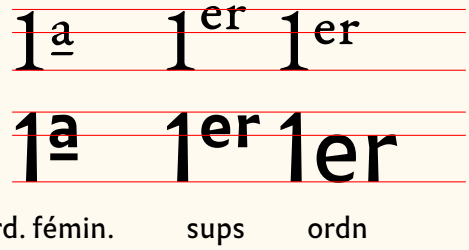


FIGURE 18 – Positionnement des ordinaux (non implémentés dans *Infini*).

des supérieures dont l'œil est au niveau de la hauteur des x. C'est ce que prévoit Opentype avec la fonctionnalité `ordn`. Peu des fontes que nous avons consultées proposent cette fonctionnalité (en particulier *Infini* ne la propose pas). Voir figure 18. À notre avis, ce positionnement est plus agréable à l'œil que celui, très haut comme c'est souvent le cas, des supérieures.

Le caractère « numéro » №

À la fin du XIX^e^ siècle, les spécimens des fonderies proposaient de nombreux caractères pour affiches parmi lesquels existait un « numéro » (par exemple № de Schelte, 1900). Assez curieusement (puisque ça pourrait être une ligature, voir figure 12 page 49), Unicode a prévu le caractère U+2116 SYMBOLE NUMÉRO, qu'*Infini* représente ainsi : « № ». D'autres fontes proposent des glyphes pour ce caractère, qui diffèrent toujours du résultat de la macro de Babel/french [24], comme le montre le tableau suivant :

Fonte	`\\char"2116`	`\\No`
Infini	№	N°
EBGaramond	№	N°
KpRoman	№	N°
LibertinusSerif	№	N°
TeXGyrePagella	№	N°

Chiffres

Le spécimen d'*Infini* [1] nous montre deux choses, au demeurant très classiques avec les fontes *OpenType* :

— par chiffre on entend les chiffres arabes usuels (plus une variante pour le zéro barré, voir ci-dessous), mais aussi les signes suivants : `# € $ £ ƒ ¥`, dont l'origine « comptable » ne fait pas de doute !

— ces chiffres existent sous deux formes : les chiffres normaux (d'origine calligraphique que l'on utilisait quasi-systématiquement jusqu'au XIX^e^ siècle, et qu'on commencé à appeler elzéviriens à la fin de ce siècle-là, tandis que les Américains employaient le mot *oldstyle*) et les chiffres alignés sur la ligne de base (nés en même temps que les linéales) et qu'on appelle donc alignés ou *lining*. Par ailleurs chacun d'entre eux peut avoir soit une chasse proportionnelle (ou variable, comme les lettres), soit une chasse fixe (la même pour tous les chiffres) de façon à aligner des nombres en colonnes de tableaux, d'où le nom qu'on leur donne aussi : tabulaires.

Trois fonctionnalités d'Open-Type gèrent les appels correspondants : `onum` pour avoir les chiffres *oldstyle*, c.-à-d. elzéviriens, `lnum` pour avoir les chiffres alignés et

pnum pour avoir les chasses variables (proportionnelles, et -pnum pour avoir les fixes). On pourrait utiliser les commandes `RawFeature=...` comme d'habitude, mais c'est ici l'occasion de montrer que fontspec utilise une commande de plus haut niveau : `Numbers=` dont les valeurs peuvent être `OldStyle` ou `Lining` et `Proportional` ou `Monospaced`.

En résumé, on a quatre jeux de chiffres comme le montre l'exemple suivant, qui utilise la macro `\test`, définie comme suit :

Exemple 38

```
\newcommand{\test}{%
  \_1234567890%
  {\addfontfeature{RawFeature=zero}0}%
  \# € \$ £ \textflorin{} ¥ \_%
}
```

Exemple 39

code

```
\begin{tabular}{|l|l|l|}\hline
&Elzéviriens&Alignés\\\hline
  Proportionnels&
 \addfontfeature{Numbers={OldStyle,Proportional}}\test&
\addfontfeature{Numbers={Lining,Proportional}}\test\\
  Tabulaires&
\addfontfeature{Numbers={OldStyle,Monospaced}}\test
&\addfontfeature{Numbers={Lining,Monospaced}}\test\\\hline
\end{tabular}
```

résultat

	Elzéviriens	Alignés
Proportionnels	_1234567890 0 # € $ £ ƒ ¥ _	_1234567890 0 # € $ £ ƒ ¥ _
Tabulaires	_1 2 3 4 5 6 7 8 9 0 0 # € $ £ ƒ ¥ _	_1 2 3 4 5 6 7 8 9 0 0 # € $ £ ƒ ¥ _

Pour les chiffres, Sandrine Nugue a jugé utile de choisir comme fonctionnalité par défaut les elzéviriens à chasse proportionnelle. Choix que nous respectons ici pour le texte. Toutefois, nous prenons le droit d'utiliser les chiffres alignés pour des choses techniques, trouvant cela plus heureux dans un contexte numérique.

Notons que si l'on emploie des chiffres supérieurs ou inférieurs (voir l'exemple 39), ceux-ci sont systématiquement des alignés tabulaires.

Zéro barré

Dans un récent *Cahier* GUTenberg, Charles Bigelow [25] a montré l'origine et l'utilité de cette variante du chiffre zéro qu'est le zéro barré « 0 ». Une fonctionnalité *OpenType* est prévue pour cela, `zero`. Pour avoir ce chiffre, il suffit d'écrire :

Exemple 40

```
  \addfontfeature{RawFeature=zero}0}
```

Si l'on s'en sert souvent, on mettra cette instruction dans une macro !

Fractions

On a vu dans la liste des fonctionnalités de *Infini* l'existence de `frac`[tions]. En fait il s'agit de ligatures permettant d'écrire par exemple « ¼ » au lieu de « 1/4 » et il serait logique d'en parler ici. Mais ce sont des ligatures et nous y reviendrons plus bas, en page 75.

Accès aux caractères spéciaux : `ss0i`

OpenType, reprenant d'ailleurs ce qui existait dans *TrueType* et *Type1*, offre des possibilités de définir des *stylistic sets*, des ensembles stylistiques de caractères, qui sont laissés au bon désir de chaque dessinateur de fonte. C'est souvent la meilleure façon de définir des glyphes qu'Unicode ne veut pas connaître. Typiquement, dans *EBGaramond*, c'est ainsi que sont codés les « Q à longue queue » (capitale, petite capitale, etc.). Le principe est d'avoir le Q à sa position Unicode (U+0051) et de mettre le Q dans une case d'une *private area* (zone inoccupée par Unicode pour justement y mettre des glyphes privés) voire en dehors du domaine Unicode (ici en 11000D). L'utilisateur n'a pas à connaître cette adresse mais il doit savoir comment y accéder. Ceci devrait être explicité dans quelque manuel accompagnant la fonte. Faute de quoi, il faut se débrouiller par exemple en regardant la fonte dans `fontforge`[117], ou en utilisant un programme d'analyse comme celui que nous citons juste ci-dessous. En l'occurrence, voici ce que l'on peut écrire pour accéder à ce Q long :

Exemple 41

code

```
1 \setmainfont{EBGaramond}
2 Qui ?
3 {\addfontfeature{RawFeature=ss06}Q}%
4 uoi ?
```

résultat

Qui ? Quoi ?

C'est aussi la façon de coder des caractères non alphabétiques non prévus par Unicode ou des variantes de caractères Unicode : on utilise alors des caractères du clavier pour leur saisie. Comme on va le voir, avec *Infini*, on appelle la flèche ↑ en utilisant la touche + (et `ss01`).

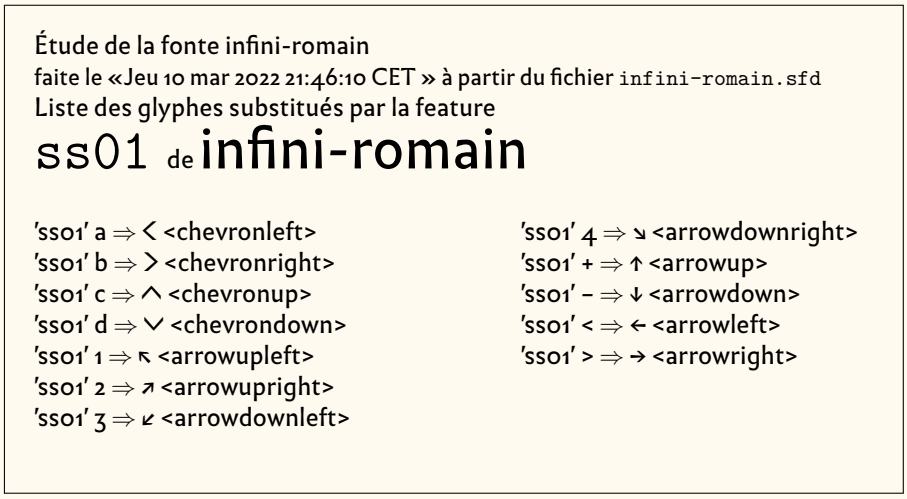
+ ss01 ↑

Ces substitutions stylistiques peuvent être ventilées, toujours au gré des dessinateurs, avec 20 classes possibles, de `ss01` à `ss20`. Sans que ce soit normé, on a pris l'habitude de réserver certaines classes à des cas spécifiques.

Comment connaître les glyphes concernés par des substitutions ?

Cette question se pose dès que l'on veut savoir comment utiliser au mieux une fonte *OpenType*. On vient de dire que d'une part on peut utiliser les documents décrivant la fonte, comme le manuel d'utilisation du package associé, s'il existe… On peut aussi consulter des produits comme `fontforge` (voir note 117).

117. On devine que ce doit être une substitution liée à « Q » ; dans la fenêtre principale de `fontforge`, on sélectionne la case de « Q » ; en regardant dans *glyph info*, on voit qu'effectivement la substitution `ss06` pointe sur le *Q.long* ; on peut alors rechercher, par nom, ce *Q.long* et voir que c'est bien ce que l'on attend.


Étude de la fonte infini-romain
faite le «Jeu 10 mar 2022 21:46:10 CET » à partir du fichier `infini-romain.sfd`
Liste des glyphes substitués par la feature
ss01 de infini-romain

'sso1' a ⇒ < <chevronleft>
'sso1' b ⇒ > <chevronright>
'sso1' c ⇒ ^ <chevronup>
'sso1' d ⇒ v <chevrondown>
'sso1' 1 ⇒ ↖ <arrowupleft>
'sso1' 2 ⇒ ↗ <arrowupright>
'sso1' 3 ⇒ ↙ <arrowdownleft>
'sso1' 4 ⇒ ↘ <arrowdownright>
'sso1' + ⇒ ↑ <arrowup>
'sso1' - ⇒ ↓ <arrowdown>
'sso1' < ⇒ ← <arrowleft>
'sso1' > ⇒ → <arrowright>

FIGURE 19 – Liste des glyphes d'*Infini* associés par substitutions stylistiques `ss01` (sortie d'un programme de [21]).

On peut utiliser aussi un programme d'analyse d'une fonte. L'idéal, en utilisant LuaLaTeX, serait d'utiliser une procédure écrite en `lua` faisant le travail voulu. Nous devons avouer ne pas en avoir trouvé. Nous avons donc conçu un tel programme, `analfonte` [21], encore en version α. En fait il s'agit d'un programme `perl` qui lit le fichier `.sfd` d'une fonte *OpenType*, tel que généré par `fontforge`, et qui en extrait un fichier de résultats prêt à être imprimé par un programme annexe écrit en LuaLaTeX. Par exemple, la figure 19 donne le résultat d'une recherche des glyphes associés à la commande `ss01` pour la fonte *Infini*. La sortie donne le glyphe de départ, le glyphe de substitution et son nom de glyphe [118] qui peut donner des indications supplémentaires. On trouvera figure 20 page ci-contre un autre exemple de sortie.

Infini et ses fonctionnalités `ss01`

Les trois fontes d'*Infini* montrent des substitutions de style activées par `ss01`. La liste en est donnée en figure 19 : elle vaut également pour les styles italique et gras.

Comme pour les autres substitutions, l'appel est très simple, par exemple :

Exemple 42

code

```
\setmainfont{Infini}
<ici> et
{\addfontfeature{RawFeature=ss01}a}%
là{\addfontfeature{RawFeature=ss01}b}
```

résultat

<ici> et <là>

118. *OpenType* a repris une forme de normalisation des noms de glyphes développée par Adobe [8]. On y retrouve une approximation des noms Unicode, p .ex. `Q` pour « Q », `Ugrave` pour « Ù », et des noms clairs pour les variants, `Q.long` pour Q long, `Q.swsh` pour le Q « swash » (à panache), `Q.sc` pour la small-cap, `Q.sups` pour la supérieure, etc. Et `f_f_i` pour la ligature ffi. Ces noms sont en général suivis pour toutes les fontes, mais il existe des déviances, notamment pour les choses compliquées.

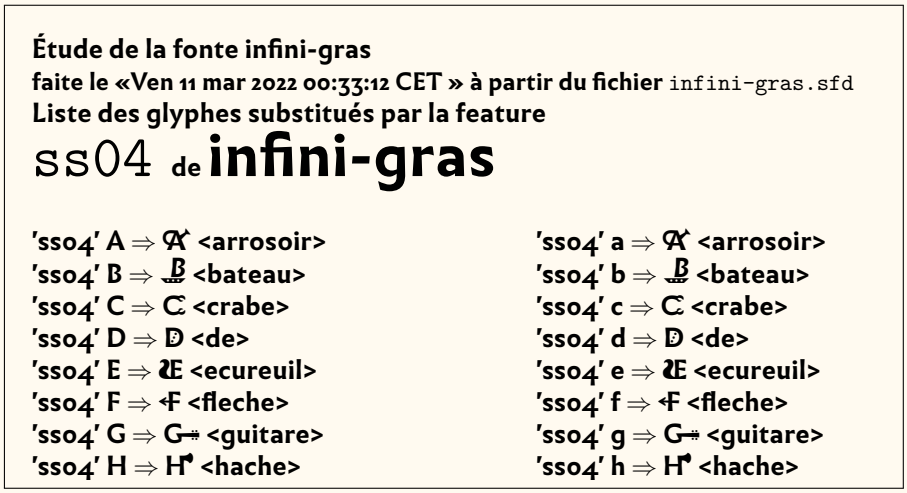


FIGURE 20 – Début de la liste des glyphes d'*Infini* associés par substitutions stylistiques `ss04`.

Pictogrammes d'*Infini*

Quand nous avons donné la liste des fonctionnalités possibles avec les fontes *Infini*, nous avions signalé l'existence de substitutions `ss04` pour le gras. Revenons-y.

La figure 20 donne la liste des glyphes concernés. On voit que si on tape un A, on obtient le caractère (dont le nom est `arrosoir`, ce qui confirme notre intuition d'un système acrophonique, en page 45). On obtient la même chose avec la touche a. De même pour le reste de l'alphabet majuscule/minuscule. On pourrait facilement écrire une macro comme :

Exemple 43

```
\newcommand{\picto}[1]{
  \textbf{\addfontfeature{RawFeature=ss04}#1}}
```

… avec laquelle `\picto{GUTenberg}` donnerait

La fonte casseau *picto.otf*

Ces pictogrammes de *Infini* gras peuvent intéresser des rédacteurs qui n'ont pas pour autant envie d'utiliser *Infini*. Sandrine Nugue a donc décidé d'en faire une fonte spécifique ne contenant que cela : ce que l'on appelle une fonte casseau. Comme on l'a déjà dit dans une précédente *Lettre* [20, 26], ces fontes n'ont aucune possibilité (comme l'italique, les petites capitales, etc.) des familles de fontes, ce ne sont que des tableaux de glyphes isolés. Dans ce cas, l'utilisation de `\fontspec{...}` est plus logique que celle de `\setmainfont{...}`.

Avec cette fonte, `infini-picto.otf`, les 26 pictogrammes occupent les positions des capitales correspondantes. Par exemple `{\fontspec{infini-picto.otf}S}` imprime .

Pour rendre les pictogrammes d'*Infini* utilisables depuis n'importe quel texte (dont la fonte courante est quelconque), on peut comme dans l'exemple suivant définir une macro `\picto` et l'appeler depuis un source quelconque (à condition de rendre accessible le fichier `infini-picto.otf`) :

Exemple 44

code

```
\newcommand{\picto}[1]{%
  {\fontspec{infini-picto.otf}\uppercase{#1}}%
}
\picto{Merci}
```

résultat

MERCI

Mais on peut aussi profiter des possibilités du package fontspec [6] sur les déclarations de fonte et définir par `\newfontface\Picto{infini-picto}` notre commande de fonte casseau ; alors `{\Picto FIN}` imprime FIN.

C'est ce que nous faisons dans cette *Lettre*, comme on l'a dit page 54.

Ligatures

Rappelons le mécanisme des ligatures en TEX : si dans le fichier d'entrée, TEX trouve une lettre qui peut être (selon le fichier `.tfm`) le début d'une ligature (p .ex. « f »), il regarde la suivante (p .ex. encore un « f ») et prépare la sortie de la ligature « ff » et continue ainsi (p .ex. la suivante est « i ») jusqu'à ce que l'on trouve un caractère non prévu dans la liste de `.tfm`, et alors il sort la ligature en construction (ici « ffi »). Ce mécanisme peut être désactivé.

On retrouve ceci avec les fontes *OpenType*, avec les différences suivantes : *OpenType* propose divers types de ligatures (avec les codes `liga`, `dlig`, ...), chacun pouvant être activé ou désactivé à tout moment et dépendre du contexte (nous n'en parlerons pas ici puisque ça sert surtout pour des langues non latines)

Infini propose essentiellement, comme toutes les fontes *OpenType*, deux types de ligatures :

— les ligatures standard, contrôlées par la fonctionnalité `liga` [119] ;

— et celles propres à la fonte et contrôlées par la fonctionnalité `dlig` (pour *discretionary ligatures*, en anglais).

Ligatures standard : `liga`

fj —liga→ fj

Ces ligatures sont tellement usuelles qu'elles sont souvent considérées par Unicode comme des caractères et ont donc leur code précis. Ce sont celles de la figure 21 page ci-contre. Ces ligatures sont systématiquement activées par le mécanisme lua/fontspec dès l'ouverture de toute fonte. En revanche, on peut les désactiver par l'option négative [120]. Exemple :

119. À ne pas confondre avec les ligatures *classiques*, ou *historiques*, que contrôle la fonctionnalité `hlig` (pour *historical ligatures*, en anglais).

120. On peut désactiver systématiquement toutes ces options par défaut au moyen de la commande `\addfontfeatures{Ligatures={NoRequired, NoCommon, NoContextual}}`.

FIGURE 21 – Liste des ligatures usuelles par défaut d'*Infini*.

«Étude de la fonte infini-romain
faite le «Ven 11 mar 2022 13:14:28 CET » à partir du fichier `infini-romain.sfd`
Liste des glyphes substitués par la feature
`liga` de infini-romain

'liga' f f => ff <U+FB00>
'liga' f i => fi <U+FB01>
'liga' f f i => ffi <U+FB03>
'liga' f j => fj <U+E182>
'liga' f f j => ffj <U+E180>
'liga' f l => fl <U+FB02>
'liga' f f l => ffl <U+FB04>

Exemple 45

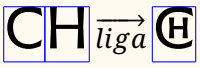
code

```
1  fin fjord effilé flou
2
3  {\addfontfeature{RawFeature=-liga}
4  fin fjord effilé flou}
```

résultat

fin fjord effilé flou
fin fjord effilé flou

Ligatures propres : `dlig`

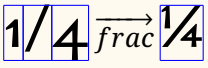


C'est sous cette dénomination que se mettent les ligatures non conventionnelles. *Infini* qui, on l'a vu dès le début de cet article, a pour spécificité sa richesse en ligatures « originales », va donc avoir dans ses tables beaucoup de places occupées par ces ligatures, puisque beaucoup sont composées de quatre lettres séquentielles (comme ASSE ou LIGA TURE qui donnent respectivement ꜩ et LIGꜩURE, que nous venons de saisir en tapant

```
{\addfontfeature{RawFeature=dlig}ASSE et LIGATURE}
```

La figure 22 page suivante montre quelques-unes des 54 ligatures relevées par notre programme d'analyse. On remarque qu'elles sont toutes regroupées dans une zone privée, aux numéros U+E0B9 à U+E17D.Leurs noms de glyphe respectent la nomenclature de la *Adobe Glyph List* [8], p .ex. `A_S_S_E` pour la combinaison des lettres `ASSE` (voir figure 15 page 55).

Fractions



Les fractions ont depuis longtemps préoccupé les graveurs de caractères qui proposèrent de nombreux types pour améliorer la composition : par exemple 1/4 est remplacé par ¼. En particulier, au XIX^{e} siècle, les spécimens des fonderies et imprimeries proposent souvent de très nombreux huitièmes, dixièmes etc. Il n'est donc pas étonnant qu'Unicode en ait repris certaines(figure 23 page 77). *OpenType* propose de les traiter comme des ligatures, c'est-à-dire que la saisie des trois caractères `1/4` donne donc « le » glyphe ¼. Comme toujours, le codage est simple :

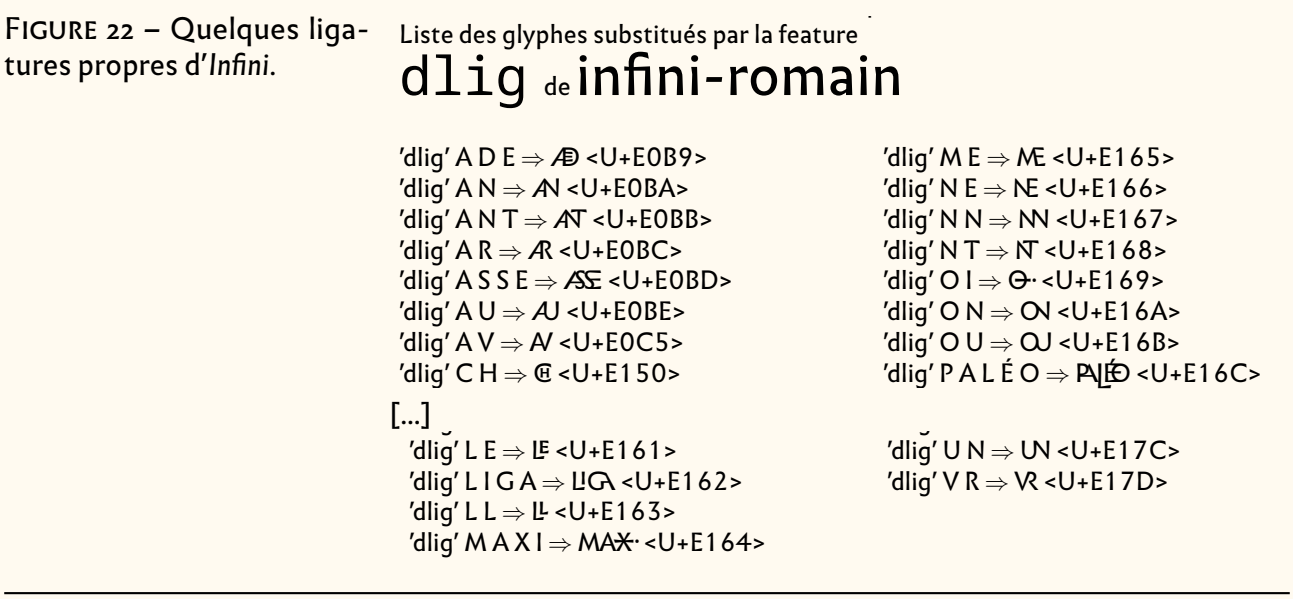


FIGURE 22 – Quelques ligatures propres d'*Infini*.

Exemple 46

code

```
1  1/4 + 1/2 = 3/4
2
3  {\addfontfeature{RawFeature=frac}
4  1/4 + 1/2 = 3/4}
```

résultat

1/4 + 1/2 = 3/4
¼ + ½ = ¾

Mais ces ligatures ne sont déclenchées que pour des glyphes initiaux prévus ; il n'est possible de faire une « belle » fraction 17/23 que si ces deux nombres sont spécifiquement prévus pour entrer dans la ligature `frac`. Ce qui n'est évidemment pas le cas. *Infini* se limite même aux seuls caractères prévus par Unicode (figure 23 page ci-contre). Néanmoins, on admirera le dessin de ces belles fractions, où l'inclinaison à 45 degrés de la barre de fraction ménage l'espace nécessaire à l'insertion des chiffres en haut à gauche et en bas à droit de cet élégant glyphe. Enfin, on pourrait créer de toutes pièces la « belle » fraction citée plus haut, en utilisant les chiffres supérieurs et inférieurs et en jouant sur leurs crénages respectifs, ce que permet fontspec ; mais une telle expérimentation dépasse le propos de cet article [121].

Autres substitutions

On a fait à peu près le tour des substitutions prévues pour *Infini*. Citons rapidement quelques autres substitutions, absentes d'*Infini*, en les montrant donc sur d'autres fontes.

121. Nous avons néanmoins utilisé un tel artifice dans les pages précédentes. Qui dans le lectorat l'aura repéré ?

FIGURE 23 – Les ligatures de fractions prévues par *Infini* donnent des caractères Unicode.

'frac' 1 / 4 ⇒ ¼ <U+ BC>
'frac' 1 / 4 ⇒ ¼ <U+ BC>
'frac' 1 / 2 ⇒ ½ <U+ BD>
'frac' 1 / 2 ⇒ ½ <U+ BD>
'frac' 3 / 4 ⇒ ¾ <U+ BE>
'frac' 3 / 4 ⇒ ¾ <U+ BE>

'frac' 0 / 0 ⇒ % <U+ 25>
'frac' 0 / 0 ⇒ % <U+ 25>
'frac' 0 / 0 0 ⇒ ‰ <U+2030>
'frac' 0 / 0 0 ⇒ ‰ <U+2030>

Substitution `titl`

Cette substitution permet d'utiliser une graphie différente des caractères pour faire des titrages. Souvent ces caractères, uniquement en capitales, sont un peu plus grands (mais avec la même graisse) que les capitales, voire occupent tout le corps (ils ont alors une profondeur (*depth*) quasi-nulle, comme les « lettres de deux points » autrefois). Mais souvent, ils permettent de faire un dessin légèrement moins académique que les caractères normaux « à lire », mais ne seraient alors plus utilisables en caractère du texte courant. Les caractères pictogrammes d'*Infini* auraient très bien (et même mieux) pu être codifiés comme tels. Voici un autre exemple, celui de la fonte *Brito tri*, de Fañch Le Henaff, utilisée ici dans sa version *du* (ce mot breton signifie *noir* et désigne ici le corps gras de ces caractères) qui propose pour le breton des caractères de titrage d'allure plus celtique :

Exemple 47

code

```
\setmainfont{BritoTri-Du}%
Park an Arvorig

\addfontfeature{RawFeature=titl}
Park an Arvorig
```

résultat

Park an Arvorig
Park an Arvoriz

Substitution `swsh`

Ces substitutions permettent de remplacer des lettres (souvent italiques) par leurs variantes « avec panache », comme celles à la mode notamment aux XVII-XVIII^es siècles.

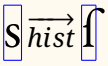
Exemple 48

code

```
\setmainfont{EBGaramond-Italic}%
Axe
{\addfontfeature{RawFeature=swsh}Axe}
```

résultat

Axe *Axe*



Substitutions `hist` **et** `hlig`

La substitution `hist` permet d'avoir des formes anciennes de lettres. C'est typiquement ce qu'on utilise pour le « s long », qui était jusqu'à la fin du XVIII[e] siècle la forme normale « ſ » du s ; ce signe existe dans Unicode (LETTRE MINUSCULE LATINE S LONG U+017F). On y met aussi des formes qui ont pu exister quelques temps, notamment avant une certaine normalisation des types. On trouve ainsi, selon les fontes, la capitale « J » (qui s'écrivait « I » avant son « invention » au XVI[e] siècle), des esperluettes, etc.

Dans ce même esprit d'écriture à l'ancienne, Unicode définit la ligature « ﬅ » (LIGATURE MINUSCULE LATINE S LONG T, U+FB05) et il est donc normal qu'*OpenType* permette d'utiliser cette ligature et d'autres éventuelles, ce qui se fait avec la fonctionnalité `dlig` qui n'est pas implémentée dans *Infini*. Mais avec *EBGaramond* on peut écrire :

Exemple 49

code

```
\fontspec{EBGaramond-Regular}
\addfontfeature{RawFeature={hist,hlig}}
          Je sais qu'il est estudiant.
```

résultat

Ie ſaiſ qu'il eſt eſtudiant.

Mais cette sortie est erronée : on sait en effet que le « ſ » était remplacé par le « s » dit rond en position terminale et devant quelques lettres comme b ou k. Il faut donc tenir du contexte pour écrire correctement cette phrase. Et ça c'est possible avec *OpenType* qui propose quelques fonctionnalités dépendant du contexte, telles que `salt` (*Stylistic Alternates*), `cswh` (*Contextual Swash*), etc. Mais comme nous avons dit nous limiter au simple cas de ce qui figure dans *Infini*, nous n'en parlerons pas cette fois.

De même, nous ne parlerons pas non plus des autres possibilités offertes par les tables GPOS et MATH : pour intéressantes qu'elles sont, elles ajouteraient des dizaines de pages à cet article déjà trop long.

Conclusion

Infini est donc une fonte que nous trouvions intéressante à essayer ici. Nous espérons avoir montré comment faire pour s'en servir, préférant cette approche ascendante de prendre les besoins d'un utilisateur un par un et de monter dans les difficultés (on aurait pu terminer en écrivant un « style » pour l'usage de cette fonte) à une approche descendante qui aurait consisté à citer ce « style » à utiliser et le démonter à titres d'exemples.

Remerciements

Nous tenons à remercier les diverses personnes qui nous ont fourni des informations reprises dans cet article ou qui nous ont aidé à le mettre en forme, en particulier Daniel Flipo, René Fritz, Christian Laucou, Denis Bitouzé, Maxime Chupin.

Patrick Bideault & Jacques André

Références

[1] Sandrine NUGUE. *Spécimen du caractère typographique Infini*. Centre national des arts plastiques, 2014. URL : https://www.cnap.fr/sites/infini/telechargement/specimen-infini_web.pdf.

Manuels de référence

[2] *LuaTeX Reference Manual*. Juill. 2021. URL : https://www.pragma-ade.com/general/manuals/luatex.pdf.

[3] Philipp LEHMAN. *The Font Installation Guide. Using Postscript fonts to their full potential with Latex*. Déc. 2004. URL : http://mirrors.ctan.org/info/Type1fonts/fontinstallationguide/fontinstallationguide.pdf.

[4] *OpenType® Specification (OpenType 1.9) - Typography*. 1er avr. 2022. URL : https://docs.microsoft.com/en-us/typography/opentype/spec/.

[5] THE UNICODE CONSORTIUM. *The Unicode® Standard Version 14.0.0*. Sept. 2021. URL : https://www.unicode.org/versions/Unicode14.0.0/.

[6] Will ROBERTSON. *The fontspec Package*. 1er avr. 2022. URL : http://mirrors.ctan.org/macros/unicodetex/latex/fontspec/fontspec.pdf.

[7] *Registered Features, a-e (OpenType 1.9) - Typography*. 1er avr. 2022. URL : https://docs.microsoft.com/en-us/typography/opentype/spec/features_ae.

[8] *Adobe Glyph List Specification*. Adobe Type Tools, 2019. URL : https://github.com/adobe-type-tools/agl-specification.

Articles et notes techniques

[9] Yannis HARALAMBOUS. *Fontes & Codages*. O'Reilly France, 2004. URL : https://hal.archives-ouvertes.fr/hal-02112931.

[10] Jacques ANDRÉ. « Caractères numériques : introduction ». In : Cahiers *GUTenberg* 26 (1997), p. 5-44. URL : http://www.numdam.org/item/CG_1997___26_5_0/.

[11] *Ligatures & caractères contextuels* 22 (1992) : Cahiers *GUTenberg*. ISSN : 1140-9304. URL : http://www.numdam.org/item/CG_1995___22/.

[12] Frank MITTELBACH et Michel GOOSSENS. *LaTeX Companion*. Trad. par Jacques ANDRÉ et al. 2e éd. Paris : Pearson Education France, oct. 2005. 1120 p. ISBN : 978-2-7440-7182-9.

[13] Charles BIGELOW. « The Font Wars, Part 1 ». In : *IEEE Annals of the History of Computing* 42.1 (jan. 2020), p. 7-24. ISSN : 1934-1547. DOI : 10.1109/MAHC.2020.2971202.

[14] Patrick ANDRIES. *Unicode 5.0 en pratique : Codage des caractères et internationalisation des logiciels et des documents*. Dunod, 16 avr. 2008.

[15] Paul ISAMBERT. « OpenType Fonts in LuaLaTeX ». In : *TUGboat* 33.1 (2012), p. 59-85. URL : https://tug.org/TUGboat/tb33-1/tb103isambert.pdf.

[16] Martin WENZEL et Christoph KOEBERLIN. *An Introduction to OpenType Substitution Features*. URL : https://ilovetypography.com/OpenType/opentype-features.html.

[17] *Introduction à LuaTeX* 54-55 (2010) : Cahiers *GUTenberg*. ISSN : 1140-9304. URL : http://www.numdam.org/issues/CG_2010___54-55/.

[18] Daniel FLIPO. « De pdfLaTeX à LuaLaTeX ». URL : http://daniel.flipo.free.fr/doc/luatex/pdf2lua.pdf.

[19] Manuel PÉGOURIÉ-GONNARD. « Un guide pour LuaLaTeX ». In : Cahiers *GUTenberg* 54-55 (2010), p. 13-35. ISSN : 1140-9304. URL : http://www.numdam.org/item/CG_2010___54-55_13_0/.

[20] Jacques André. « TeX Gyre Pagella et autres fontes Unicode en utf-8 ». In : *La* Lettre *GUTenberg* 39 (2015).
[21] Jacques André. « Utilitaires de manipulation de fontes OpenType ». 2015. url : http://jacques-andre.fr/fontex/utilitaires+OpenType.pdf.
[22] *Syntaxe des fonctionnalités OpenType en CSS*. url : https://helpx.adobe.com/fr/fonts/using/open-type-syntax.html.
[23] *Character Design Standards*. Microsoft Typography documentation. url : https://docs.microsoft.com/en-us/typography/develop/character-design-standards/.
[24] Daniel Flipo. *Mode d'emploi du module babel-french*. 1er avr. 2022. url : http://mirrors.ctan.org/macros/latex/contrib/babel-contrib/french/frenchb-doc.pdf.
[25] Charles Bigelow. « Histoire d'O, d'o et de o ». In : Cahiers *GUTenberg* 57 (2012), p. 5-53. issn : 1140-9304. url : http://www.numdam.org/item/CG_2012___57_5_0.pdf.
[26] Jacques André. « Utilisation locale de fontspec pour des caractères spéciaux ». Sept. 2018. url : http://jacques-andre.fr/fontex/casseau+fontspec.pdf.

Autres références

[27] Fred Smeijers. *Les Contrepoinçons : Fabriquer des caractères typographiques au XVIe siècle, dessiner des familles de caractères aujourd'hui*. Trad. par Amarante Szidon. 1re éd. Paris : B42, 14 nov. 2014. 228 p. isbn : 978-2-917855-51-5.
[28] Patrick Bideault. « Un ouvrage historique — *Les Contrepoinçons* de Fred Smeijers ». In : *La* Lettre *GUTenberg* 44 (2021).
[29] Jacques André. *Histoire de l'écriture typographique. Le XXe siècle*. T. I. II t. Atelier Perrousseaux, 2016. Chap. Louis Jou, un marginal génial.
[30] Geoffroy Tory. *Champ fleury, au quel est contenu lart et science de la deue et vraye proportion des lettres attiques, quon dit autrement lettres antiques et vulgairement lettres romaines, proportionnees selon le corps et visage humain*. 1529. url : https://gallica.bnf.fr/ark:/12148/btv1b86095803.
[31] Jacques André. *Histoire de l'écriture typographique. Le XXe siècle*. T. II. II t. Atelier Perrousseaux, 2016. 253 p.
[32] Christian Laucou. *Histoire de l'écriture typographique. Le XIXe siècle*. Atelier Perrousseaux, 2013. Chap. Variations sur la casse française et son rangement.
[33] Jacques Poitou. *Abréviations et Sigles*. Langages, écritures, typographies. url : http://j.poitou.free.fr/pro/html/typ/abrev.html.
[34] Maurice Laugier. « La composition des mathématiques. Évolution des techniques au travers d'une expérience professionnelle ». In : Cahiers *GUTenberg* 43 (2003), p. 5-32. url : http://www.numdam.org/item/CG_2003___43_5_0.pdf.