Ω + Virtual METAFONT = Unicode + Typography (First Draft)

Yannis HARALAMBOUS et John PLAICE

6th January 1996

Abstract

In this paper we present the principles and first stages of development of the Unicode-based Ω output encoding UT and it's realization in the frame of the Computer Modern font meta-family. The technique for obtaining virtual fonts for all styles and point sizes of Computer Modern parameters, called Virtual METAFONT, is also described. Finally future plans and a raw timing for forthcoming Ω development is given.

Résumé

Dans cet article nous présentons les principes et les premières étapes de développement du codage UT de police Ω, qui couvre les besoins du codage d'échange d'informations Unicode, ainsi que de sa réalisation dans le cadre de la famille de meta-polices Computer Modern. La technique qui permet d'obtenir des polices virtuelles pour un style et un corps donné, à partir des paramètres METAFONT correspondant, appelée « METAFONT virtuel » est également décrite. Finalement nous donnons un aperçu des futurs développements Ω.

1 Preliminaries

1.1 Unicode and Typography

Unicode is an encoding intended for information interchange, this has been said and written zillions of times. But what does it really mean? how do we distinguish an encoding for typography and one for information interchange? the limits are not clear, and in our case it is even worse: Unicode is not always consistent in that extent. After all, who can really say what is typographic tradition and what belongs to the linguistic "features" of a language?

Let's take an example: in Greek, a variant of the letter sigma 'σ' is used at the end of a word: the *final sigma* 'ς'. This glyph has not always been used: in several critical editions of ancient Greek texts (but never in Greece), a glyph close to the Latin 'c' is used, *for both forms*. One could say: "that proves the fact that the medial/final variation is purely typographic; hence, it should not be taken into consideration in an information interchange encoding". Nevertheless, there are two sigmas in Unicode (even three: the 'c'-like sigma is also included, it is called GREEK LUNATE SIGMA SYMBOL). A possible justification of this, is the fact that in some cases the medial/final difference can be linguistically significant: when a word is abreviated at a sigma letter, the latter remains medial, so that φιλοσ. makes sense: it may be φιλοσοφία or φιλοσοφικῶς, and so on. But this case is rare, and the same information can be retreived from the context; so this is hardly a reason for including a final sigma in an encoding such as Unicode.

The same kind of questions arises for most of the writing systems. In some cases the answer is trivial, in others very hard. Another example: in Hebrew, there are five consonants having final versions, like in Greek. But contrarily to Greek, these glyphs are used as numerals, with different values than the corresponding medial forms. This argument is strong enough to prove the necessity of their inclusion, but should they be encoded as letters or as numerals? When an Hebrew text is written (except in some very rare cases in the Bible), the form of these consonants (medial vs. final) can be algorithmically defined; this means that a distinction medial/final is absolutely needless for transmitting an Hebrew text, since the rendering machine can "do the work". Final Hebrew consonants should be coded only as numerals (in Unicode, both Arabic and Roman numerals are coded as such). Well, :-(this is not what happened: final consonants are coded just as medials are. When Unicode officials are asked for the reasons of such decisions, they usually reply: "for reasons of compatibility with pre-existing local standards"...

This is not a critique of Unicode, just a couple of examples illustrating the difficulty in deciding what is typography and what is "pure" information. In the remaining of this paper we will take Unicode "as is" (no more critique, or... almost), and see how Ω can make the most out of it.

1.2 The Weak Sides of TeX

TeX is marvelous, it is one of the major achievements of this century. But it also has some weaknesses, especially when we want to make it grow into Ω, a Unicode-capable tool. Take for instance the way TeX treats fonts: the fonts used in DVI files have a certain encoding, this encoding *must* be used also for hyphenation patterns. To be able to combine arbitrary font encodings with arbitrary hyphenation patterns, we can use virtual fonts; but these, (a) have to be generated, (b) need an extra step in processing (devirtualization) or DVI drivers compatible with virtual fonts (not the case of every DVI driver).

Suppose now that you are using a font with a lot of ligatures: for example a calligraphic font. If the ligatures are performed by the TFM file, then they do not interfere with hyphenation patterns. But it may also happen that these ligatures are explicitly given by the user, in form of macros (in a good calligraphic font there must be many several variants of ligatures). In that case, hyphenation patterns have to be re-written so that supplementary codes are taken into account. And if the ligatures are complicated, then we may as well forget all about hyphenation...

An example: in well-printed French texts (for example in the *Pléiade* collection) there is a ligature ‘st’. If this ligature is to be used *in all cases*, then it can be handled by the TFM file and the information TEX will read will be simply `st`. If the user wants to use a macro for inserting the ‘st’ ligature, for example a macro like `\char'234`, where `'234` is the octal position of the ligature in the font, then there is only one way of telling TEX that this code may eventually be hyphenated into ‘t’ and ‘s’: the `\discretionary{\char'234}{t-}{s}` command. But what if it is a three-letter ligature? there is no such thing like a ‘multi-discretionary’ command.

Because of this fact, there can’t be a perfect separartion of ‘input’, ‘internal’ and ‘output’ encoding, at least not in the present version of Ω. This means that some compromises have to be taken when defining the ‘output’ encoding of Ω.

1.3 Does Unicode `verbatim` make sense?

Everyone having written the least text on TEX or on programming in general was forced to use the concept of `verbatim` typesetting to express the fact that certain blocks of text (ranging from a single character to many pages) are supposed to be programming code.

What’s so important about computer code typesetting? of course the fact that every piece of information counts: for example, blank spaces and carriage returns are significant; but most importantly the fact that *each letter or symbol represents a single code in some encoding*. This seems natural for information interchange encodings: after all, in a computer listing we must be able to display the information we interchange, without breaking the sacred principles of `verbatim` typesetting. This is what Don Knuth has done via the CMTT family of fonts (by the way, Computer Modern is the only font family in the world featuring typewriter slanted *and* italics!).

It sounds good, but is it still feasible for Unicode? contextual writing systems with different writing directions are not exactly what we are used to see in computer listings... How do you write a backslash or a vertical bar in Berber, where these signs are letters? Should Arabic `verbatim` letters be connected or separated? What about Khmer consonantic complexes? After all, it *is* their everyday writing system, why shouldn’t they use it in, let’s say, computer code comments?

There is one important rule of `verbatim` typesetting that we haven’t mentioned yet: *glyph uniqueness*. Don Knuth and other typewriter font designers have done a great job when designing glyphs like `0` vs. `O`, `I` vs. `l` vs. `1`, etc. so that every glyph clearly represents a sole code. But what about ‘`H`’? is it a Latin ‘H’, a Greek eta or a Cyrillic en? a *real* `verbatim` font should be able to distinguish them—although the author has no idea how to do it, without introducing even more ambiguity and strongly affecting lisibility...

Maybe it is time to change the definition of `verbatim` typesetting (if you can’t win a game, change its rules[1]). In any case, it is clear that the so called ‘typewriter’ font style needs special treatment. In section 3 we discuss two different approaches of `typewriter` style: “typesetting terminal” and “commercial typewriter”.

[1]See also James T. Kirk’s solution of the Kobayashi Maru problem

2 UT

In the previous section we presented some problems involved in the project of adapting TEX/Ω to Unicode. Follows a global overview of a 'Unicode-like' Ω output encoding and the description of the latest results.

As mentioned before Unicode, is not intended for typography. Hence, a font coded in strict Unicode will give a relatively poor result when used as TEX output font. What is needed to make a typographically useful font, out of one encoded in strict Unicode?

First of all, it is clear that such a font should provide *all* Unicode characters. Other glyphs needed are

1. ligatures;
2. glyph variants;
3. specially adapted characters with diacritics: often a letter with diacritic is drawn somehow differently than just the combination of the standalone letter and the standalone diacritic;
4. letters or letterlike symbols omitted from Unicode for various reasons.

Contrarily to 8-bit font encodings where it was hard to decide which glyph should be included, and which should be left out (256 positions are quickly filled), in Unicode there is enough empty space to fill with every kind of glyph. Hence, we can say that, at least concerning the number and variety of glyphs, the only limit is our imagination. Unfortunately this is not so when we deal with TEXnical constraints.

For example, the morally correct way to fill the Unicode encoding with new glyphs would be to use the private zone; in other words, theory wants us to encode a TEX output font in Unicode and to add the 'typographical' codes in the private zone (`0xE000`–`0xFFFE`). But the size of a TFM file is proportional to the number of positions in the font, and not to the number of characters *actually used*. Such a font would have at least `0xE000` = 57,344 positions, which is an enormous figure, just to be able to include the slightest 'private' glyph.

The reader will agree that for practical reasons this is impossible. We have to bypass the strict Unicode encoding, and define a 'Unicode-like' one. 'Unicode-like' must contain all the Unicode positions of the writing systems we need, *and* our 'private' glyphs in positions *in between*, so that the size of TFM files (in this case called XFM files) remains the smallest possible. And since we take the decision to bypass the strict appliance of an encoding, we can go even farther and optimize this encoding for use with TEX, by eliminating empty zones between Unicode character collections (see fig. 1), and by splitting UT in several parts, without loss of typographic quality [there should be no kerning pairs or font-level ligatures between elements of different parts of UT].

Here is the splitting of UT we propose:

The 'Unicode-like' TEX output encoding UT, consists of five parts:

UT1 Latin, Greek, Cyrillic, Armenian, Georgian, and miscellanea (punctuation, diacritics, etc.);

UT2 Hebrew, Arabic, Syriac;

UT3 Dingbats and non-letterlike (that is font style independent) symbols;

UT4 Indic and South-East Asian writing systems;

UT5 Unified Chinese, Japanese Kanji, Korean Hancha.

UT1 covers *alphabetic*, *non-contextual*, *left-to-right* writing systems. UT1 is intended to be the continuation of OT1 (Computer Modern encoding) and T1 (Cork encoding), it will be described in more details below. UT2 covers the *right-to-left* writing systems. The change of writing direction prohibits the use of ligatures or kerning pairs between letters from UT2 and other parts of the Unicode encoding; hence there is no need to keep them in the same TFM file. The fact of having a complete 16-bit font just for Hebrew, Arabic and Syriac, allows us to keep track of a huge amount of Arabic ligatures and Hebrew combinations of letters, vowels and Masoretic diacritics. UT4 covers the *contextual*, *left-to-right* writing systems. Since most of these writing systems derive from the same script, namely Brahmi, the rules of contextual analysis are more-or-less the same. This will facilitate their structuration as well as the writing of ΩTPs, common to several Indic scripts. UT5 is the 'big' part of Unicode, it has to be treated separately. Finally, UT3 gathers all non-letterlike symbols: symbols which are not directly dependent on the font style, such as Zapf Dingbats, smilies, graphical characters (heritated from PC screen fonts) etc.

2.1 UT1

UT1 aims to cover the following collections of characters (this Unicode lingo, a *collection of characters* stands for a set of characters belonging to the same writing system, or the same category of symbols):

1. BASIC LATIN. This is what we call plain ASCII; one should be careful not to confuse it with the encoding of Computer Modern: character `0x20` is a blank space, characters `0x22`, `0x27`, `0x60` are straight quotes: single and double, and the grave accent (and not apostrophe, closing double quotes and opening quote as in Computer Modern). Position `0x7F` is empty.

2. LATIN-1 SUPPLEMENT. The higher part of ISO Latin-1, the basic encoding of X Window. One of the most illogical and clumsy encodings worldwide: no French 'œ', character positions for diacritics and cedilla (? ! ?), mathematical operators $\pm$, $\div$, $\cdot$, $\times$, $\neg$, a useless 'ÿ' (used only in Old French) without uppercase form, etc.

3. LATIN EXTENDED-A. French 'œ', Dutch 'ij', Catalan 'l·', characters for Central Europe languages, Baltic languages, Maltese, Turkish, Esperanto, Sami, and a long 's'.

4. LATIN EXTENDED-B. Characters for African languages, Croatian digraphs, and other exotica.

5. IPA EXTENSIONS. International Phonetic Alphabet, minus the signs already found in African languages (those poor people have to use phonetic characters in their everyday life because of some decisions taken by linguists or missionaries in the last two centuries...)

6. SPACING MODIFIER LETTERS. Superscript letters or small signs looking like diacritics, but used *between* words, mostly in phonetic transcriptions.

7. COMBINING DIACRITICAL MARKS. Diacritics, diacritics, diacritics.

8. BASIC GREEK. ‘Basic’ should be understood as ‘one-accent’, the result of that unhappy spelling reform of 1980 the first author will never forgive... Note that ‘; ’ is found again in this character collection, as a Greek question mark.

9. GREEK SYMBOLS AND COPTIC. Nobody is really happy with this (rather small) collection of characters. The Coptic characters included here are only those which “differ from Greek characters”, a distinction which makes no sense since Greek and Coptic alphabets are as far (or as close?) as Latin and Cyrillic... *Passons...*

10. CYRILLIC. Characters for European and Asian languages using (or having used) the Cyrillic alphabet, characters and diacritics for average Slavonic.

11. ARMENIAN. Armenian letters and punctuation marks.

12. HEBREW (BASIC AND EXTENDED). Again, the title is misleading. ‘Extended’ Hebrew characters are not (as one may think) Masoretic diacritics, but Yiddish ligatures. This collection covers modern Hebrew, with vowels, semi-vowels and punctuation.

13. ARABIC. Although this collection is not called ‘Extended’ like the previous one, it is far more complete: characters for all languages using (or having used) the Arabic alphabet, symbols for typesetting the Qur’ān, Arabic and Persian numerals.

14. GEORGIAN. Georgian characters both in modern Georgian (Khedruli) and in an old Church alphabet (Khutsuri). The fact of having both versions of the Georgian alphabet contradicts Unicode principles: a similar choice in Western languages would be to code German Fraktur or Black English letters separately...

15. LATIN EXTENDED ADDITIONAL. Diacriticized Latin letters used mainly for transcriptions, characters for Welsh and Vietnamese. Note that this collection contains *only* combinations of letters and diacritics from collections 1–7.

16. GREEK EXTENDED. The first author would call this collection simply “Greek”, but this is a matter of taste. It contains the multi-accent characters missing from BASIC GREEK, and necessary for typesetting pre-1980 and ancient Greek texts.

17. GENERAL PUNCTUATION. If the reader thinks that we are far away from the top of the encoding, and that only highly exotic characters remain to be coded, he/she

is wrong: in this collection we have the usual everyday punctuation marks (opening and closing quotes, Gänsefüßchen, single guillemets, en & em dashes, etc.). Also a big variety of typographic blank spaces (en quad, em quad, en space, em space, three-per-em space, and so on). Is this pure information? doesn't it sound more like typography?

18. SUPERSCRIPTS AND SUBSCRIPTS. Numerals, '+', '-', '=', '(', ')', in superscript and subscript form (except those already present in ISO Latin-1).

19. CURRENCY SYMBOLS. More-or-less weird currency symbols (there is a symbol for the French Franc; the first author has shown it to many Frenchmen, nobody knew of its existence...).

20. LETTERLIKE SYMBOLS. Commercial, mathematical, physical and chemical symbols, sharing more or less the style and weight of the font ($\Re$, $\wp$, ℓ, ...)

21. NUMBER FORMS. Fractions ($\frac{1}{3}$, $\frac{2}{3}$, ... $\frac{7}{8}$ and a $\frac{1}{}$ character just in case), and Roman numerals (I to XII, L, C, D, M, five thousand and ten thousand) in uppercase and lowercase form.

22. CONTROL PICTURES. The computer codes NUL, SOH, ESC, etc. and the `verbatim` blank space ␣. Letters are condensed and placed diagonally.

23. ENCLOSED ALPHANUMERICS. Numerals 1–20 enclosed in cercles, numerals 1–20 enclosed in parentheses, numerals 1–20 followed by a period, lowercase letters enclosed in parentheses, uppercase and lowercase letters enclosed in cercles (not to confuse with black-on-white and white-on-black numerals enclosed in cercles as in the Zapf Dingbats collection of Unicode).

Since there can be kerning operations between any pair of characters in this list, it would be practical to construct a single font out of them. The encoding of this font is called UT1: the reader can find an overview on fig. 1. It takes 11 pages of 256 characters.

3 UC, a Computer Modern realization of UT1

To present the Computer Modern encoding OT1, Don Knuth used Computer Modern fonts CMR, CMTI, etc. When the extended TEX encoding T1 was defined in Cork, in 1990, there was no realization of it in Computer Modern yet, it followed in 1991: the DC fonts by Norbert Schwarz. UT1 also has to be presented (and distributed together with Ω) in form of a Computer Modern realization (even if in the meantime many people affirm that the Computer Modern typeface has been "over-used" in the world and it would be more appropriate to work on other typefaces).

We call UT1's Computer Modern realization "UC" (Unicode-like Computer Modern). UC fonts will be named using the standard DOS-aware TEX syntax:

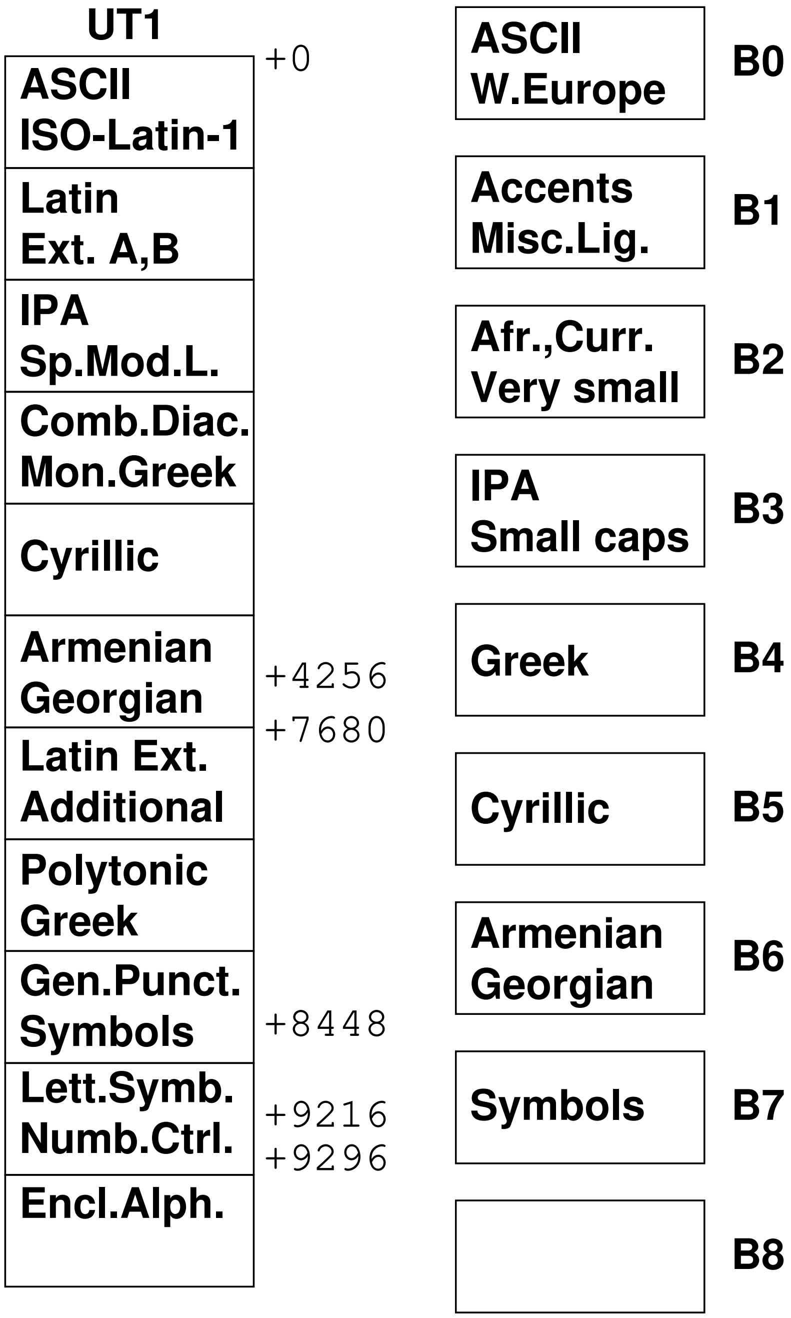


Figure 1: An overview of the UT1 font structure vs. the set of real glyph banks B0–B8

UCR	for the Roman family of UC fonts
UCR10	Unicode-like Computer Modern Roman 10 points
UCBX10	Unicode-like Computer Modern Roman Bold Extended 10 points
UCTI10	Unicode-like Computer Modern Italic 10 points
UCBXTI10	Unicode-like Computer Modern Italic Bold Extended 10 points
UCCSC10	Unicode-like Computer Modern Roman Small Caps 10 points
UCTT10	Unicode-like Computer Modern Typewriter 10 points
...	

(the authors still don't know how to express a Bold Slanted Small Caps font in this syntax and remain in the 8-letter limit; *passons...*).

There is one new style: "CT" (Commercial Typewriter). The difference between "TT" (Typesetting Terminal) and "CT" is that in the former every character has the same width: for example the Dutch ligature 'ij' is represented by a glyph of the same width as any character, made out of condensed letters 'i' and 'j'; in the latter, characters can have widths which are integral multiples of the standard width (the ligature 'ij' will have the width of a chain of two letters). Also in the former we will attempt to (optionally) differentiate glyphs representing several characters from different writing systems.

The reason for this separation is that "`typewriter`" style is used mainly for two reasons: (a) typesetting of computer code, in which case we need the uniqueness of width and the uniqueness of glyphs; and (b) commercial correspondence, where a typewriter machine is simulated: in that case our prime goal is esthetic quality.

The porting of these two font design directions (TT versus CT) to UT2 and UT4 will certainly not be trivial, and many questions remain open.

3.1 Fonts just love being virtual

Can you imagine the size of UC fonts in PK format, for—let's say—a 300dpi resolution? Enough to prevent anyone with less than a working station of using Ω ever... Not to mention the fact that students wanting to write their thesis on a personal computer at home will not really appreciate the fact of spending Megabytes of disk space just because the slightest UCR10 PK file must contain Greek, Cyrillic, African, Armenian, Georgian, and other exotic scripts...

The solution to this problem is to make UC fonts *virtual*. For every UC font there will be two files: for example, for UCR10 there will be `ucr10.tfm` and `ucr10.vf`. These will take a minimum of place, since they only contain metric information: width, height, depth, italic correction for every character, font parameters, ligatures and kerning pairs.

The bitmaps will be contained in several, relatively small PK files, to which a DVI driver will be redirected by reading the `ucr10.vf` file. A PK file not needed by the user don't even need to be on the disk, in any size. DVI drivers (and DVIcopy for devirtualization) are clever enough to ignore PK files from which no glyphs are needed, even though they appear in VF files (nevertheless, the TFM files of these "real" fonts must be on the disk: once again these are rather small).

The real fonts underlying virtual UC fonts are called B0, B1, ..., B8. Once again we use the standard DOS-aware syntax to name the different font styles and sizes: B0R10 for B0 Roman 10 points, B5BXTI5 for B5 boldface extended italic 5 points, etc. "B" stands for “Base”, and the digit shows that these are objects internal to the Ω system, with absolutely no pretention of being encoded in any standard way. The reader should regard these as *banks of glyphs*, their only purpose being to supply virtual UC fonts with the necessary glyphs to build characters.

Using virtual fonts has another tremendous advantage: many characters (for example accented letters) are constructed by combining glyphs. For example, the whole collection of characters LATIN EXTENDED ADDITIONAL can be defined virtually, by using glyphs taken from other collections of characters. To state some figures: UT1 consists of $11 \times 256 = 2816$ positions, but only $8 \times 128 = 1024$ real glyphs are actually necessary to produce them virtually!

3.2 B0

Follows the table of font B0, represented in Roman style. The METAFONT code of these characters has been written mainly by Don Knuth; some characters were added/modified by Norbert Schwarz, Jörg Knappen, and the first author.

Font B0R.

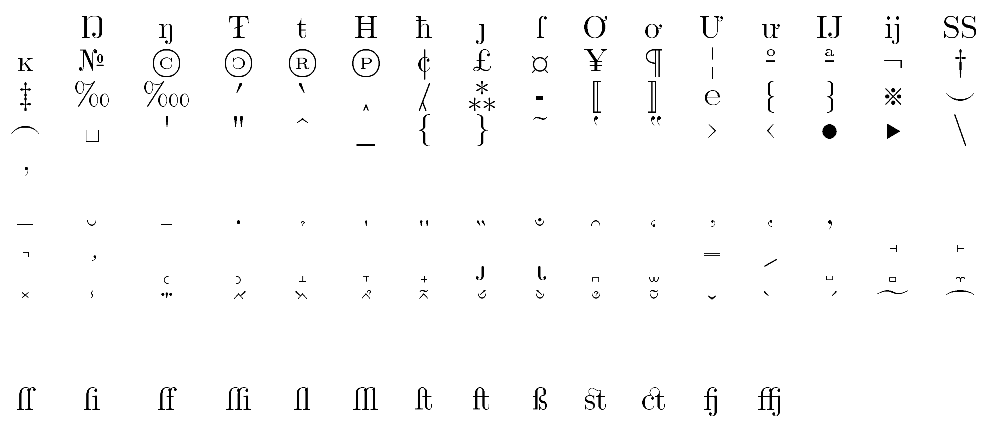
	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	§	]	¡	¿
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	«	|	»	ı	
`	´	ˆ	˜	¨	˝	˚	ˇ	đ	Đ	ð	¸	˛	Æ	æ	ß
“	Ø	„	ø	Þ	–	—	þ	Ł	ł	Œ	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ
œ	˚														

The reader will notice that this font doesn’t obey to any encoding scheme; it is not even ASCII, since instead of blank space, straight double quote, straight quote, backslash, ASCII circumflex, ASCII underline, grave accent, left and right braces and tilde, one finds œ, closing double quote, closing quote, section sign, Spanish punctuation marks, double guillemets and dotless i. The idea behind this arrangement of glyphs is that they are the most frequently used for languages of Western and Central Europe. In other words, an average text in some Latin-alphabet European language (except Maltese and Sami) will use only glyphs from B0 (combined of course through virtual font methods to construct diacriticized characters in the UT1 encoding). Since this font uses only 128 characters, this somehow brings us back to the pre-TEX3 era of 7-bit fonts: B0 fonts take *less* space than DC fonts, and produce the same (and even better) results!

3.3 B1

The table below presents the B1 font in Roman style (font B1R). The METAFONT code of these characters has been written by Don Knuth, Norbert Schwarz, Jörg Knappen, and the first author.

Font B1R.



This font is a collection of miscellaneous glyphs:

- diacritics. All Unicode diacritics (covering the COMBINING DIACRITICAL MARKS character collection), except those allready included in B0;
- more-or-less exotic letters. Sami ‘ng’ and crossed ‘t’, Maltese ‘H’ with double bar, dotless ‘j’, long ‘s’, Vietnamese vowels with hook, Dutch ‘ij’ ligature, German uppercase es-zet, Greenlandic kra;
- editorial marks. Section and paragraph sign, caret insertion point, asterism, etc.;
- letterlike symbols. ‘No’, copyright, copyleft (this is not a Unicode character, but aims to express the authors’ sympathy to the GNU project), estimated symbol (round ‘e’), per thousand, per ten thousand, dagger, double dagger, etc.;
- some currency symbols. Yen, cent, sterling pound;
- ASCII characters, missing from B0: blank space sign, straight and double quotes, ASCII circumflex, underline, braces, tilde;
- slightly more exotic punctuation than in B0. Greek second level opening quotes, single guillemets, double brackets, broken bar, bullet, black triangle, single opening Gänsefüßchen;

- and finally, ligatures. Long-s ligatures: ‘ss’, ‘si’, ‘sf’, ‘ssi’, ‘sl’, ‘ssl’, ‘st’, long-s + short-s (a variant to ß, designed by Jörg Knappen), ‘ft’, ‘fj’, ‘ffj’, and the two French ligatures short-s + t, ‘ct’.

There is still room both in the B0 and B1 tables: after all there is no absolute necessity to remain in the 7-bit range; the idea was rather to keep room enough for future additions of glyphs. An example of supplementary glyphs is character `0x2B`: it is a haček accent, flatter than the one of the B0 encoding: this is one is used for uppercase letters in Czech and Slovak.

3.4 Typewriter versions of B0 and B 1

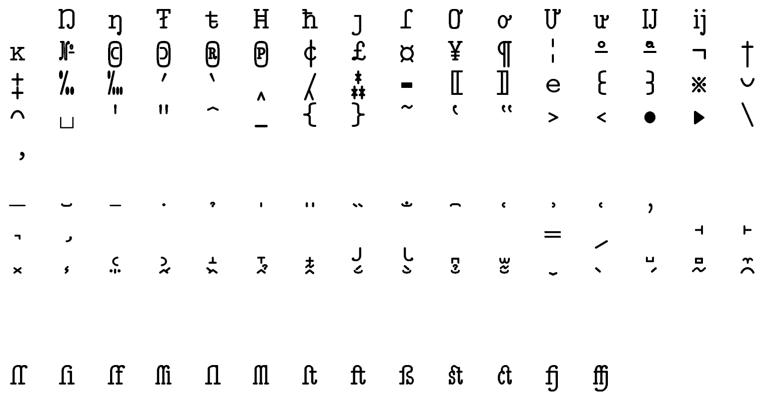
And here are the tables of the same fonts, in typewriter style (B0TT and B1TT), to illustrate some of the problems of designing typewriter-like Unicode characters:

Font B0TT.

	!	”	#	$	%	&	’	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	§	]	¡	¿
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	«	|	»	ı	
`	´	ˆ	˜	¨	˝	˚	˘	đ	Đ	ð	¸	˛	Æ	æ	ß
“	Ø	„	ø	Þ	–	—	þ	Ł	ł	Œ	ﬀ	ﬁ	ﬂ	ﬃ	ﬄ
œ	°														

The typewriter version of B0 didn’t presented any particular difficulty (except of the fact that triple ligatures ‘ffi’ and ‘ffl’ look rather clumsy). The real problems started with font B1TT:

Font B1TT.



Writing the METAFONT code for B1 was quite an adventure. First of all because of the way Computer Modern code transforms glyphs so that they all have the same width, which is far from trivial to manipulate; secondly because the original METAFONT code of most of the pre-existing characters in this font was not intended to be run with typewriter parameters (for example running the METAFONT code for the sterling pound showed some serious flaws), and thirdly and more importantly because of the inherent contradictions between the form of certain characters and the principle of keeping the same width for all of them: the most obvious example is the one of the 'per ten thousand' sign. The three little zeros under the fraction bar have no interior white space; they *cannot* have any, unless we keep the line thickness very thin. But Computer Modern typewriter has a fixed, very well defined line thickness! Another, less obvious problem arose with the copyright and other similar symbols: in TEX, when you ask for a © symbol, you get the letter 'c' enclosed in a circle, taken from a math symbol font. In a typewriter font, the copyright symbol should have the same width as other letters, so it is out of question to use that circle. And it can't even be a perfect circle, because of the dimensions of the box in which the character has to fit. The best we can have (taking under consideration also the 'superness' level of typewriter style), is a glyph similar to uppercase 'O', enclosing a tiny 'c'. This was quite hard to program.

...to be followed by sections on virtual METAFONT, examples of the use of UC, and future plans. *À suivre !*