Extending TEX for Unicode

Richard J. Kinch
6994 Pebble Beach Ct
Lake Worth FL 33467 USA
Telephone (561) 966-8400
FAX (305) 644-6978
kinch@truetex.com
http://www.truetex.com

Abstract

TEX began its "childhood" with 7-bit-encoded fonts, and has entered adolescence with 8-bit encodings such as the Cork standard. Adulthood will require TEX to embrace 16-bit encoding standards such as Unicode. Omega has debuted as a well-designed extension of the TEX formatter to accommodate Unicode, but much new work remains to extend the fonts and DVI translation that make up the bulk of a complete TEX implementation. Far more than simply doubling the width of some variables, such an extension implies a massive reorganization of many components of TEX.

We describe the many areas of development needed to bring TEX fully into multi-byte encoding. To describe and illustrate these areas, we introduce the TrueTEX® Unicode edition, which implements many of the extensions using the Windows Graphics Device Interface and TrueType scalable font technology.

Integrating TEX and Unicode

You cannot use TEX for long without discovering that character encoding is a big, messy issue in every implementation. The promise of Unicode, a 16-bit character-encoding standard [15, 14], is to clean up the mess and simplify the issues.

While Omega [5, 13] has upgraded TEX-to-DVI translation to handle Unicode [3], the fonts and DVI-to-device translators are far too entrenched in narrow encodings to be easily upgraded. This paper will develop the concepts needed to create Unicode TEX fonts and DVI translators, and exhibit our progress in the TrueTEX Unicode edition.

A fully Unicode-capable TEX brings many substantial benefits:

- TEX will work smoothly with non-TEX fonts. While TEX already has a degree of access to 8-bit PostScript and TrueType fonts, there are many limitations that Unicode can eliminate.

- TEX will eliminate the last vestiges of its deep-seated bias for the English language and US versions of multilingual platforms like Windows. It will adapt freely and instantaneously to other languages, not just in the documents produced, but in its run-time messages and user interface. This flexibility is crucial to quality software, especially to a commercial product in an international marketplace.
- With access to Unicode fonts, the natural ability of TEX to process the large character sets of the Asian continent will be realized. Methods such as the Han unification will be accessible.
- TEX will install with fewer font and driver files. Many 8-bit fonts will fit into one 16-bit font, and in systems like Windows, which treat fonts as a system-wide resource, fewer fonts are an advantage. Only one application will be needed to translate from Unicode DVI to output device.
- TEX documents will convert to other portable forms (like PDF, OpenDoc, or HTML) and will work with Windows OLE, without tricks and without pain.
- Computer Modern and other TEX fonts will be usable in non-TEX Unicode applications. The 8-bit encoding problems have broken Computer Modern on every variety of Microsoft Windows.
- When 16-bit encodings overcome the resistance of the past — and we have every reason to hope that they will — TEX will play a continuing role in software of the future, instead of becoming an antique.

Claiming these promises involves some trouble along the way, but without 16 bits to use for encoding, we will never have a solid solution.

Let us survey in the rest of this paper what is needed to achieve these various aspects of integrating TEX and Unicode.

Omega and Unicode

The goal of our work has been to create a Unicode-capable DVI translator, and to reorganize the TEX fonts into a Unicode encoding. TEX itself (that is, the formatter) already has a Unicode successor, namely Omega.

The chief advance of Omega is that it generalizes the TEX formatter to handle wider encodings. What Omega is less concerned with is the DVI format (which has always provided for wider encodings, up to 32 bits), the encoding of the existing TEX fonts, and the translation of `.dvi` files for output devices. In fact, Omega has side-stepped DVI translation altogether with its extended *xdvicopy* translation, whereby Omega operates within the old environment of 8-bit TEX fonts and the old DVI translators. Since Unicode rendering is supported on Windows but not on other popular TEX platforms (UNIX, DOS, etc.), a devotion to Omega's portability requires that Omega use the old fonts and DVI translators. Lacking any compulsion to extend the DVI translators for Unicode, the Omega project has justifiably invested most of its effort into earlier stages of the typesetting process [4, page 426].

Our Unicode TEX fonts and Unicode DVI translator, while having a natural connection to Omega, are capable of connecting TEX82 to Unicode as well. Through the mechanism of virtual fonts [11], TEX can access Unicoded fonts while using its old 8-bit encodings itself.

What's the fuss?

Wishing for 16 bits of Unicode sounds like, *hey presto*, we just widen some integer types, double some constants, and type "`make`" somewhere very, very high in a directory tree. The task is far from this simple for several reasons:

One, TEX and TEXware is full of 256-member tables which enumerate all code points. These would have to grow to 65,536 members. While Haralambous and Plaice want us to agree that this is "impossible" for practical reasons [6], they assume that we are not going to re-implement the 8-bit-encoded software for sparse arrays. Applying sparse-array techniques to manage per-character data will avoid an impossible increase in execution time and/or memory, although it will require an initial extra effort to upgrade the software.

Two, these tables have to be stored in files, and we need to carefully and deliberately extend the file formats to handle the extensions. Not only could we come up with a bad design that limits us unnecessarily in the future, but all the old TEXware has to be upgraded, and then we have to port the upgrades.

Three, we must rationalize the old ad-hoc character sets into a big union set. Just cataloging and managing this data is a large task: many tens of thousands of items, where we used to have only hundreds before. Some degree of database management tools must be applied to get the codes into a form which we can compile into software; it is not enough to just type in some array initializers here and there.

Encoding standards are necessarily incomplete or imprecise in some aspects, and none fit the TEX enterprise. While many of the Unicode math symbols were taken from TEX, many of the TEX characters are missing from Unicode. But Unicode is about the closest encoding to TEX math that we can expect from an unspecialized encoding, and with Unicode we gain a powerful connection to multilingual character sets.

Extending Computer Modern to Unicode

A "rational" encoding establishes a mapping of character names to unique integers, and this mapping does not vary from font to font. The Computer Modern fonts were not encoded rationally. For example, code `0x7b` is overloaded about 8 different characters, and character dotlessi appears in different codes in different fonts. Given an 8-bit limit on encoding, this was inevitable. But this makes for many troubles; moving up to a rational, 16-bit encoding is a clean solution.

Computer Modern is also "incomplete" in the sense that if you made a table having on one axis the list of all the specific styles (Roman, Italic, typewriter, sans serif, etc.) and on the other axis all the characters in all the fonts (A–Z, punctuation, diacritics, math symbols, etc.), the table would have lots of holes when it came to what METAFONT source exists. Commercial text fonts have all of these holes filled, or at least the regions populated in the table are rectangular. In Computer Modern the regions are randomly shaped.

Furthermore, the character axis of this (very large) imaginary table is missing many characters considered important in non-TEX encodings. For example, ANSI characters like florin, perthousand,

cent, currency, yen, brokenbar, etc., are not implemented in Computer Modern. Certain of these symbols can be unapologetically composed from existing Computer Modern symbols: a Roman multiply or divide would come from the math symbol font, trademark from the T and M of a smaller optical size, and so on. But many other characters will just have to be autographed anew in METAFONT (at least one related work is in progress [6]).

The job of extending Computer Modern to be a rational and complete set of fonts first requires that we reorganize the existing characters into a clean, 16-bit encoding. Then we are in a strong position to fill in the missing characters.

We not only want to give TEX access to 16-bit-encoded fonts, we also want the converse: non-TEX applications to have access to the Computer Modern fonts in TrueType form. This mandates adherence to the Unicode standard wherever possible, and an organized method to manage the non-Unicode characters in Computer Modern.

Here is a list of the components we consider essential to a Unicode rationalization of Computer Modern. In this list we take a different approach from Haralambous' Unicode Computer Modern project [7], which is aimed at producing virtual fonts which resolve to 8-bit `.pk` fonts from METAFONT. Our aim is a set of Unicoded TrueType fonts.

- A METAFONT-to-outline converter, a very difficult although not impossible task, as illustrated in MetaFog [9].

- A database system to treat the converted Computer Modern glyphs as atoms, for input to a TrueType font-builder.

- A database of character names which covers all the characters in the TEX fonts and in Unicode. We call the grand union TEX character set TEXUNION, in the same way that we denote the Unicode characters as the UNICODE character set. (We will use SMALL CAPS to indicate a formal set.) TEXUNION contains 1108 characters by the present inventory. Producing this database involves some work because there are no standards for TEX character names (that is, single-word alphanumeric names such as are used in PostScript encodings). The standard Unicode character descriptions are lengthy phrases instead of single words, making them unjoinable to the TEX names. For example, the Unicode standard provides the verbose entry for the code `0x00ab`, "LEFT-POINTING DOUBLE ANGLE QUOTATION MARK". The PostScript names[1] in common use come from an assortment of sources, and they exhibit inconsistencies, conflicts, and ambiguities which frustrate computation of set projections and joins.

- A database of TEX encodings, which tells which characters appear in which TEX fonts. TEX uses a gumbo of no less than 28 (!) distinct encodings (Table 1). This number may come as a surprise, but has been hidden by the web of METAFONT source files. The sum of all TEX encodings constitutes a database of 3415 name-to-code pairs, each name being taken from the 1108 members of TEXUNION. We designate this set of encodings (that is, a set of mappings of names to integers) as TEXENCOD.

  As if the miasmic fog of encoding conventions were not confused enough, small-caps fonts present still more encoding problems. They represent an axis of variation that is hardly defined in the usual set of font parameters. We must consider small-caps characters to be different from their corresponding parents. If this is not done, then there is no way to compose virtual small-caps fonts from their lowercase counterparts, because we would have no way of knowing which characters are to shrink (a jumbled set of letters and accented letters) and which do not (punctuation and all the rest). Thus, for each encoding used anywhere by a small-caps font, we must make a duplicate small-caps version (altering the lowercase character names to small-caps names) of the encoding in the list of all the encodings. Thus we have a csc2 for roman2, t1csc for t1, and so on.

  To produce these duplicate encodings, we need a rule to convert lowercase names (both letters a–z and diacritical letters) to small-caps lowercase names and back. We have simply been appending "sc" to the name (this works because there are no collisions with names that happen already to end in "sc"). For example, a small-caps letter "a" is "asc".

  Adobe has been appending "small" to their names (this often causes character names to exceed a traditional limit of 15 characters in length), as in the MacExpert encoding [2]. This is done in an irregular manner by appending to the uppercase character names (for example,

[1] There is an attempt at standardization in PostScript-style names from the Association for Font Information Interchange (AFII), but the standard is proprietary and on paper only. The names are serial numbers as opposed to abbreviated descriptions.

Table 1: TEX Single-Byte Encodings (TEXENCOD). Covers all Computer Modern, $\mathcal{AMS}$ math symbol, and Euler fonts. Each item maps a set of 128 or 256 character names to integers.

Name	Description
csc0	TEX caps and small caps ($ligs = 0$)
csc1	TEX caps and small caps ($ligs = 1$)
euex	$\mathcal{AMS}$ Euler Big Operators
eufb	$\mathcal{AMS}$ Euler Fraktur Bold†
eufm	$\mathcal{AMS}$ Euler Fraktur
eur	$\mathcal{AMS}$ Euler
eus	$\mathcal{AMS}$ Euler Script
lasy	LATEX symbols
lcircle	LATEX circles
line	LATEX lines
logo	METAFONT logo
manfnt	*TEXbook* symbols font
mathex2	TEX math extension
mathit1	TEX math italic ($ligs = 1$)
mathit2	TEX math italic ($ligs = 2$)
mathsy1	TEX math symbols ($ligs = 1$)
mathsy2	TEX math symbols ($ligs = 2$)
msam	$\mathcal{AMS}$ symbol set A
msbm	$\mathcal{AMS}$ symbol set B
roman0	TEX Roman ($ligs = 0$)
roman1	TEX Roman ($ligs = 1$)
roman2	TEX Roman ($ligs = 2$)
t1	LATEX NFSS T1 encoding
t1csc	T1 with small caps
texset0	TEX "texset" encoding ($ligs = 0$)
textit0	TEX text italic ($ligs = 0$)
textit2	TEX text italic ($ligs = 2$)
title2	TEX 1-inch capitals ($ligs = 2$)

†A superset of eufm with two extra chars

the Adobe small-caps for aacute is Aacutesmall, while ours is aacutesc); apparently someone mistook appearance for semantics. Furthermore, Adobe has a small-caps version of bare diacritics in their MacExpert encoding, although the diacritical character name is irregularly changed to an initial capital (for example, Adobe small-caps for acute is Acutesmall).

The LATEX T1 encoding, which was supposed to have been uniform for all DC fonts, also has an irrational aspect, in that the T1 encoding is overloaded when it is applied to both lowercase and small-caps fonts. Somewhere in the LATEX macros is buried something tantamount to another small-caps encoding of T1, which indicates which codes are letters or diacritics. Another related limitation of T1 encoding is the lack of small-caps accents.

A wealth of code positions does not exempt Unicode from pecksniffian absurdities. The Unicode committee will not provide encodings for a small-caps alphabet (small-caps being a matter of typography and not information content), although they provide encodings for some small-caps characters (which appear in older encoding standards subsumed by Unicode).

- A rationalization of the TEX character sets into their largest common subsets (Table 2). This represents the relations between the TEX encodings and the character subsets as organized in Knuth's METAFONT sources. The first item in each entry of Table 2 gives the TEX encoding as given in Table 1, known by the `.mf` source file used to generate the font; the remaining items are the common subsets generated via the METAFONT source files of the same names.

  The relation set forth in Table 2 is not refined for the distinctions regarding the ligature setting. Certain of Knuth's encodings appear overqualified, namely, mathex2, mathit{12}, and mathsy{12} do not vary with the *ligs* setting, although it is specified in the METAFONT driver file.[2]

  These decompositions of the various TEX encodings may be considered close to the "greatest common" subsets, although we do not require a full decomposition here. To be completely decomposed, the {012}-numbered items on the right should be further decomposed into the unnumbered common set and the various numbered differential sets. The sets on the right column of Table 2 we will use below as the set known as TEXPAGES. We have not yet made the effort to elaborate the members of each TEXPAGES set, which is needed to compute the remaining work to complete the style axes of Computer Modern.

- A database giving the mapping of TEX fonts to their encodings as known above (Table 3). The table below lists TEX font names and their encoding name; an *N* indicates a wildcard for any optical point size integer, excluding sizes of the same style already matched earlier in the table. If a new optical size for a font name is not in this table, the presumption should be

[2] Also, the comments at the top of `romsub.mf` are in error about what happens when $ligs = 2$. Apparently, no one has tried any other *ligs* setting!

Table 2: TEXENCOD Decomposition into TEXPAGES. Set romanlsc is romanl with small-caps semantics; it does not actually appear in Computer Modern. Braced digits indicate factored suffixes.

TEXENCOD Member	Covering TEXPAGES Members
csc0	accent0 cscspu greeku punct romand romanp romanu romspu romsub0 romanlsc
csc1	accent12 comlig cscspu greeku punct romand romanp romanu romspu romsub1 romanlsc
mathex2	bigdel bigop bigacc
mathit{12}	romanu itall greeku greekl italms olddig romms
mathsy{12}	calu symbol
roman0	accent0 greeku punct romand romanl romanp romanu romspl romspu romsub0
roman1	accent12 comlig greeku punct romand romanl romanp romanu romspl romspu romsub1
roman2	accent12 comlig greeku punct romand romanl romanp romanu romlig romspl romspu
texset	punct romand romanl romanp romanu tset tsetsl
textit0	accent0 greeku itald itall italp italsp punct romanu romspu romsub0
textit2	accent12 comlig greeku itald italig itall italp italsp punct romanu romspu
msam	calu asymbols
msbm	calu bsymbols xbbold
...	(... and so on for the rest ...)

that its encoding ought to be the *lowest* optical size in the table of the same name. The wild card "*" matches any suffix, such as variations on style or optical size, for names which do not match higher in the table. We designate the set {cmb10, ..., eusm*} as TEXFONTS.

- A fuzzy-matching operator which, when joining, selecting, and projecting the above databases, can resolve the redundancy, synonyms, and ambiguities in the character names and their composition. Here is an inventory of issues known to date:

Table 3: Mapping of TEXFONTS to TEXENCOD. *N* indicates an optical point size; asterisk a suffix wildcard.

TEX Font	Encoding	TEX Font	Encoding
cmb10	roman2	cmbsy5	mathsy1
cmbsy*N*	mathsy2	cmbx*N*	roman2
cmbxsl10	roman2	cmbxti10	textit2
cmcsc*N*	csc1	cmdunh10	roman2
cmex*N*	mathex2	cmff10	roman2
cmfi10	textit2	cmfib8	roman2
cminch	title2	cmitt10	textit0
cmmi5	mathit1	cmmi*N*	mathit2
cmmib5	mathit1	cmmib*N*	mathit2
cmmr10	mathit2	cmmb10	mathit2
cmr5	roman1	cmr*N*	roman2
cmsl*N*	roman2	cmsltt10	roman0
cmss*N*	roman2	cmssbx10	roman2
cmssdc10	roman2	cmssi*N*	roman2
cmssq8	roman2	cmssqi8	roman2
cmsy5	mathsy1	cmsy*N*	mathsy2
cmtcsc10	csc0	cmtex*N*	texset0
cmti*N*	textit2	cmtt*N*	roman0
cmu10	textit2	cmvtt10	roman2
lasy*	lasy	lcircle*	lcircle
line*	line	logo*	logo
manfnt	manfnt	msam10	msam
msbm10	msbm	dccsc*N*	t1csc
dctcsc*N*	t1csc	dc*	t1
euex*	euex	eufb*	eufb
eufm*	eufm	eurb*	eur
eurm*	eur	eusb*	eus
eusm*	eus		

  - bar (vertical bar) vs. brokenbar (vertical broken bar)
  - macron vs. overscore
  - minus vs. hyphen vs. endash vs. sfthyphen vs. dash
  - grave vs. quoteleft in code `0x60`
  - space (`0x40`) vs. nbspace (`0xa0`) vs. visiblespace vs. spaceopenbox vs. spaceliteral
  - rubout in code `0x7f`
  - ring vs. degree
  - dotaccent vs. periodcentered vs. middot vs. dotmath; Zdotaccent vs. Zdot, etc.
  - quotesingle vs. quoteright
  - slash vs. virgule
  - star vs. asterisk
  - oneoldstyle vs. one, etc.
  - diamondmath vs. diamond vs. lozenge

- openbullet vs. degree
- nabla vs. gradient
- cwm vs. compoundwordmark (a T1 problem) vs. zeronobreakspace
- perzero vs. zeroinferior vs. perthousandzero para perthousand
- slash vs. suppress vs. polishlslash, as in Lslash and Lsuppress
- Ng vs. Eng (and ng vs. eng)
- hungarumlaut vs. umlaut, as in Ohungarumlaut vs. Oumlaut
- dbar vs. thorn
- tilde vs. asciitilde
- tcaron transmogrifies to tcomma, et al.
- mu vs. mu1 vs. micro, code `0xb5`
- Dslash vs. Dmacron, code `0x0110`; dslash vs. dmacron, code `0x0111`
- florin (not in Unicode) vs. fscript, code `0x0192`
- fraction vs. fraction1 vs. slashmath, code `0x2215`
- circleR vs. registered, circlecopyrt vs. copyright
- arrowboth vs. arrowlongboth
- aleph vs. alef vs. alephmath
- Ifractur (eus, mathsy1, mathsy2) vs. Ifraktur (Unicode) vs. Rfractur (eus, mathsy1, mathsy2, *and* Unicode); the spelling should uniformly be "fraktur"
- smile vs. smileface vs. invsmileface vs. Unicode `0x263A` (unnamed)
- Omega vs. ohm, Omegainv vs. mho
- names not starting with letters: 0script (`0x2134`), 2bar (`0x01bb`)

We represent these items in a text file having the following format: Each line of the file gives character name synonyms, one group of synonyms per line. Any of the names on one line are synonyms, and can be freely exchanged. For example, the line "`visiblespace spaceliteral`" means that the character names `visiblespace` and `spaceliteral` are completely equivalent names. (The former was used in the TEX DC fonts [10], while the latter was the PostScript name used in the Lucida Sans Unicode TrueType font of Windows NT.)

A special case of "synonym" is the Unicode fall-back. This is a code number which is a "synonym" for TEXUNION members not in UNICODE, and is our assignment of the Unicode "private zone" codes for the misfits. For example, the line "`ff 0xf001 0xfb01`" (Microsoft fonts have an undocumented usage like this) means that the character `ff` (which is a ligature not to be found in Unicode) carries a recommended private-zone code assignment of `0xf001` or `0xfb01`. One or more such recommended codes may appear, in order of preference. In resolving a private-zone conflict, a font-building program may take the recommended codes in order until a non-conflicting code is found. Only after the recommended codes are exhausted should the program make a random private-zone assignment. Codes may be given in decimal, octal (leading `0`), or hex (leading `0x`) formats. Programs using these tables take care to distinguish character codes (which contain only hexadecimal digits if starting with `0x`, otherwise only octal digits if starting with a leading zero, otherwise only decimal digits) from names (anything else, including names which start with digits). Lucida Sans Unicode contains some names like "2500" (for code position `0x2500`); if this presents a problem we might have to prefix a letter to these names.

A name may appear in more than one synonym group, although such groups do not join within the matching algorithm. The first name in any group is the "canonical" name. The canonical name is the name which should be output by programs which compute set operations on the encoding sets. This helps to achieve a "filtered" result which does not contain troublesome synonyms. For example, if the synonym file contained the lines:

```
joseph jose yosef josephus 0xfb10
joseph joe joey
```

the names `jose`, `yosef`, and `josephus` would have fall-back code `0xfb10`, the names `joe` and `joey` would have no fall-back code, and all the names above would invariably be transformed to `joseph` on output.

The TRUETEX filter accessory program, `joincode`, performs a relational join on two font encoding sets, making a new encoding. It resolves the issues of a given synonym file according to the rules we have stated.

- A database of non-TEX encodings, which tells which characters appear in various encoding standards such as ANSI or Unicode. This list presently constitutes a database of 3523 entries from a set of 1814 characters. Some of the

Table 4: Some Non-TEX Encodings.

Name	Description
ase	Adobe PostScript "AdobeStandardEncoding", the built-in encoding of many Type 1 fonts
belleek	Belleek [8] scheme for LATEX T1 encoding on 8-bit TrueType
belleekc	Belleek with small-caps
latin1	Latin 1 (ISO 8859-1)
latin1ps	Adobe PostScript "ISOLatin1Encoding" (which is *not* ISO Latin-1!)
mac	Macintosh
macexp	Adobe PostScript "MacExpert" encoding (used by Acrobat in PDF [2]), containing ligatures, small caps, fractions, typesetting niceties
mre[3]	Adobe PostScript "MacintoshRomanEncoding" (used by Acrobat in PDF), a subset of "mac", which omits some math characters and the Apple trademark
pdfdoc	Adobe PostScript "PDFDocEncoding" (used by Acrobat in PDF), an ad-hoc encoding used in PDF outline entries, text annotations, and Info dictionary strings, consisting of a remapped set = {ase ∪ mre ∪ wae}
unicode	16-bit Unicode (Windows NT and 95, AT&T Plan 9)

Table 5: Some Encodings Used in Windows.

Name	Description
wae	Adobe "WinAnsiEncoding" (used in Acrobat in PDF), "winansi" with bullets in the .notdef positions, some semantic synonyms
winansi	Windows ANSI 8-bit (US/Western Europe code page) (Includes certain non-ANSI characters in `0x80`–`0x9f` range of codes.)
winansiu	Windows ANSI Unicode (US/Western Europe code page) (Same characters as are present in the winansi encoding, except the non-ANSI characters are in their Unicode positions.)
winmultu	Windows Multi-Lingual Unicode (Windows 95/NT) (655 characters supporting all Latin alphabets, Greek, Cyrillic, OEM screen characters.)
win*NNNN*	Windows (for code pages numbered *NNNN*)

common examples of commercial importance today are given in Tables 4 and 5. Having these sets allows us to export virtual fonts for any of the encodings represented, so we can virtualize non-Unicode, non-TEX fonts.

- A TrueType-font-builder that takes the converted outlines from various TEX fonts, organizes sets of them based on an output encoding, and builds binary TrueType fonts from the reorganized glyph data.

  A sub-tool for the font-builder incorporates a redundancy-elimination feature that allows you to specify a table listing which characters in a given TEX font may be taken from other TEX fonts without repeating a costly METAFONT glyph conversion. One example of such a redundancy is how DC fonts largely replicate the Computer Modern fonts; it would be a waste of effort to convert the glyphs twice. Another example is that many font variants are slanted versions of the upright face, and the geometric slant is easily applied to an already-converted glyph rather than slanting in METAFONT and repeating the glyph conversion. This technique is also used to compose accents and letters for "purely" accented characters (where the accent and letter do not overlap), since the MetaFog conversion is applied only to the accent part of such glyphs, allowing the redundant letter conversion to be done only once.

  Another sub-tool builds these redundancy tables by comparing the encoding tables for sets of fonts against a target font. For example, a DC font combines punctuation and symbol characters spread across several Computer Modern fonts.

Table 6: DC fonts in LaTeX (Rev. 1/95). The "funny" fonts (Fibonacci Roman, etc.) are omitted. This list is made by examining names in `*.fd` from the LaTeX distribution.

Font	5	6	7	8	9	10	12	17
dcb	•	•	•	•	•	•	•	•
dcbx	•	•	•	•	•	•	•	
dcbxsl	•	•	•	•	•	•	•	
dcbxti						•	•	•
dccsc						•	•	•
dcitt				•	•	•	•	•
dcr	•	•	•	•	•	•	•	•
dcsl				•	•	•	•	•
dcsltt				•	•	•	•	
dcss				•	•	•	•	•
dcssbx						•		
dcssdc						•		
dcssi				•	•	•	•	•
dctcsc						•	•	•
dcti			•	•	•	•	•	•
dctt				•	•	•	•	
dcu			•	•	•	•	•	•

In our system, we actually produce textual versions of the binary fonts and convert them to Type 1 and TrueType formats with separate tools. This allows a general conversion to be optimized for the ultimate binary format. For example, Type 1 glyphs require knot-pivoting, following by combing, to insert extrema tangent points. The hinting methods also differ between Type 1 and TrueType.

Once a binary version of a font is prepared, containing all the glyphs, a re-encoding tool (TrueTeX accessory program `ttf_edit` [17], which is a stack-oriented TrueType font encoding editor) must be applied to finish the font for real-world use. The re-encoding stage not only re-encodes, but can optionally adjust the metrics and kerning information. By making these aspects "afterthoughts" we can fine-tune fonts without going back into the detailed conversion process. The re-encoding stage can also upgrade any 8-bit-encoded TrueType font to an arbitrary Unicode encoding, which is important since many commercial font editors can only output 8-bit TrueType fonts.

- A notion of what TeX fonts we want to convert. If we consider the DC fonts a good target, we come up with quite a list (Table 6).

Rationalizing TeX fonts in Unicode sets

Let us consider the shuffling and dealing needed to reorganize Computer Modern into a Unicode encoding. With the luxury of thousands of code positions, we can un-do the "scattering" of characters amongst the TeX fonts. For example, the math italic set (mathit{12}) contains the regular (not italic) lowercase Greek letters. Conversely, we are going to have to scatter a few TeX fonts that happened to combine dissimilar styles into one 7-bit font, such as the math symbol fonts (mathsy{12}) which contain calligraphic capitals.

In set-theoretic terms, the rationalization task involves the following steps:

- Begin with the union of all TeX characters, the set we have called TeXunion. Remember that this is the set of character names, not the glyphs themselves.
- Partition this union set into the largest subsets which do not cross encodings. This partitioning is a set of proper subsets of TeXunion; we call this set TeXpages. For example, all the uppercase letters A–Z make such a subset. A combination of all upper- and lowercase letters A–Z and a–z do not, because the small-caps fonts do not contain the lowercase letters. These subsets are equivalent to Knuth's Computer Modern METAFONT "program" files, because this was the highest level of source file nesting in which he did not make conditional the generation of characters.
- Encode the TeX character union set for a new, universal 16-bit encoding. That is, we invent a mapping of TeXunion members to unique 16-bit integer codes. Most of the members of TeXunion appear in Unicode and so have a natural encoding already determined. For the TeXunion members not in Unicode (which includes all the small-caps letters), we shall promulgate (by fiat) assignments to the Unicode private-zone codes. We designate this subset of TeXunion as TeXpzone; this subset finds its concrete representation in the private-zone codes expressed in the character synonym table. We have the relation

  $$(\text{Unicode} \cup \text{TeXpzone}) \supset \text{TeXunion},$$

  since Unicode contains many characters not used in TeX. We can compute a mapping:

  $$\text{TeXunion} \mapsto (\text{Unicode} \cup \text{TeXpzone})$$

  by matching character names from the left to the names on the right; in this way we arrive at a Unicode code number for each TeX

character. We designate this mapping (a 16-bit number for each member of TEXUNION) as TEXERU ("TEX encodings rationalized to Unicode", rhymes with "kangaroo"). This mapping is the key result of rationalizing Computer Modern into Unicode.

- We can think again of a large table having, on one axis, the TEX fonts, on the other, the members of TEXPAGES, and bullets wherever Computer Modern implements METAFONT glyphs. This table will be sparsely and irregularly populated. The sparseness reflects the fact that the TEX fonts cover a wide range of characters, while the variations in style mostly are typographic distinctions on alphabets and punctuation; in other words, TEX provides more than a few symbols sets, in contrast to the simple ANSI/Symbol set distinction in 8-bit Windows fonts. The irregularity in this imaginary table results from the ad-hoc arrangment of TEXPAGES among TEXFONTS. The sparseness is not a deficiency, but we ought to have some goal in mind for the rational extension of Computer Modern and the other TEX fonts to populate areas of this table for the sake of regularity. Realizing this goal would require drafting of new METAFONT code and translation to outlines. This is similar to how the DC font project extended Computer Modern to the rational T1 encoding.
- At this stage we are ready to determine a list of actual Computer Modern Unicode fonts which will cover TEXFONTS. While Haralambous retained 8-bit PK fonts as the actual fonts for virtual Unicode fonts [7, 6], we will create a converse realization, namely Unicode TrueType fonts as the actual fonts for virtual CM, T1, or UT1 encodings.

  Using the imaginary table just described, we can take the union of row subsets such that columns are not overlapped. If we want to maintain stylistic uniformity within individual fonts, we merge rows subject to a personal decision as to which rows "belong together" in a stylistic sense. On the other hand, if we want to minimize the number of actual fonts and don't mind different styles in a single font, we can make a "knapsack" optimization to pack the rows as tightly as possible. (Indeed, if we discard the Unicode conformity, we could put all of Computer Modern into a single Unicode font!) In any case, this collapsing of rows involves imprecise judgments to arrive at an optimized reduced set of fonts.

  Implicit in this reduction is the factoring of wildcarded optical sizes that was introduced in TEXFONTS; we call this reduced set of fonts TEXINUNI, which will have a similar wildcarding to its parent TEXFONTS, but fewer members in parallel with the reduction.
- To produce each real Unicode font (a member of TEXINUNI), we assemble the glyphs and metrics from TEXFONTS and install them via TEXERU into each code of the mapped-to TEXINUNI member.
- We must finally produce Omega virtual fonts (that is, `.xvp` files) which will map 8-bit DVI codes from the old TEX fonts into TEXERU codes in TEXINUNI members. For this we use the TRUETEX metric exporter to generate an `.xvp` file, and XVPtoXVF to convert this to an `.xvf` file; the `.xfm` file also produced contains the same information as the METAFONT `.tfm` and may be discarded if only TEX82 is to be used for formatting.

Generating Unicode virtual fonts for non-TEX fonts

Let us consider a converse task: instead of converting single-byte-encoded Computer Modern fonts into Unicode fonts, let us assume we have a Unicode font in TrueType form, and want to make it usable with TEX or Omega. To use a font TEX (and Omega) require a `.tfm` (or `.xfm`) metric file and a `.vf` (or `.xvf`) virtual font file. The virtual font is necessary only if a remapped encoding or composition is needed (usually the case).

To generate metric and virtual font files for Unicode fonts in Windows, TRUETEX provides a `File + Export Metrics` item which takes the user through several steps which illustrate the elements of such a translation:

- First, the user selects the font from the Windows standard font-selection dialog. For example, in Windows, standard fonts include Arial (a Helvetica clone), Courier, and Times New Roman (a Times Roman clone), together with their bold and/or italic variations. Windows will also install other TrueType fonts or (with Adobe Type Manager) Type 1 fonts. After the user selects a font, TRUETEX has a "font handle" with which it can access all the geometric information needed to calculate global and per-character metric quantities for the font.
- Second, the user must select names for the output `.xvp` file. Font names in Windows are verbose strings containing several words (such

as, "Times New Roman Regular (TrueType)"), while TEX insists on a single-word alphabetic name; therefore TRUETEX selects a TEX-style name for the font based on the TrueType file name (such as "times"). The metric output in this case would be a file `times.xvp`.

- Third, the user must specify the "input" and "output" encodings for the virtual font. The input encoding is the encoding of the actual font, typically ANSI or Unicode. The output encoding is the virtual encoding which the user desires to construct for TEX's point of view, and is typically a member of TEXENCOD. TRUETEX uses encodings in the form of `.cod` files (each line contains a hex code field followed by the character name field) or `.afm` (Adobe Font Metric [1]) files[4] To select an encoding, the user selects an item from a list which TRUETEX presents, each item giving a description of the encoding (for example, "TEXRoman with *ligs* = 2" corresponds to the `roman2.cod` file).

  The user can also browse for encoding files instead of selecting from the canned list. The user can edit custom encoding files (which are just text files in `afm` format), and thereby gains complete flexibility of input versus output encoding in the virtual fonts, including automatic production of composites for missing input characters. The `ttf_edit` [17] program originates `afm` encoding tables from existing TrueType fonts, allowing maximal compatibility with randomly-encoded fonts.

- User input is now complete. TRUETEX begins analysis of the information provided by forming in-memory encoding tables from the encoding files (using sparse-array techniques to manage large, sparse code ranges). TRUETEX sorts and indexes the tables for fast content-addressibility by either code or character name, assembles the global metrics in `xvp` terms for the font, and visits each input code in the font to build a table of per-character metrics and ligatures. `xvp` file building may now begin.

- For each output character name, TRUETEX determines if an exact match exists to an input name, and thus to an input code. If there is such an exact match, TRUETEX emits `xvp` commands which give the character's metrics and which re-map the TEX PUT/SET commands to the Unicode positions.

- For output characters which have no exact to input characters, TRUETEX invokes the "composition engine." If the composition engine can compose or substitute a glyph for the character, it emits the `xvp` commands for the metrics and other actions. If the composition engine is "stumped", TRUETEX emits an `xvp` comment to note that the character is unencoded.

Note that virtual fonts can use more than one input font to produce a virtual output font. This would allow, for example, a text font, an expert font (such as might contain ligatures), and a symbol font (such as might contain math symbols) to contribute to a single TEX Unicode font. Another use for this technique would be the assembling of a Unicode font from the old bit-mapped PK fonts. TRUETEX supports all of the TEX and Omega-extended virtual font mechanisms, but does not yet directly support multiple input fonts for metric export (or bit-mapped fonts, for that matter), although an expert user can merge multiple virtual-property-list files to produce such a virtual font.

Composing missing characters

The UNICODE and TEXUNION sets are disjoint, but the virtual font mechanism allows users to create virtual TEXUNION characters missing from a UNICODE font with various composition or substitution methods. TRUETEX uses this technique to create completely populated virtual fonts when the underlying TrueType fonts are missing accented characters or ligatures.

To allow for easy upgrading, the "composition engine" in TRUETEX uses a user-modifiable script in a PostScript-like language to control the composition and substitution process. By changing the script, the user can add new composition methods or substitution rules, or adapt the methods to various typographic conventions. TRUETEX, using its own mini-PostScript interpreter, interprets this script at metric-export time, which means that users who speak PostScript and know a bit of font design can customize the composer. A good script yields a much better TEX virtual font, since commercial fonts are typically missing characters that TEX considers important, and the script can fill in most of the missing pieces.

When the composition script receives control from TRUETEX, all the encoding and metric information for the font and input and output encodings are defined as PostScript arrays and dictionaries.

[4] `.afm` files need not contain metrics; they can simply define an encoding for a dummy font, using only the C, CH, and N fields of the CharMetrics table. We thus maintain compatibility with other `afm`-reading software and avoid inventing yet another file format.

Table 7: Composable Accent Characters.

Name	Position	Example
umlaut	top-center	ö
acute	top-center	ó
breve	top-center	ŏ
caron[1]	top-center	ô
cedilla	bottom-center	o̧
circumflex	top-center	ô
comma	top-right	Ľ
dieresis	top-center	ö
dotaccent	top-center	ȯ
grave	top-center	ò
hungarumlaut	top-center	ő
macron	top-center	ō
ogonek[1]	bottom-right	
period	top-center	ȯ
ring[2]	top-center	

[1]For D/d/L/l, changes to comma at top-right.
[2]Accent is not present in the Computer Modern fonts used in this portable document.

The standard composition script in TrueTEX implements the following techniques:

- Accent-plus-letter composition: if the name implies that the character is an accented letter, the script decomposes the name into the letter and accent components, and (when the letter and accent exist in the input font) uses the geometric information and typographic conventions to overlay the accent onto the letter in a virtual accented character, as shown in Table 7. Since the composer has detailed geometric information on the glyph shape, which is more elaborate than the bounding-box metrics TEX uses, it can do a careful job of placing accents.
- Ligature composition from sequence of letters: A ligature character (not to be confused with the ligature rules of the exported TEX metrics, a different topic) such as the T1 character "SS" will not usually exist in a TrueType font. Code positions for ligatures are not part of the Unicode standard,[5] so even common ligatures are often not present. The composition script forms these by concatenating the component letters within the bounding box of the TEX character. This is applied to the ligatures: ff, fi, fl, ffi, ffl, IJ, ij, SS, Æ, æ, Œ, and œ.
- Remapping of certain names: The synonym table shows the problems of character names which are not standardized. An ad-hoc section of the composition script fixes up any ambiguities by recognizing ambiguous names and making the appropriate substitutions.
- Future extensions: Several more exotic composition methods are possible for an upgraded composition script. A novel idea is an "emergency fall-back" character generator: as a last resort for a missing character, the script could have a table of low-resolution renderings for all of TeXunion, consisting of (for example) an 8 × 16 dot-matrix or a plotter-style stick font; virtual font commands would render these with rules. In this method we could render any TEX document in a crude but accurate fashion without any fonts or `special`'s at all![6]

  Another idea would use the ability of virtual fonts to call upon the TEX `\special` command. A Bézier curve `special` could draw and fill glyphs without the need for operating system support for fonts. This would not be efficient, and hinting would be missing on low-resolution devices, but it would place all the scalable font information in an XVF file.

Projecting ligature rules into TrueType fonts

The TrueType fonts in Windows do not supply any ligature rules such as are contained in Computer Modern. To export metrics containing the usual TEX ligature rules, TrueTEX considers the rules in Table 8 when exporting the global `vpl` (`xvp`) metrics, when the target ligatures exist in the font, or when the ligatures can be produced by the composition engine.

Supporting metric export formats

TrueTEX supports both `.vpl` (TEX Virtual Property List) and `.xvp` (Omega Extended Virtual Property List) file formats when exporting font metrics. This is more than merely a variation in format; when exporting to `.vpl` format, the encodings are truncated to 8 bits, so that the composition process for missing characters will likely be more intensive. TEXware programs VPtoVF and XVPtoXVF translate the property list files to their respective `.tfm`, `.xfm`, `.vf`, and `.xvf` binary formats for use with

[5] Unicode does contain ligatures which are phonetic letters in certain languages, but this does not include typographic ligatures such as the f-ligatures

[6] This could solve the problem of rendering TEX documents in HTML browsers.

Table 8: Ligature Rules Applied to Exported Fonts.

First	Second	Result	Description
Dashes			
hyphen	hyphen	endash	`--` to endash
endash	hyphen	emdash	endash`-` to emdash
Shortcuts to national symbols			
comma	comma	quotedblbase	`,,` to quotedblbase
less	less	guillemotleft	`<<` to left guillemot
greater	greater	guillemotright	`>>` to right guillemot
exclam	quoteleft	exclamdown	`!‘` to exclamdown
question	quoteleft	questiondown	`?‘` to questiondown
F-ligatures			
f	f	ff	ff to ff
f	i	fi	fi to fi
f	l	fl	fl to fl
ff	i	ffi	ffi to ffi
ff	l	ffl	ffl to ffl
Paired single quotes to double quotes			
quoteleft	quoteleft	quotedblleft	‘‘ to quotedblleft
quoteright	quoteright	quotedblright	’’ to quotedblright

TEX and Omega. TrueTEX launches the appropriate translator after the property-list export is complete.

TrueTEX metric export also supports the older `.pl` (TEX Property List) metric file format, and the companion program PLtoTF, should it be needed for use with older TEX software. In this case the user can specify only an input font encoding, and the property list reflects this encoding as applied to the TrueType font selected, without virtual remapping.

While exporting `.xvp` files will connect the TrueTEX previewer to Windows' Unicode fonts, the TEX82 formatter requires `.tfm` metric files, not Omega `.xfm` files. If the output font has an 8-bit encoding, the resulting virtual font is nevertheless compatible with the original TEX formatter's 8-bit character codes, and will not require the Omega formatter. To create a `.tfm` file for such a font, a TrueTEX filter program `xvptovpl` truncates the virtual codes in the `.xvp` file and produces a truncated `.vpl` file, and via VPtoVF, a `.tfm` file for use with TEX. The `.vf` file created in this process is discarded, since it does not properly map the 8-bit characters to Unicode. The `.xfm` and `.tfm` files produced by this process will contain the same information; the `.xfm` format is needed only if Omega is to be used. TrueTEX uses the `.xvf` file to map the 8-bit TEX characters to Unicode positions.

Implementing sparse metric tables

In implementing programs which use metric data, we must take care to apply sparse-matrix techniques to avoid enormous memory demands from nearly-empty font-metric tables. Sixteen-bit encoded fonts are typically sparsely populated. For example, the Windows NT text fonts contain about 650 characters each; most codes are in the range `0`–`0x2ff`, with some symbols in the `0x2000` vicinity and a few odd characters in the private zone at `0xf001` or `0xfb01`. We would expect such a segmented locality in a typical font.

One technique is to use a segmented table with binary-search lookup; this is close to the method used in TrueType fonts. A hash table for the two-byte keys may be used instead of the binary search. Segmented tables will require the least storage, at the expense of a possible hashing performance problem in the event of degenerate tables. Since all Unicode-capable operating systems are advanced enough to support virtual memory, the performance risk does not justify the memory savings.

TrueTEX uses a 2-level pointer technique: metrics for a 16-bit code table consist of a table of 256 pointers to metric tables with 256 entries each. In this way a typical font having perhaps 6 or 7 contiguous code populations has very little wasted space. The worst-case and best-case performance are both acceptable. The lookup time is accelerated

by avoiding a null-pointer test by pointing unencoded pages to a dummy table of zero-width metrics.

A time-bomb of a problem looms with Omega's `.xfm` format [12], which recklessly ignores any sparse-array issues. An `.xfm` file, like the older `.tfm` format, keeps an unpacked *char_info* array which spans the smallest to largest codes (that is, the interval $[bc, ec]$). Since the Unicode private-zone population typically extends through `0xfb00`, practically all fonts will have a *char_info* of about 126 KB, almost all wasted space. This will result in a typical `.xfm` file being 130 KB, instead of about 4 KB that a simple sparse-array technique would provide. It is imperative that we upgrade the `.xfm` format and `xfm`-reading programs to better handle sparse encodings.[7]

Summary

Let us review the areas of development needed to extend all of TEX to Unicode:

- Extending the `.tfm` file format and its run-time forms to large, sparsely populated fonts, without sacrificing backward compatibility and without exploding file lengths. We have seen that the `.xfm` format can represent the information, although it needs improvement for sparsely populated fonts.
- Extending Computer Modern and other metafonts to fully populate the appropriate Unicode positions. A complete Unicode text font requires about 500 symbols. While it is unlikely that all the styles of Computer Modern will receive the attention to fill the tables completely, we can at least insert legible placeholders.
- Creating a formal database of TEX character names, joinable to the Unicode official names. Some standard is necessary for any development, and there is no reason to favor anyone's favorite names. What is crucial is that the registry be initiated now, so that TEX software authors have an early start on making interchangeable fonts and documents. While TEX users cannot themselves dictate the standard Unicode names, we can at least make a stand-in version for our own use (since none seem to exist at present), and if an acceptable set of Unicode names comes along, we can adopt it later.
- Creating a formal database of all 28 TEX font encodings and their 31 greatest common subsets, joinable to Unicode and other encoding standards. The TRUETEX tables are available to all for examination and use [16]. Once these are improved by public usage and scrutiny, they should be adopted as a formal standard for TEX.
- Identifying and assigning non-Unicode TEX character names to the Unicode private zone, thereby promoting inter-operability of Unicode TEX implementations. These should be concretely represented in a synonym table, which also needs to be published. There are many potential conflicts, and taking counsel from as many different users of TEX as possible is the only way to maximize the compatibility of the result. The Omega project has already started to stake out claims on the virgin Unicode real estate [4, Table 4, page 425] for - -Babel. There are no doubt synonyms and ambiguities outside our own experience; one can only hope there is sufficient room for all interested parties.
- Extending DVI translators to accommodate the extended `.tfm`, `.vf`, and `.dvi` formats, including sparse-array techniques for efficient run-time performance. The Omega project has issued this call to "DVI-ware developers" [4, page 426, Conclusion], although with surprising aplomb for the implications. We hereby respond with our implementation in TRUETEX, and invite others to build on our experience.
- Promulgating the ongoing TEXERU, the TEX-in-Unicode mapping, based on a seasoned registry. An ongoing authority for additions and corrections will be vital. This authority will be responsible for registering new TEX character names and avoiding Unicode conflicts.
- Changing the plain TEX and LaTEX macros to accommodate the 16-bit encoding extensions, while maintaining backward compatibility from a single source. This is a tall order, and one we have not touched.
- Extending `\special` handling for 16-bit character sets. Now we open up the carousing command of TEX to a whole new vista of revelry, with the gift of tongues. This is another item that we shall put off for now.
- Implementing TEX and DVI translation user interfaces in selectable languages. While Omega *processes* in Unicode, it *talks* to the user in the old 8-bit fashion. Perhaps it is a bit much to expect a Web2C TEX change file to incorporate *wchar_t* and other Unicode constructs of the C programming language. But DVI translators

[7] The `.vf` and `.xvf` virtual font formats have always packed sparse per-character information, so they need no such attention.

for Windows NT and other Unicode-capable platforms should have this designed in from the start. A properly designed application can be reimplemented for another language by any non-programmer who knows the application and can translate the messages; no programming or recompilation is required.

- Establishing provisions for orderly extension of the fonts and character sets, so that new TEX fonts, characters, and encodings may be incorporated into later versions. Having made the effort to retrofit METAFONT output for an altogether different way of encoding, we would hope that font designers recognize the shortcomings of the 7- and 8-bit encodings and keep the promises of Unicode in mind.
- Rationalizing the TEX fonts into orthogonal styles and weights (such as `cmmb10` and `cmmr10`, for example). As TEX users we don't care about this, but if Computer Modern is to be accepted in non-TEX applications, the style axes will have to be fully varied and populated along the conventional ranges.
- Providing a means for creating virtual fonts for non-TEX Unicode fonts in TrueType or Type 1 format. Although this capability is available now only in the commercial TrueTEX Unicode edition, a new TEXware stand-alone tool could interpret TrueType font files (or whatever typeface technology is supporting Unicode rendering) and join the encoding and other information into an `.xvp` file.

References

[1] Adobe Systems Incorporated. *Portable Document Format Reference Manual*. Reading, Mass.: Addison-Wesley, 1993.

[2] Adobe Systems Incorporated. *Adobe Font Metrics (AFM) File Format Specification*, Version 4.1, October 1995. [Published in PDF and PostScript form at `ftp://ftp.adobe.com`.]

[3] Fairbairns, Robin. "Omega— Why Bother with Unicode?" *TUGboat* 16,3 (1995), pages 325–328.

[4] Haralambous, Yannis, John Plaice, and Johannes Braams "Never Again Active Characters! - -Babel." *TUGboat* 16,4 (1995), pages 418–427.

[5] Haralambous, Yannis. "- , a TEX Extension Including Unicode and Featuring Lex-like Filtering Processes." *Proceedings of the Eighth European TEX Conference*, Gdańsk, Poland, 1994, pages 153–166. [There is an Omega Web page at `http://www.ens.fr/omega`. See also the Omega FTP site [12].]

[6] Haralambous, Yannis, and John Plaice. "- + Virtual METAFONT = Unicode + Typography (First Draft)." `ftp://nef.ens.fr/pub/tex/yannis/omega/cernomeg.ps.gz`, January 1996.

[7] Haralambous, Yannis. "The Unicode Computer Modern Project." [A document link on the Omega Web page [5].]

[8] Kinch, Richard. "Belleek: TEX Virtual T1-Encoded Fonts for Windows TrueType." Available on the author's Web site as a LATEX document in `belleek.zip`. [The Belleek software is an implementation of T1-encoded fonts using TrueType scaling technology under Microsoft Windows. Belleek consists of TrueType fonts which render elements of METAFONT glyphs in scalable form, plus TEX virtual fonts which remap and compose T1 characters from these elements.]

[9] Kinch, Richard. "MetaFog: Converting METAFONT Shapes to Contours." *TUGboat* 16,3 (1995), pages 233–243.

[10] Knappen, Jörg. "The European Computer Modern Fonts." CTAN file: `tex-archive/fonts/dc/mf/dcdoc.tex`.

[11] Knuth, Donald. "Virtual Fonts: More Fun for Grand Wizards." *TUGboat* 11,1 (1990), pages 13 – 24.

[12] Plaice, John, and Yannis Haralambous. "Draft Documentation for the - System." `ftp://nef.ens.fr/pub/tex/yannis/omega/first.tex`, February 1995. [See also the Omega Web page [5].]

[13] Plaice, John. "Progress in the - Project." *TUGboat* 15,3 (1994), pages 320 – 324.

[14] The Unicode Consortium, Inc. `http://www.unicode.org`. [This site holds files listing codes and descriptive phrases. There are no sample character images, and there are no PostScript names. A monograph gives paper images [15].]

[15] Addison-Wesley Publishing Company. *The Unicode Standard: Worldwide Character Encoding, Version 1.0*, Volumes I and II.

[16] The encodings (consisting at present of 46 encoding sets containing over 9000 pairs, represented in `.afm` format) herein listed in Tables 1, 4, and 5, are available via the author's Web site.

[17] The programs `ttf_edit` and `joincode` for DOS, Windows, and Linux are available via the author's Web site.