TUGboat

Volume 17, Number 2 / June 1996
1996 Annual Meeting Proceedings

Opening Address

Fonts

Encoding and Multilingual Support

TeX Systems

Graphics

News & Announcements

TUG Business

Advertisements

TeX Users Group

Memberships and Subscriptions

TUGboat (ISSN 0896-3207) is published quarterly by the TeX Users Group, Flood Building, 870 Market Street, #801; San Francisco, CA 94102, U.S.A.

1996 dues for individual members are as follows:

- Ordinary members: $55
- Students: $35

Membership in the TeX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in the annual election.

TUGboat subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. Subscription rates: $70 a year, including air mail delivery.

Periodicals postage paid at San Francisco, CA, and additional mailing offices. Postmaster: Send address changes to *TUGboat*, TeX Users Group, 1850 Union Street, #1637, San Francisco, CA 94123, U.S.A.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group. For further information, contact the TUG office.

Printed in U.S.A.

Addresses

All correspondence,
payments, parcels,
etc.
TeX Users Group
1850 Union Street, #1637
San Francisco,
CA 94123 USA

If you are visiting:
TeX Users Group
Flood Building
870 Market Street, #801
San Francisco,
CA 94102, USA

Telephone

+1 415 982-8449

Fax

+1 415 982-8559

Electronic Mail

(Internet)

General correspondence:
`TUG@tug.org`

Submissions to *TUGboat*:
`TUGboat@AMS.org`

1996 Annual Meeting Proceedings

TeX Users Group
Seventeenth Annual Meeting
Dubna, Russia, July 28–August 2, 1996

TUGboat

COMMUNICATIONS OF THE TeX USERS GROUP

TUGboat Editor	Barbara Beeton
Proceedings Editors	Mimi Burbank
	Christina Thiele

Volume 17, Number 2 • June 1996
San Francisco • California • U.S.A.

Production Notes

What's different this year?

My goodness — much! This is the second annual TUG meeting to be held outside North America, and the first meeting that takes place in a country with a non-Latin alphabet.

This is also the first (and probably the last) year to attempt to present the Proceedings issue of *TUGboat* before the actual meeting. We have had varying success in this regard, and will most likely return to our former schedule in 1997, if only because a pre-conference publication schedule does not allow authors and editors sufficient time to properly fine-tune the final version of a text. As a result of the delay in receiving articles from some authors, there are articles in this issue which have not been reviewed. Russian typographic styles differ from those of *TUGboat*, and these differences may be noted in a variety of articles in this issue.

Many of our speakers present material that is actually "work-in-progress" and is in some cases dependent upon discussions and input from others in a particular area of expertise who attend the annual meetings. Therefore, some of the material to be presented at the conference in Dubna will be appearing in future issues of *TUGboat*. Look in the next issue for a more detailed report of the TUG'96 Conference in Dubna and list of participants.

Of the total number of articles submitted, fourteen were by Russian authors, thirrteen were by authors whose primary language was *not* English, and two articles were submitted by North American authors — surely a tribute to the international flavor of TUG.[1]

Macros

During the three-month process of editing these proceedings, I've had the opportunity to use at least three different versions of LaTeX 2_ε macros. Look in the next issue for an article by Robin Fairbairns on the current state of the *TUGboat* style files.

Only five articles were submitted as `plain` TeX source; the others were submitted as LaTeX 2_ε.

Fonts

A large area of focus at the TUG'96 meeting will be on "languages", "encoding" and "fonts". The issue is set primarily in Computer Modern (or DC, version 1.3) fonts, using Malyshev's BaKoMa PostScript Type 1 versions.[2] The Ω article by Haralambous necessitated the creation of proper `.tfm` files from `.afm` files provided by the author. The article by O. Lapko used special Cyrillic fonts, a result of the ongoing Russian Cyrillic font project. The `wncyr` fonts developed at the University of Washington and distributed by the American Mathematical Society were also widely used in this issue.

Owing to the wide variation in Cyrillic fonts (the two mentioned above are far from being the only ones in existence) and methods for using them, many of the Cyrillic examples were included in the form of PostScript graphics prepared by the authors, so that additional fonts would not have to be installed and incorporated into the *TUGboat* styles for just one use.

Output

The editor will have to confess to many problems during the editing and production of this issue and any remaining errors are mine (MB). Without help from my co-editor, Christina Thiele, and all of the production team members, final output would have been impossible. Final output was prepared at SCRI on an IBM RS6000 running AIX, using the *Web2C* implementation of TeX. Output was printed on a QMS 680 print system at 600 dpi.

◇ Mimi Burbank
Supercomputer Computations
Research Institute
Florida State University
Tallahassee, FL 32306–4502
`mimi@scri.fsu.edu`

◇ Christina Thiele
15 Wiltshire Circle
Nepean, Ontario
K2J 4K9, Canada
`cthiele@ccs.carleton.ca`

[1] What an experience for someone whose first language is *'merkan* and who speaks English as a 'second language'.

[2] Jörge Knappen's article on page 99 required the upgrade from 1.2 to 1.3 for this issue.

Opening Words by the President

Michel Goossens
CERN, Geneva, Switzerland
goossens@cern.ch

A lot has happened in those twelve months since the last TUG Conference (the 16th) in St. Petersburg (Florida). We now know that the revolution in the area of electronic documents is here to stay. It becomes more and more commonplace to find individuals connecting from home computers to the Internet, and even in Europe "going global" starts to become affordable, with most PTT's now offering ISDN lines at prices comparable to normal telephone connections and many cable operators providing Internet services.

TeX users worldwide are finally beginning to profit fully from these electronic wonders, and can now download everything they need from a CTAN site via the Internet, or put one of several CD-ROMs which appeared over the last twelve months (one for MS-DOS by NTG, one containing the CTAN archives by DANTE, and a TDS-based plug-and-play one for Unix by TUG, GUTenberg and UK-TUG) in their CD-drive. Efforts to regularly update these CD-ROMs and extending the target domain to Microsoft Windows (NT and 95), and Macintosh are already underway, so that I have good hopes that dealing with TeX will one day become almost as simple as running Word, WordPerfect, or other easy-to-install commercial products. At the same time translation programs between LaTeX sources and various electronic hyper-formats, such as HTML, Acrobat, have been further improved. It is thus fair to say that LaTeX users are certainly not the worst placed to fully profit in an almost effortless manner from both the typographic excellence of the TeX engine and the easy integration of their documents in the global information hyperhighway. Therefore, I would like to thank all individuals who have worked hard to develop these important tools and I can only hope that they will contribute to make TeX better known in the PC commodity market, where the action in the world of electronic publishing (and computing in general) will be more and more concentrated in the future.

I am writing these words a fews weeks before the start of TUG'96, the 17th TUG Annual Meeting, taking place in the Joint Institute for Nuclear Research (JINR) in Dubna (Russia). It is only the second time that TUG's Annual Meeting has taken place outside of North America (the first time was in 1993, in Aston, Birmingham, United Kingdom). Moreover, this time we are moving beyond the English-speaking world altogether, as we are guests in the heart of Russia, the largest country in the world, spanning eleven time zones, and where the majority of the 150 million inhabitants speak Russian, which is written using the non-Latin Cyrillic alphabet. I sincerely thank CyrTUG for the invitation to come to Russia, and JINR for helping with the technical organization of the Conference. It shows that TeX is truly international and knows no borders, and that it is an ideal vehicle to promote friendship, and foster cultural exchanges and scientific collaboration.

The subject area covered by the papers at this Conference shows the diversity of the TeX culture in the world, more particularly in Central and Eastern Europe. When reading the articles, one may sometimes be struck by a "funny" or "unexpected" not-very-English-sounding expression, but this merely shows the true richness of all contributions. It underlines the fact that various approaches exist to attain a certain goal, and reading about these solutions often opens up new horizons. And then there are the "classics", like Omega, the `dc` fonts, "Blue", whose steady improvements we have been following over the years, as well as the efforts of the LaTeX3-team providing us each semester with a more robust LaTeX 2_ε, and the promises of eTeX, as announced at TUG'95. All this makes it clear that TeX is more alive than ever, and looking for a seat on the front row when Unicode, hypersurfing, and virtual (ir)reality take over the world.

Let me conclude by expressing my gratitude to the *TUGboat* Production Team, especially Mimi and Christina, for their tremendous effort on the present proceedings. I also want to thank all authors, for their continued support by communicating their work to the wider TeX community, thus increasing the pool of TeX-pertise available to everybody. And, last but not least, let me mention the support of DANTE, GUTenberg, NTG, and UKTUG, who donated funds to the TUG Bursary or otherwise contributed in kind to the Conference.

Cyrillic Alphabets

Karel Píška
Institute of Physics, Academy of Sciences
180 40 Prague, Czech Republic
piska@fzu.cz, piska@cern.ch
URL: http://www-hep.fzu.cz/~piska/

Abstract

A collection of Cyrillic-based language alphabets is presented. The contribution contains the data about more than 50 languages using Cyrillic script. A "Unicode-like" coded font is used for the rendering of the Cyrillic texts. The aim is to take part in creating a universal Cyrillic font for TEX and the Ω project and to further help languages using Cyrillic join the TEX community.

Introduction

Cyrillic-based alphabets are (or have been) used by nations in the Russian Federation, and a number of nations in Europe and Asia, including many nations of the former USSR now beyond the Russian border. The article aims to present a list of currently existing written languages using the Cyrillic script and having a codified literary form.

Most of the character encoding systems for Cyrillic used in Russia, Ukraine, Belarus, and also the UCS/Unicode [5] standard are based on the Russian alphabet. They contain a continuous ordered code sequence only for Russian letters. Other characters are non-standard, they are missing or they are coded "accidentally". I will call these characters "additional" (relative to the usual computer encoding standards!). Many Cyrillic alphabets were borrowed from the Russian alphabet. We can consider their "non-Russian" letters being "additional" or "new", often they were created (appended) as "new" characters. Of course, the previous assertion is not true for languages which have traditionally used Cyrillic script – Belarusian, Ukrainian, Serbian, Macedonian and Bulgarian.

One of my most important sources has been Р.С. Гиляревский and В.С. Гривнин (1960). Unfortunately this unique book may be obsolete today. I would be very grateful for corrections and remarks and also references to another sources; especially if the reader is an expert in any language. Please contact me by email. More information about languages can be found on my WWW Home Page (e.g., complete alphabetical orders). And please overlook my lack of knowledge of English.

Language Names and Codes

The ISO standard 639-2 (1993) and the *Ethnologue* base *eth* (1990) contain the English names of languages and also their three-letter codes. Another source of language names I have used is Webster's Dictionary (1989). Unfortunately the English and international terminology is not stabilized. When I began translation from Russian I could not find unambiguous names for languages in English. On the other hand, the Russian names are fixed in most cases.

One example, "адыгейский (язык)", is evident in Russian – but I have not selected the best examples from the following variants: Adyghe, Adyge, Adygey, Adygei, Adighe, Circassian, Lower Circassian, West Circassian, Kinkh, Kjkax, Cherkes. Therefore, I have decided to present one (rarely two) language name(s) in Russian and one (maximally two) name(s) in English (often selected "arbitrarily"). The ISO and *eth* language codes are also shown in the table of languages which use Cyrillic. If the first code (ISO) is not defined or the codes are different then the second code (*eth*) is presented.

Real Font

The B5 font family (borrowing from the Computer Modern) is a bank of Cyrillic glyphs corresponding to the Ω project (Haralambous and Plaice, 1995). The proposed encoding of a real 8-bit font is based on Unicode (ISO-IEC 10646-1, 1993) — more exactly, `"04xx mod "100`. Thus the character codes are well defined and standardized. This can simplify communication between authors supporting and improving the fonts.

The table of the `b5r12` font (Computer Modern Cyrillic Roman 12 point) is on the last page of the article. I repeat: the real font is only a bank of glyphs and cannot be used autonomously in a simple and effective way.

Virtual Font

A virtual font was created for the present article to enable access to Cyrillic letters using ASCII characters. For creating `.tfm` files for virtual fonts, the program VFComb (Berdnikov and Turtia, 1995) was used. It allows the definition (or redefinition), mapping, ligature and kerning data *once* for all font sizes, and then merging them with metric information of the real fonts (reading proper list files). It is necessary to mention that *every* font in TEX (real or virtual) can contain no more than 256 characters and it is complicated, or impossible, using fonts with many characters. This is a good reason for introducing Ω – the 16-bit extension of TEX. The virtual font used in this article combines a real font with Unicode-like encoding (`mod "100`), a font with alternative glyphs (located separately) and several characters from the original CM (e.g., parentheses). The way of referencing the "I's" is shown in the following example.
A segment from the `.tbf` file (input file for VFComb)

```
(LIGTABLE
   (LABEL C I)
   (LIG C 1 O 006)
   (LIG C 2 O 007)
   (LIG C 3 O 300)
   (LIG C 4 O 342)
   (LIG C 5 O 344)
   (STOP)
   (LABEL C i)
   (LIG C 1 O 022)
   (LIG C 2 O 211)
   (LIG C 4 O 343)
   (LIG C 5 O 345)
   (STOP)
   )
```

results
`'Ii' =>` Ии % "Standard" 'I'
`'I1i1' =>` Ii % Ukrainian/Belarusian 'I'
`'I2i2' =>` Її % Ukrainian 'YI'
`'I3' =>` I % Caucasian aspiration sign "палочка"
`'I4i4' =>` Й̄й̄ % Tadzhik 'I' with stress
`'I5i5' =>` Й̈й̈

Cyrillic Character Set and Unicode

The ISO/IEC 10646-1/Unicode (1993 E) covers most of the *letters* used in current *living* written languages uses a Cyrillic-alphabet (in my opinion). I would like to add the following comments:

- I have no data about other characters; for example, punctuation marks, special signs and other symbols.
- I don't present information about additional characters not in current use.
- Old Cyrillic is omitted and is not a subject of inquiry in this paper.
- Regarding the variant forms: more alternative glyphs may be stored in a font bank and then selected to depict a particular character. This problem is solvable in TEX.
- Regarding letters with diacritics: there are important differences in the three distinct applications of diacritical marks (with possible disagreement in different languages).
   1. The accented symbol denotes the distinct letter as opposed to the same symbol without an accent and it may even be positioned independently in the alphabet.
   2. An accent can be used to modify the symbols representing vowels and consonants: for example; vowels can be marked for length or nasalization, consonants can be marked for palatalization. The presence of the accent when writing is significant but unlike the above item, the combination does not constitute a new or special letter, and therefore would be alphabetized in the same position as the letter without such a diacritic.
   3. An accent is used to mark stress. These "stressed" letters are not part of the writing system but are, nevertheless, necessary for entries in dictionaries and textbooks. A few examples illustrate the use of stress marks (above, right or below):
      акце́нт, АКЦЕ′НТ
      ac′cent mark′, r **Akzẹnt**

Alphabetical Orders and Sort

The greater number of languages using Cyrillic in Russia and the former USSR have adopted words from Russian or, with modifications, in the original form (especially proper names) and their alphabets include *all* Russian letters. Not often exceptions are Ukrainian, Belarusian, Moldavian^{cyr}or Abkhazian.

Alphabetical orders of distinct languages may be different. "Additional" letters have been appended to the end or may occur in the middle of alphabets. Two letters may be located in the opposite order. And then the order of similar or even identical words in dictionaries or indexes may be different.

Examples (1960, 1990)[1] [2]

Russian	Ukrainian
Ьь < Юю < Яя	Юю < Яя < Ьь
вальс < валяться	валятися < вальс
польский < полюс	полюс < польський
сальный < салют	салют < сальний

Correspondence Cyrillic vs. Latin

Many languages now written in Cyrillic used Latin-like alphabets in the 1930s (e.g., Tatar or Kazakh). Several languages have used both Latin and Cyrillic alphabets — at the last count these included Serbo-Croatian, Kurdish, Moldavian, and Azerbaijani. Several nations are preparing projects to migrate from Cyrillic to Latin. The alphabetical orders for Cyrillic and Latin are different but I am sure it will be possible to define algorithms for automatic transliteration, use of common hyphenation patterns and compile and print texts from the one source, in either writing system, to produce for a reader the script with which she/he is familiar.

Cyrillic Letters and Symbols

The table contains the Cyrillic characters defined in the Unicode standard. Russian letters (used in most alphabets) and old Cyrillic letters and symbols are omitted in the list. Corresponding symbolic names of characters can be found in [5, 6]. It would too long to present them here.
Example: `CYRILLIC CAPITAL LETTER IO` is the Unicode name for `"0401 =>` Ё.

Explanatory notes and comments

^{cyr} The Cyrillic-alphabet languages presented here also uses other alphabets (usually Latin-like).
Languages using the following letters are unknown (to me):
1. Ӂӂ
2. for Ӳӳ I have two candidates – 2 letters undefined in Unicode:
У̃у̃(=Ӱӱ)? in Chuvash and
У́у́ (=Ӱӱ)? in Karachay-Balkar.

The confusion perhaps may be in my sources or in Unicode.

Unicode codes

`"0401`	`"0451`	Ё ё	Many languages use the letter Ёё. The list of the languages Ёё is **not** used in is shorter: Ukrainian, Bulgarian, Serbo-Croatian^{cyr}, Macedonian, Kurdish^{cyr}, Moldavian^{cyr}, Azerbaijani, Abkhazian, Abazin(?)
`"0402`	`"0452`	Ђ ђ	Serbo-Croatian^{cyr}
`"0403`	`"0453`	Ѓ ѓ	Macedonian
`"0404`	`"0454`	Є є	Ukrainian
`"0405`	`"0455`	Ѕ ѕ	Macedonian
`"0406`	`"0456`	І і	Ukrainian, Belarusian, Kazakh, Khakass, Komi (Zyrian), Komi-Permyak
`"0407`	`"0457`	Ї ї	Ukrainian
`"0408`	`"0458`	Ј ј	Serbo-Croatian^{cyr}, Macedonian, Azerbaijani, Altaic (Oirot)
`"0409`	`"0459`	Љ љ	Serbo-Croatian^{cyr}, Macedonian
`"040A`	`"045A`	Њ њ	Serbo-Croatian^{cyr}, Macedonian
`"040B`	`"045B`	Ћ ћ	Serbo-Croatian^{cyr}
`"040C`	`"045C`	Ќ ќ	Macedonian
`"040D`	`"045D`	(This position shall not be used)	
`"040E`	`"045E`	Ў ў	Belarusian, Uzbek, Dungan
`"040F`	`"045F`	Џ џ	Serbo-Croatian^{cyr}, Macedonian, Abkhazian

`"0410..`"`042F` **uppercase Russian**
`"0430..`"`044F` **lowercase Russian**
`"0460..`"`0486` **Old Cyrillic**

`"0490`	`"0491`	Ґ ґ	Ukrainian (now used again!)
`"0492`	`"0493`	Ғ ғ	Tadzhik, Uzbek, Uighur, Kazakh, Azerbaijani, Khakass,(Bashkir), (Karakalpak)
variant		Ғ ғ	Bashkir, Karakalpak
`"0494`	`"0495`	Ҕ ҕ	Yakut (Sakha), Abkhazian, Eskimo (Yuit)^{cyr}
`"0496`	`"0497`	Җ җ	Uighur, Turkmen, Tatar, Kalmyk, Dungan

[1] Referee's note: The Ukrainian Academy of Sciences changed the official order of the Ukrainian alphabet in 1991 (or thereabouts), and the soft sign is no longer the last letter of the alphabet.

[2] Author's note: Reworking and reprinting of all the dictionaries of any language will not be easy. I will keep this example to demonstrate "real life" changes.

"0498	"0499	Ҙ	ҙ	Bashkir
variant		Ҙ	ҙ	Bashkir
"049A	"049B	Қ	қ	Tadzhik, Uzbek, Uighur, Kazakh, Karakalpak, Abkhazian
variant		Қ	қ	
"049C	"049D	Ҝ	ҝ	Azerbaijani
"049E	"049F	Ҟ	ҟ	Abkhazian
"04A0	"04A1	Ҡ	ҡ	Bashkir
"04A2	"04A3	Ң	ң	Uighur, Kazakh, Turkmen, Kirghiz, Tatar, Bashkir, Khakass, Tuva (Soyot), Kalmyk, Dungan
"04A4	"04A5	Ҥ	ҥ	Altaic (Oirot), Yakut (Sakha), Mari-low
"04A6	"04A7	Ҧ	ҧ	Abkhazian
"04A8	"04A9	Ҩ	ҩ	Abkhazian
"04AA	"04AB	Ҫ	ҫ	Chuvash, Bashkir
variant		Ҫ	ҫ	Bashkir
"04AC	"04AD	Ҭ	ҭ	Abkhazian
"04AE	"04AF	Ү	ү	Uighur, Kazakh, Turkmen, Kirghiz, Azerbaijani, Tatar, Bashkir, Tuva (Soyot), Yakut (Sakha), Mongolian^{cyr}, Buryat, Kalmyk, Dungan
"04B0	"04B1	Ұ	ұ	Kazakh
"04B2	"04B3	Ҳ	ҳ	Tadzhik, Uzbek, Karakalpak, Abkhazian, Eskimo (Yuit)^{cyr}
variant		Ҳ	ҳ	
"04B4	"04B5	Ҵ	ҵ	Abkhazian
"04B6	"04B7	Ҷ	ҷ	Tadzhik, Abkhazian
"04B8	"04B9	Ҹ	ҹ	Azerbaijani
"04BA	"04BB	Һ	һ	Kurdish^{cyr}, Uighur, Kazakh, Azerbaijani, Tatar, Bashkir, Yakut (Sakha), Buryat, Kalmyk
"04BC	"04BD	Ҽ	ҽ	Abkhazian
"04BE	"04BF	Ҿ	ҿ	Abkhazian
"04C0		Ӏ		Abazin, Adyge, Kabardian-Circassian, Avar(ic), Lezgin, Lak(i), Dargwa, Tabasaran, Chechen, Ingush
"04C1	"04C2	Ӂ	ӂ	**???**
"04C3	"04C4	Ӄ	ӄ	Khanty-Vakhi, Chukcha, Eskimo (Yuit)^{cyr}, Koryak (Nymylan)
"04C5	"04C6			(This position shall not be used)
"04C7	"04C8	Ӈ	ӈ	Khanty (Ostyak), Chukcha, Eskimo (Yuit)^{cyr}, Koryak (Nymylan)
"04C9	"04CA			(This position shall not be used)
"04CB	"04CC	Ӌ	ӌ	Khakass
"04D0	"04D1	Ӑ	ӑ	Chuvash
"04D2	"04D3	Ӓ	ӓ	Mari-high, Khanty (Ostyak), (Kalmyk)
"04D4	"04D5	Ӕ	ӕ	Ossetic
"04D6	"04D7	Ӗ	ӗ	Chuvash
"04D8	"04D9	Ә	ә	Kurdish^{cyr}, Uighur, Kazakh, Turkmen, Azerbaijani, Tatar, Bashkir, Kalmyk, Khanty (Ostyak), Abkhazian, Dungan
"04DA	"04DB	Ӛ	ӛ	Khanty (Ostyak)
"04DC	"04DD	Ӝ	ӝ	Udmurt (Votyak)
"04DE	"04DF	Ӟ	ӟ	Udmurt (Votyak)
"04E0	"04E1	Ӡ	ӡ	Abkhazian
"04E2	"04E3	Ӣ	ӣ	Tadzhik
"04E4	"04E5	Ӥ	ӥ	Udmurt (Votyak)
"04E6	"04E7	Ӧ	ӧ	Kurdish^{cyr}, Altaic (Oirot), Khakass, Mari-low, Mari-high, Udmurt (Votyak), Komi (Zyrian), Komi-Permyak, Khanty-Vakhi, (Kalmyk)
"04E8	"04E9	Ө	ө	Uighur, Kazakh, Turkmen, Kirghiz, Azerbaijani, Tatar, Bashkir, Tuva (Soyot), Yakut (Sakha), Mongolian^{cyr}, Buryat, Kalmyk, Khanty (Ostyak)
"04EA	"04EB	Ӫ	ӫ	Khanty (Ostyak)
"04EC	"04ED			(This position shall not be used)
"04EE	"04EF	Ӯ	ӯ	Tadzhik
"04F0	"04F1	Ӱ	ӱ	Khakass, Mari-low, Mari-high, Khanty-Vakhi, Altaic (Oirot), (Kalmyk)
"04F2	"04F3	Ӳ	ӳ	**???**
"04F4	"04F5	Ӵ	ӵ	Udmurt (Votyak)
"04F6	"04F7			(This position shall not be used)
"04F8	"04F9	Ӹ	ӹ	Mari-high

Languages Using Cyrillic Script

The following table contains a short overview of Cyrillic-alphabet languages and their "additional" letters (according to 'usual' standard encoding systems). ISO (ISO Committee Draft 639-2, 1993)/*eth* (Ethnologue, 1990)are two different three-letter language codes. The order is "quasi-linguistic-geographical-historical" ("cognate" or "neighbouring" nations being together).

Indo-European Group		Codes	Additional characters
Slavic Languages / Славянские языки		ISO/*eth*	
• Russian	русский	`rus`	(Ёё)
• Ukrainian	украинский	`ukr`	Ґґ Єє Іі Її (’)
• Belarusian	белорусский	`bel`/*ruw*	Іі Ўў
• Bulgarian	болгарский	`bul`/*blg*	
• Serbo-Croatian^{cyr}	сербскохорватский	`src`	Ђђ Јј Љљ Њњ Ћћ Џџ
• Macedonian	македонский	`mac`/*mkj*	Ѓѓ Ѕѕ Јј Ќќ Љљ Њњ Џџ
Iranian Languages / Иранские языки			
• Ossetic	осетинский	`oss`/*ose*	Ӕӕ
○ Kurdish^{cyr}	курдский	`kur`	Әә Ӧӧ Һһ Һ’һ’ Ԛԛ Ԝԝ
• Tadzhik	таджикский	`tgk`/*pet*	Ғғ Ӣӣ Ққ Ӯӯ Ҳҳ Ҷҷ
Romance Languages / Романские языки			
○ Moldavian^{cyr}	молдавский	`mol`	

Altaic Group			
Turkic Languages / Тюркские языки			
• Uzbek	узбекский	`uzb`	Ўў Ққ Ғғ Ҳҳ
• Uighur	уйгурский	`uig`	Ққ Ңң Ғғ Үү Өө Җҗ Әә Һһ
• Kazakh	казахский	`kaz`	Әә Ғғ Ққ Ңң Өө Ұұ Үү Һһ Іі
• Turkmen	туркменский	`tuk`/*tck*	Җҗ Ңң Өө Үү Әә
• Kirghiz	киргизский	`kir`/*kdo*	Ңң Өө Үү
○ Azerbaijani	азербайджанский	`aze`	Ғғ Әә Јј Ҝҝ Өө Үү Һһ Ҹҹ ’
• Tatar	татарский	`tat`/*ttr*	Әә Өө Үү Җҗ Ңң Һһ
• Bashkir	башкирский	`bak`/*bxk*	Ғғ (=Ғғ) Ҙҙ (=Ҙҙ) Ҡҡ Ңң Өө Ҫҫ (=Çç) Үү Һһ Әә
• Karachay-Balkar	карачаево-балкарский	*krc*	(У́у́ =)Ӱӱ
• Kumyk	кумыкский	*ksk*	
• Nogay	ногайский	*nog*	
• Karakalpak	каракалпакский	`kaa`/*kac*	Ққ Ғғ (=Ғғ) Ҳҳ
• Altaic (Oirot)	алтайский	*alt*	Јј Ҥҥ Ӧӧ Ӱӱ
• Khakass	хакасский	*kjh*	Ғғ İ i (=Іі) (Нъ нъ=) Ңң Ӧӧ Ӱӱ (Чч=) Ӌӌ
• Tuva (Soyot)	тувинский	`tyv`/*tun*	Ңң Өө Үү
• Chuvash	чувашский	`chv`/*cju*	Ӑӑ Ӗӗ Ҫҫ У̃у̃(=Ӱӱ)
• Yakut (Sakha)	якутский	`sah`/*ukt*	Ҕҕ Ҥҥ Өө Һһ Үү
Mongolian Languages / Монгольские языки			
• Mongolian^{cyr}	монгольский	`mon`/*khk*	Өө Үү
• Buryat	бурятский	`bua`/*mnb*	Өө Үү Һһ
• Kalmyk	калмыцкий	*kgz*	Әә Һһ Җҗ Ңң Өө Үү

Tungusic-Manchu Languages / Тунгусо-маньчжурские языки

• Evenki (Tungus)	эвенкийский	*evn*	
• Even (Lamut)	эвенский	*eve*	
• Nanai (Gold)	нанайский	*gld*	

Uralic Group

Finno-Ugric Languages / Финно-угорьские языки

• Mari-low	марийский луговой	*mal*	Ҥҥ Ӧӧ Ӱӱ
• Mari-high	марийский горный	*mrj*	Ӓӓ Ӧӧ Ӱӱ Ӹӹ
• Mordvin-Erzya	мордовский эрзянский	*myv*	
• Mordvin-Moksha	мордовский мокшанский	*mdf*	
• Udmurt (Votyak)	удмуртский	*udm*	Ӝӝ Ӟӟ Ӥӥ Ӧӧ Ӵӵ
• Komi (Zyrian)	коми	*kpv*	Ii Ӧӧ
• Komi-Permyak	коми-пермяцкий	*koi*	Ii Ӧӧ
• Mansi (Vogul)	мансийский	*mns*	
• Khanty-Vakhi	хантыйский-ваховский	*kca*	Ӓӓ Ӄӄ Ӈӈ Ӧӧ Өө Ӫӫ Ӱӱ Әә Ӛӛ
• -Kazim	-казымский		Ӓӓ Әә Ӛӛ Ӈӈ Өө Ӫӫ
• -Shurishkar	-шуришкарский		

Samoyedic Languages / Самодийские языки

• Nenets (Yurak)	ненецкий	*yrk*	
• Selkup	селькупский	`sel`/*sak*	

Caucasian Languages / Кавказские языки

• Abkhazian	абхазский	`abk`	Ҕҕ Џџ Ҽҽ Ҿҿ Ӡӡ Ққ Ҟҟ Ҩҩ Ҧҧ Ҭҭ Ҳҳ Ҵҵ Ҷҷ Әә
• Abazin	абазинский	*abq*	I
• Adyge	адыгейский	*ady*	I
• Kabardian-Circassian	кабардино-черкесский	*kab*	I
• Avar(ic)	аварский	`ava`/*avr*	I
• Lezgin	лезгинский	`lez`	I
• Lak(i)	лакский	*lbe*	I
• Dargwa	даргинский	*dar*	I
• Tabasaran	табасаранский	*tab*	I
• Chechen	чеченский	`che`/*cjc*	I
• Ingush	ингушский	*inh*	I

Sino-Tibetan Group / Китайско-тибетские языки

• Dungan	дунганский		Әә Җҗ Ңң Ўў Үү

Paleo-Asiatic Languages / Палеоазиатские языки

• Chukcha	чукотский	*ckt*	Ӄӄ Ӈӈ ’
• Eskimo (Yuit)^{cyr}	эскимосский	*ess*	(Г’г’=) Ҕҕ (К’к’=) Ӄӄ (Н’н’=) Ӈӈ (Х’х’=) Ҳҳ ’
• Koryak (Nymylan)	корякский	*kpy*	(В’ в’) (Г’ г’) Ӄӄ Ӈӈ (К’ к’) (Н’ н’)
• Nivkh (Gilyak)	нивхский		’

Explanatory notes

○	the language is migrating (again!) to Latin writing
(Ёё)	a character not used today (particularly or entirely)
(=Ғғ)	variant not preferred
(Г’ г’=)	a former variant

b5r12 Font Table

(Computer Modern Roman 12 point)
Modern Cyrillic Part of ISO 10646-1/Unicode

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x		Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	″0x
´01x	Ј	Љ	Њ	Ћ	Ќ		Ў	Џ	
´02x	А	Б	В	Г	Д	Е	Ж	З	″1x
´03x	И	Й	К	Л	М	Н	О	П	
´04x	Р	С	Т	У	Ф	Х	Ц	Ч	″2x
´05x	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	
´06x	а	б	в	г	д	е	ж	з	″3x
´07x	и	й	к	л	м	н	о	п	
´10x	р	с	т	у	ф	х	ц	ч	″4x
´11x	ш	щ	ъ	ы	ь	э	ю	я	
´12x		ё	ђ	ѓ	є	ѕ	і	ї	″5x
´13x	ј	љ	њ	ћ	ќ		ў	џ	
´14x	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	″6x
´15x	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ			
´16x	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	″7x
´17x									
´20x									″8x
´21x									
´22x	Ґ	ґ	Ғ	ғ	Ҕ	ҕ	Җ	җ	″9x
´23x	Ҙ	ҙ	Қ	қ	Ҝ	ҝ	Ҟ	ҟ	
´24x	Ҡ	ҡ	Ң	ң	Ҥ	ҥ	Ҧ	ҧ	″Ax
´25x	Ҩ	ҩ	Ҫ	ҫ	Ҭ	ҭ	Ү	ү	
´26x	Ұ	ұ	Ҳ	ҳ	Ҵ	ҵ	Ҷ	ҷ	″Bx
´27x	Ҹ	ҹ	Һ	һ	Ҽ	ҽ	Ҿ	ҿ	
´30x	Ӏ	Ӂ	ӂ	Ӄ	ӄ			Ӈ	″Cx
´31x	ӈ			Ӌ	ӌ				
´32x	Ӑ	ӑ	Ӓ	ӓ	Ӕ	ӕ	Ӗ	ӗ	″Dx
´33x	Ә	ә	Ӛ	ӛ	Ӝ	ӝ	Ӟ	ӟ	
´34x	Ӡ	ӡ	Ӣ	ӣ	Ӥ	ӥ	Ӧ	ӧ	″Ex
´35x	Ө	ө	Ӫ	ӫ			Ӯ	ӯ	
´36x	Ӱ	ӱ	Ӳ	ӳ	Ӵ	ӵ			″Fx
´37x	Ӹ	ӹ							
	″8	″9	″A	″B	″C	″D	″E	″F	

Comments

Alternative glyphs referenced in the article (Ғғ Ққ Ҳҳ Ҙҙ Ҫҫ У́у́) are located in the additional separate font.

The old Cyrillic part `("60.."86)` is not used in the article and has not been completed.

Conclusion

I think the Cyrillic portion of the Unicode Mapping Table covers quite a good character set for most of the languages using the Cyrillic alphabet today. On the other hand UCS/Unicode does not offer a sophisticated solution for alphabetical ordering. And I am afraid that the situation with special symbols, particularities, and multiple accents for dictionaries and textbooks, etc., will be more complicated.

Acknowledgements

I would like to thank Prof. Donald E. Knuth for providing TeX, METAFONT and the Computer Modern Fonts, Michel Goossens for important documents about languages and Unicode standard, Yannis Haralambous, Olga Lapko and other authors for METAFONT sources of Cyrillic fonts, Aleksandr Berdnikov for the program VFComb (See his article on VFComb in this issue), and Mimi Burbank for help with the English text.

References

[1] ISO Committee Draft 639-2. Code for the representation of names of languages. Part 2: Alpha-3 code. International Information Centre for Terminology (INFOTERM), Wien 1993. (preliminary draft, not published)

[2] Ethnologue Database, `ftp://ftp.std.com/obi/Ethnologue/eth.Z`, 19 February 1990.

[3] Webster's Encyclopedic Unabridged Dictionary of the English language, Portland House, New York, 1989.

[4] Yannis Haralambous, John Plaice, 'Ω + Virtual METAFONT = Unicode + Typography', *Cahiers GUTenberg* n21, juin 1995.

[5] International Organization for Standardization. "Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1 : 1993, (First edition, 1993-05-01), Geneva, 1993, (Unicode).

[6] `ftp://unicode.org/pub/MappingTables/UnicodeDataCurrent.txt.Z`, 21 May 1996.

[7] Р.С. Гиляревский, В.С. Гривнин, 'Определитель языков мира по письменностям', Издательство восточной литературы, Москва 1960.

[8] A.S. Berdnikov, S.B. Turtia: VFComb - a program for design of virtual fonts. Proceedings of the Ninth European TeX Conference, Arnhem 1995.

The dc fonts 1.3: Move towards stability and completeness

Jörg Knappen
Unternehmensberatung, Barbarossaring 43, D-55118 Mainz, Allemagne
knappen@vkpmzd.kph.uni-mainz.de

Introduction

With the release 1.2 of the dc fonts and the first release of the tc fonts last year [1], a big step forward was made towards the final ec fonts. The ec fonts are designed to replace the cm fonts as the basic text fonts for LaTeX, and they will be as stable as the cm fonts are now.

The release 1.3 emphasises two points: stability and completeness. In order to achieve the first goal, there are only a few new features included in this release, and there will be only bug fixes to the next release, which will be ec fonts version 1.0

Stability

About two dozen bugs were reported after the release 1.2 of the dc fonts, some of them sleeping in the source code since the very first release in 1990, others newly introduced by new features and designs of the 1.2 release.

Most serious was a bug reported and analysed by Andreas Schwab: the metrics of several accented characters were resolution dependent. The reason was that the character dimensions were set after drawing the letter.

The fix finally applied (thanks to Bogusław Jackowski) was to compute the character twice, once in a canonical sharp mode to fix its dimensions, and a second time in printer's mode to generate the actual letter. Note that it must be possible to switch the mode on the fly to use this method — one can no longer load the current `modes.mf` 3.0 *after* `dxbase.mf`.

The fine positioning of the accents was very much improved, and I appreciate the valuable contribution by Daniel Taupin for the French letters.

A significant number of other corrections were incorporated, including the improvements Knuth made to the cm fonts since 1990 [2].

Completeness

There are some new features added to the new release. First I want to mention that there new characters added to the tc fonts: *comma*, *full stop*, and *capital cwm*. The first two were added to facilitate a macro for oldstyle digits using the text companion font in the following way:

```
\def\oldstyledigits#1{%
  {\fontencoding{TS1}\selectfont{#1}}}
```

This enables decimal separators in oldstyle numbers. There are now also italic oldstyle digits in the tc fonts, contributed by Gert-Jan Lockhorst.

1 2 3 4 5 6 7 8 9 0

Fig. 1: Italic oldstyle digits

The *capital cwm* was added, because I decided to give the compound-word mark a height of 1 ex. This adds a new functionality to this zero width invisible character: it can act as a carrier for accents placed 'between' letters, as used in the german -burg abbreviature.

b˘g

Fig. 2: German -burg abbreviature, input as `b\u\cwm g`

The *capital cwm*, of course, has the height of capital letters.

Scandinavian have asked often for a culturally correct italic æ — the design from the cm fonts aims for a maximum difference to the italic œ, but looks unusual in a Scandinavian context. Therefore, a new shape for the *æ* was added, designed to share its metric with the original design, and the two designs can be switched by setting a boolean variable in the parameter files.

æ æ

Fig. 3: New shape of the italic æ contrasted with the original one

There are many more kerning pairs added, for example 'nV' (nanovolt), 'Ay' (occurs very often in turkish proper names), 'Ad', and 'Ae'.

Ayşe

Fig. 4: Kerning for 'Ay' is added to release 1.3

Completeness also means that some missing fonts were added to the new release, including `dcxc` bold extended caps and small caps, `dcvi` variable width italic typewriter, and bold SliTeX fonts.

New font dimensions

The dc fonts used to contain some additional font dimensions, which were never documented, and some of them have turned out to be unusable at all. The issue of additional font dimensions was brought up again on the `LATEX-L` mailing list for the LaTeX3 project.

The dc fonts contain now the following 9 additional font dimensions, most of them suggested and explained by Michael Downes.

fontdimen 8 *font_cap_height*, height of capital letters (not accented)

fontdimen 9 *font_asc_height*, height of lowercase letters with ascenders

fontdimen 10 *font_acc_cap_height*, height of accented capital letters

fontdimen 11 *font_desc_depth*, depth of lowercase letters with descenders

fontdimen 12 *font_max_height*, maximum height of any glyph in the font

fontdimen 13 *font_max_depth*, maximum depth of any glyph in the font

fontdimen 14 *font_digit_width*, width of the digits contained in the font; if the digits had different widths, the maximum width over all digits is taken

fontdimen 15 *font_cap_stem*, width of the stem of the capital 'I'; can be used to determine the 'absolute boldness' of the font

fontdimen 16 *font_baselineskip*, suggested value of baselineskip to be used with the font

These new font dimensions can be read by TeX commands and employed by macros.

References

[1] Jörg Knappen, "The release 1.2 of the Cork encoded dc fonts and the text companion symbol fonts", Wietse Dol, editor, *Proc. 9th European TeX conference*, Arnhem 1995, pp. 239.

[2] Donald E. Knuth, file `em85.bug`, available from CTAN archives, in `/systems/knuth/errata/em85.bug.gz`.

A Font tables

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	`	´	ˆ	˜	¨	˝	˚	ˇ	″0x
´01x	˘	¯	˙	¸	˛	‚			
´02x			„			—	—		″1x
´03x	←	→	⁀	⁀					
´04x	ƀ				$			'	″2x
´05x			*		,	=	.	/	
´06x	0	1	2	3	4	5	6	7	″3x
´07x	8	9							
´10x									″4x
´11x						℧		◯	
´12x								Ω	″5x
´13x							↑	↓	
´14x	`		★	⚮	†				″6x
´15x					❦	⚭	♪		
´16x									″7x
´17x							~	=	
´20x	˘	ˇ	″	‶	†	‡	‖	‰	″8x
´21x	•	℃	$	¢	ƒ	₡	₩	₦	
´22x	₲	₱	₤	℞	‽	⸘	₫	™	″9x
´23x	‱	¶	฿						
´24x			¢	£	¤	¥	¦	§	″Ax
´25x	¨	©	ª		¬		®	¯	
´26x	°	±	²	³	´	µ	¶	·	″Bx
´27x		¹	º		¼	½	¾		
´32x							×		″Dx
´33x									
´36x							÷		″Fx
´37x									
	″8	″9	″A	″B	″C	″D	″E	″F	

The layout of the text companion font tcr1000. The table shows the full set implemented in release 1.3 of the dc/tc fonts, the encoding is named `TS1` within LaTeX 2ε.

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	`	´	ˆ	˜	¨	˝	˚	ˇ	˝0x
´01x	˘	¯	˙	¸	˛	‚	‹	›	
´02x	“	”	„	«	»	–	—		˝1x
´03x	₀	ı	ȷ	ff	fi	fl	ffi	ffl	
´04x	␣	!	"	#	$	%	&	’	˝2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	˝3x
´07x	8	9	:	;	<	=	>	?	
´10x	@	A	B	C	D	E	F	G	˝4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	˝5x
´13x	X	Y	Z	[	\	]	^	_	
´14x	‘	a	b	c	d	e	f	g	˝6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	˝7x
´17x	x	y	z	{	|	}	~	-	
´20x	Ă	Ą	Ć	Č	Ď	Ě	Ę	Ğ	˝8x
´21x	Ĺ	Ľ	Ł	Ń	Ň	Ŋ	Ő	Ŕ	
´22x	Ř	Ś	Š	Ş	Ť	Ţ	Ű	Ů	˝9x
´23x	Ÿ	Ź	Ž	Ż	Ĳ	İ	đ	§	
´24x	ă	ą	ć	č	ď	ě	ę	ğ	˝Ax
´25x	ĺ	ľ	ł	ń	ň	ŋ	ő	ŕ	
´26x	ř	ś	š	ş	ť	ţ	ű	ů	˝Bx
´27x	ÿ	ź	ž	ż	ĳ	¡	¿	£	
´30x	À	Á	Â	Ã	Ä	Å	Æ	Ç	˝Cx
´31x	È	É	Ê	Ë	Ì	Í	Î	Ï	
´32x	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	Œ	˝Dx
´33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	SS	
´34x	à	á	â	ã	ä	å	æ	ç	˝Ex
´35x	è	é	ê	ë	ì	í	î	ï	
´36x	ð	ñ	ò	ó	ô	õ	ö	œ	˝Fx
´37x	ø	ù	ú	û	ü	ý	þ	ß	
	˝8	˝9	˝A	˝B	˝C	˝D	˝E	˝F	

This picture shows the font dcr1000. The encoding is LaTeX 2ε's T1 encoding. The compound word mark in position ´027 is an invisible character of zero width.

	´0	´1	´2	´3	´4	´5	´6	´7	
´00x	`	´	ˆ	˜	¨	˝	˚	ˇ	˝0x
´01x	˘	¯	˙	¸	˛	‚	‹	›	
´02x	“	”	„	«	»	–	—		˝1x
´03x	₀	ı	ȷ	ff	fi	fl	ffi	ffl	
´04x	␣	!	"	#	$	%	&	’	˝2x
´05x	(	)	*	+	,	-	.	/	
´06x	0	1	2	3	4	5	6	7	˝3x
´07x	8	9	:	;	<	=	>	?	
´10x	@	A	B	C	D	E	F	G	˝4x
´11x	H	I	J	K	L	M	N	O	
´12x	P	Q	R	S	T	U	V	W	˝5x
´13x	X	Y	Z	[	\	]	^	_	
´14x	‘	a	b	c	d	e	f	g	˝6x
´15x	h	i	j	k	l	m	n	o	
´16x	p	q	r	s	t	u	v	w	˝7x
´17x	x	y	z	{	|	}	~	-	
´20x	Ɓ	Ɗ	Ɛ	Ǝ	Ƒ	Ě	Ɣ	Ħ	˝8x
´21x	Ƙ	Ɲ	Ɔ	Ń	Ʃ	Ŋ	Ʊ	Ƴ	
´22x	Ƈ	Ƥ	Š	Ṅ	N̲	Ṣ	Ʒ	Ŧ	˝9x
´23x	Ė	Ẹ	Ƭ	Ʈ	ʧ	ʄ	đ	˵	
´24x	ɓ	ɗ	ɛ	ə	ƒ	ě	ɣ	ħ	˝Ax
´25x	ƙ	ɲ	ɔ	ń	ʃ	ŋ	ʊ	ƴ	
´26x	ƈ	ƥ	š	ṅ	n̲	ṣ	ʒ	ŧ	˝Bx
´27x	ė	ẹ	ƭ	ʈ	"	¡	¿	'	
´30x	Ɩ	Ị	Ɛ̃	Ã	Ḿ	Ɔ̃	Æ	Ç	˝Cx
´31x	È	É	Ê	Ë	E̲	Ē	Ẽ	Ĩ	
´32x	Ð	Ñ	Ò	Ȯ	Ô	Õ	Ö	Œ	˝Dx
´33x	Ø	Ọ	O̲	Ō	Ǒ	Ụ	Ũ	ˏ	
´34x	ɩ	ị	ɛ̃	ã	ḿ	ɔ̃	æ	ç	˝Ex
´35x	è	é	ê	ë	e̲	ē	ẽ	ĩ	
´36x	ɖ	ñ	ò	ȯ	ô	õ	ö	œ	˝Fx
´37x	ø	ọ	o̲	ō	ǒ	ụ	ũ	ß	
	˝8	˝9	˝A	˝B	˝C	˝D	˝E	˝F	

The aFrican Computer modern font fcr10. The fc encoding is now LaTeX 2ε's T4 encoding.

TIPA: A System for Processing Phonetic Symbols in LaTeX

Fukui Rei
Department of Asian and Pacific Linguistics, Institute of Cross-Cultural Studies, Faculty of Letters,
University of Tokyo, Hongo 7-3-1, Bunkyo-ku, TOKYO 113 Japan
fkr@tooyoo.l.u-tokyo.ac.jp

Introduction

TIPA[1] is a system for processing IPA (International Phonetic Alphabet) symbols in LaTeX. It is based on TSIPA[2] but both METAFONT source codes and LaTeX macros have been thoroughly rewritten so that it can be considered as a new system.

Among many features of TIPA, the following are the new features as compared with TSIPA or any other existing systems for processing IPA symbols.

- A new 256 character encoding for phonetic symbols (`T3`), which includes all the symbols and diacritics found in the recent versions of IPA and some non-IPA symbols.
- Complete support of LaTeX 2_ε.
- Roman, slanted, bold, bold extended and sans serif font styles.
- Easy input method in the IPA environment.
- Extended macros for accents and diacritics.[3]
- A flexible system of macros for 'tone letters'.
- An optional package (`vowel.sty`) for drawing vowel diagrams.[4]
- A slightly modified set of fonts that go well when used with Times Roman and Helvetica fonts.

TIPA Encoding

Selection of symbols The selection of TIPA phonetic symbols[5] was made based on the following works.

- *Phonetic Symbol Guide* [9] (henceforth abbreviated as *PSG*).
- The official IPA charts of '49, '79, '89 and '93 versions.
- Recent articles published in the *JIPA*[6], such as "Report on the 1989 Kiel Convention" [6], "Further report on the 1989 Kiel Convention" [7], "Computer Codes for Phonetic Symbols" [3], "Council actions on revisions of the IPA" [8], etc.
- An unpublished paper by J. C. Wells: "Computer-coding the IPA: a proposed extension of SAMPA" [10].
- Popular textbooks on phonetics.

More specifically, TIPA contains all the symbols, including diacritics, defined in the '79, '89 and '93 versions of IPA. And in the case of the '49 version of IPA, which is described in the *Principles* [5], there are too many obsolete symbols and only those symbols that had had some popularity at least for some time or for some group of people are included.

Besides IPA symbols, TIPA also contains symbols that are useful for the following areas of phonetics and linguistics.

- Symbols used in the American phonetics (e.g. æ, ᴇ, ɷ, λ, etc.).
- Symbols used in the historical study of Indo-European languages (e.g. þ, ƿ, ƕ, ȥ, ь, ъ, and accents such as á́, ë́, etc.).
- Symbols used in the phonetic description of languages in East Asia (e.g. ɿ, ʅ, ȡ, ȵ, ȶ, etc.).
- Diacritics used in 'extIPA Symbols for Disordered Speech' [4] and 'VoQS (Voice Quality Symbols)' [1] (e.g. n̪͆, f͈, ḿ, etc).

It should be also noted that TIPA includes all the necessary elements of 'tone letters', enabling

[1] TIPA stands for *TeX IPA* or *Tokyo IPA*. The primary ftp site in which the latest version of TIPA is placed is `ftp://tooyoo.L.u-tokyo.ac.jp/pub/TeX/tipa`, and also it is mirrored onto the directory `fonts/tipa` of the CTAN archives.

[2] TSIPA was made in 1992 by Kobayashi Hajime, Fukui Rei and Shirakawa Shun. It is available from a CTAN archive. One problem with TSIPA was that symbols already included in `OT1`, `T1` or Math fonts are excluded, because of the limitation of its 128 character encoding. As a result, a string of phonetic representation had to be often composed of symbols from different fonts, disabling the possibility of automatic inter-word kerning. And also too many symbols had to be realized as macros.

[3] These macros are now defined in a separate file called '`exaccent.sty`' in order for the authors of other packages to be able to make use of them. The idea of separating these macros from other ones was suggested by Frank Mittelbach.

[4] This package (`vowel.sty`) can be used independently from the TIPA package. Documentation is also made separately in '`vowel.tex`' so that no further mention will be made here.

[5] In the case of TSIPA, the selection of symbols was based on "Computer coding of the IPA: Supplementary Report" [2].

[6] *Journal of the International Phonetic Association.*

all the theoretically possible combinations of the tone letter system. In the recent publication of the International Phonetic Association tone letters are admitted as an official way of representing tones but the treatment of tone letters is quite insufficient in that only a limited number of combination is allowed. This is apparently due to the fact that there has been no 'portable' way of combining symbols that can be used across various computer environments. Therefore TeX's productive system of macro is an ideal tool for handling a system like tone letters.

In the process of writing METAFONT source codes for TIPA phonetic symbols there have been many problems besides the one with the selection of symbols. One of such problems was that sometimes the exact shape of a symbol was unclear. For example, the shapes of the symbols such as ʗ (Stretched C), ʝ (Curly-tail J) differ according to sources. This is partly due to the fact that the IPA has been continuously revised for the past few decades, and partly due to the fact that different ways of computerizing phonetic symbols on different systems have resulted in the diversity of the shapes of phonetic symbols.

Although there is no definite answer to such a problem yet, it seems to me that it is a privilege of those working with METAFONT to have a systematic way of controlling the shapes of phonetic symbols.

Encoding The 256 character encoding of TIPA is now officially called the 'T3' encoding.[7] In deciding this new encoding, care is taken to harmonize with existing other encodings, especially with the T1 encoding. Also the easiness of inputting phonetic symbols is taken into consideration in such a way that frequently used symbols can be input with small number of keystrokes.

Table 1 shows the layout of the T3 encoding.

The basic structure of the encoding found in the first half of the table (character codes '000-'177) is based on normal text encodings (ASCII, OT1 and T1) in that sectioning of this area into several groups such as the section for accents and diacritics, the section for punctuation marks, the section for numerals, the sections for uppercase and lowercase letters is basically the same with these encodings.

Note also that the T3 encoding contains not only phonetic symbols but also usual punctuation marks that are used with phonetic symbols, and in such cases the same codes are assigned as the normal text encodings. However it is a matter of trade-off to decide which punctuation marks are to be included. For example ':' and ';' might have been preserved in T3 but in this case ':' has been traditionally used as a substitute for the length mark 'ː' so that I decided to exclude ':' in favor of the easiness of inputting the length mark by a single keystroke.

The encoding of the section for accents and diacritics is closely related to T1 in that the accents commonly included in T1 and T3 have the same encoding.

The sections for numerals and uppercase letters are filled with phonetic symbols that are used frequently in many languages, because numerals and uppercase letters are usually not used as phonetic symbols. And the assignments made here are used as the 'shortcut characters', which will be explained in the section entitled "Ordinary phonetic symbols" (page 105).

	'0	'1	'2	'3	'4	'5	'6	'7
'00x	Accents and diacritics							
'04x								
'05x	Punctuation marks							
'06x	Basic IPA symbols I (vowels)							
'07x			Diacritics, etc.					
'10x	Basic IPA symbols II							
'13x				Diacritics, etc.				
'14x	Pct.	Basic IPA symbols III (lowercase letters)						
'17x							Diacr.	
'20x	Tone letters and other suprasegmentals							
'23x								
'24x	Old IPA, non-IPA symbols							
'27x								
'30x	Extended IPA symbols							
'33x							Gmn.	
'34x	Basic IPA symbols IV							
'37x							Gmn.	

Pct. = Punctuation marks, Diacr. = Diacritics, Gmn. = Symbols for Germanic languages.

Table 1: Layout of the T3 encoding

[7] In a discussion with the LaTeX 2ε team it was suggested that the 128 character encoding used in WSUIPA would be refered to as the OT3 encoding.

ASCII	:	;	"							
TIPA	ː	ˑ	ˈ							
ASCII	0	1	2	3	4	5	6	7	8	9
TIPA	ʉ	ɨ	ʌ	ɜ	ɥ	ɐ	ɒ	ɤ	ɵ	ɘ
ASCII	@	A	B	C	D	E	F	G	H	I
TIPA	ə	ɑ	β	ç	ð	ɛ	ɸ	ɣ	ɦ	ɪ
ASCII	J	K	L	M	N	O	P	Q	R	S
TIPA	ʝ	ʁ	ʎ	ɱ	ŋ	ɔ	ʔ	ʕ	ɾ	ʃ
ASCII	T	U	V	W	X	Y	Z	|		
TIPA	θ	ʊ	ʋ	ɯ	χ	ʏ	ʒ	|		

Table 2: TIPA shortcut characters

As for the section for uppercase letters in the usual text encoding, a series of discussion among the members of the `ling-tex` mailing list revealed that there seem to be a certain amount of consensus on what symbols are to be assigned to each code. For example they were almost unanimous for the assignments such as ɑ for `A`, β for `B`, ð for `D`, ʃ for `S`, θ for `T`, etc. For more details, see table 2.

The encoding of the section for numerals was more difficult than the above case. One of the possibilities was to assign symbols based on the resemblance of shapes. One can easily think of assignments such as ɜ for `3` ɓ for `6`, etc. But the resemblance of shape alone does not serve as a criteria for all the assignments. So I decided to assign basic vowel symbols to this section.[8] Fortunately the resemblance of shape is to some extent maintained as is shown in table 2.

The encoding of the section for lowercase letters poses no problem since they are all used as phonetic symbols. Only one symbol, namely 'g', needs some attention because its shape should be 'ɡ', rather than 'g', as a phonetic symbol.[9]

The second half of the table (character codes `'200-'377`) is divided into four sections. The first section is devoted to the elements of tone letters and other suprasegmental symbols.

Among the remaining three sections the last section `'340-'377` contains more basic symbols than the other two sections. This is a result of assigning the same character codes as latin-1 (ISO8859-1) and T1 encodings to the symbols that are commonly included in TIPA, latin-1 and T1 encoded fonts.[10] These are the cases of æ, ø, œ, ç and þ. And within each section symbols are arraneged largely in alphabetical order.

For a table of the `T3` encoding, see Appendix C (page 114).

TIPA fonts

This version of TIPA includes two families of IPA fonts, `tipa` and `xipa`. The former family of fonts is for normal use with LaTeX, and the latter family is intended to be used with '`times.sty`'(PSNFSS). They all have the same `T3` encoding as explained in the previous section.

- `tipa`
  Roman: `tipa8`, `tipa9`, `tipa10`, `tipa12`, `tipa17`
  Slanted: `tipasl8`, `tipasl9`, `tipasl10`, `tipasl12`
  Bold extended: `tipabx8`, `tipabx9`, `tipabx10`, `tipabx12`
  Sans serif: `tipass8`, `tipass9`, `tipass10`, `tipass12`, `tipass17`
  Bold: `tipab10`
- `xipa`
  Roman: `xipa10`
  Slanted: `xipasl10`
  Bold: `xipab10`
  Sans serif: `xipass10`

All these fonts are made by METAFONT, based on the Computer Modern font series. In the case of the `xipa` series, parameters are adjusted so as to look fine when used with Times Roman (in the cases of `xipa10`, `xipasl10`, `xipab10`) and Helvetica (in the case of `xipass10`).

Usage

Declaration of TIPA package In order to use TIPA, first declare TIPA package at the preamble of a document.

```
\documentclass{article}
\usepackage{tipa}
```

Encoding options The above declaration uses `OT1` as the default text encoding. If you want to use TIPA symbols with `T1`, specify the option '`T1`'.

```
\documentclass{article}
\usepackage[T1]{tipa}
```

If you want to use a more complex form of encoding, declare the use of `fontenc` package by yourself and specify the option '`noenc`'. In this case the option '`T3`', which represents the TIPA encoding, must be included as an option to the `fontenc` package. For example, if you want to use TIPA and the University Washington Cyrillic (`OT2`) with the `T1` text encoding, the following command will do this.

[8] This idea was influenced by the above mentioned article by J. C. Wells [10].

[9] But the alternative shape 'g' is preserved in other section and can be used as `\textg`.

[10] This is based on a suggestion by Jörg Knappen.

```
\documentclass{article}
\usepackage[T3,OT2,T1]{fontenc}
\usepackage[noenc]{tipa}
```

By default, TIPA includes the `fontenc` package internally but the option `noenc` suppresses this.

Using TIPA with PSNFSS In order to use TIPA with `times.sty`, declare the use of `times.sty` before declaring `tipa` packages.

```
\documentclass{article}
\usepackage{times}
\usepackage{tipa}
```

Font description files `T3ptm.fd` and `T3phv.fd` are automatically loaded by the above declaration.

Other options TIPA can be extended by the options `tone`, `extra`.

If you want to use the optional package for 'tone letters', add '`tone`' option to the `\usepackage` command that declares `tipa` package.

```
\usepackage[tone]{tipa}
```

And if you want to use diacritics for extIPA and VoQS, specify '`extra`' option.

```
\usepackage[extra]{tipa}
```

Finally there is one more option called '`safe`', which is used to suppress definitions of some possibly 'dangerous' commands of TIPA.

```
\usepackage[safe]{tipa}
```

More specifically, the following commands are suppressed by declaring the `safe` option. Explanation on the function of each command will be given later.

- `\s` (equivalent to `\textsyllabic`)
- `\*` (already defined in plain TeX)
- `\|`, `\:`, `\;`, `\!` (already defined in LaTeX)

Input Commands for Phonetic Symbols

Ordinary phonetic symbols TIPA phonetic symbols can be input by the following two ways.

1. Input macro names in the normal text environment.
2. Input macro names or *shortcut characters* within the follwoing groups or environment.
   - `\textipa{...}`[11]
   - `{\tipaencoding ...}`
   - `\begin{IPA} ... \end{IPA}`

   (These groups and environment will be henceforth refered to as the *IPA environment*.)

A shortcut character refers to a single character that is assigned to a specific phonetic symbol and that can be directly input by an ordinary keyboard. In TIPA fonts, the character codes for numerals and uppercase letters in the normal ASCII encoding are assigned to such shortcut characters, because numerals and uppercase letters are usually not used as phonetic symbols. And additional shortcut characters for symbols such as æ, œ, ø may also be used if you are using a T1 encoded font and an appropriate input system for it.

The following pair of examples show the same phonetic transcription of a English word that are input by the above mentioned two input methods.

Input1: `[\textsecstress\textepsilon kspl`
`\textschwa\textprimstress ne`
`\textsci\textesh\textschwa n]`
Output1: [ˌɛkspləˈneɪʃən]
Input2: `\textipa{[""Ekspl@"neIS@n]}`
Output2: [ˌɛkspləˈneɪʃən]

It is apparent that inputting in the IPA environment is far easier than in the normal text environment. Moreover, although the outputs of the above examples look almost the same, they are *not* identical, exactly speaking. This is because in the IPA environment automatic kerning between symbols is enabled, as is illustrated by the following pair of examples.

Input1: `v\textturnv v w\textsca w`
`y\textturny y [\textesh]`
Output1: vʌv wᴀw yʎy [ʃ]
Input2: `\textipa{v2v w\textsca w yLy [S]}`
Output2: vʌv wᴀw yʎy [ʃ]

Table 2 shows most of the shortcut characters together with the corresponding characters in the ASCII encoding.

Naming of phonetic symbols Every TIPA phonetic symbol has a unique symbol name, such as *Turned A*, *Hooktop B*, *Schwa*.[12]Also each symbol has a corresponding control sequence name, such as `\textturna`, `\texthtb`, `\textschwa`. The name used as a control sequence is usually an abbreviated form of the corresponding symbol name with a prefix `\text`. The conventions used in the abbreviation are as follows.

- Suffixes and endings such as '-ive', '-al', '-ed' are omitted.

[11] I personally prefer a slightly shorter name like `\ipa` rather than `\textipa` but this command was named after the general convention of LaTeX 2ε. The same can be said to all the symbol names beginning with `\text`.

[12] The naming was made based on the literature listed in the section entitled "Selection of Symbols" (page 102). And users of TSIPA should be careful because TIPA's naming is slightly modified from that of TSIPA.

Symbol name	*Macro name*	*Symbol*
Turned A	`\textturna`	ɐ
Glottal Stop	`\textglotstop`	ʔ
Right-tail D	`\textrtaild`	ɖ
Small Capital G	`\textscg`	ɢ
Hooktop B	`\texthtb`	ɓ
Curly-tail C	`\textctc`	ɕ
Crossed H	`\textcrh`	ħ
Old L-Yogh Ligature	`\textOlyoghlig`	ɮ
Beta	`\textbeta`	β

Table 3: Naming of TIPA symbols

- 'right', 'left' are abbreviated to `r`, `l` respectively.
- For 'small capital' symbols, prefix `sc` is added.
- A symbol with a hooktop is abbreviated as `ht`...
- A symbol with a curly-tail is abbreviated as `ct`...
- A 'crossed' symbol is abbreviated as `cr`...
- A ligature is abbreviated as ...`lig`.
- For an old version of a symbol, prefix `O` is added.

Note that the prefix `O` (old) should be given in uppercase letter.

Table 3 shows some examples of correspondence between symbol names and control sequence names.

Ligatures Just like the symbols such as “, ”, –, —, fi, ff are realized as ligatures by inputting `‘‘`, `’’`, `--`, `---`, `fi`, `ff` in TEX, two of the TIPA symbols, namely *Secondary Stress* and *Double Pipe*, and double quotation marks[13] can be input as ligatures in the IPA environment.

Input: `\textipa{" "" | || ‘‘ ’’}`
Output: ˈ ˌ | ‖ “ ”

Special macros `\*`, `\;`, `\:` and `\!` TIPA defines `\*`, `\;`, `\:` and `\!` as special macros in order to easily input phonetic symbols that do not have a shortcut character explained above. Before explaining how to use these macros, it is necessary to note that these macros are primarily intended to be used by linguists who usually do not care about things in math mode. And they can be 'dangerous' in that they override existing LaTeX commands used in the math mode. So if you want to preserve the original meaning of these commands, daclare the option '`safe`' at the preamble.

The macro `\*` is used in three different ways. First, when this macro is followed by one of the letters f, k, r, t or w, it results in a turned symbol.[14]

Input: `\textipa{\*f \*k \*r \*t \*w}`
Output: ɟ ʞ ɹ ʇ ʍ

Secondly, when this macro is followed by one of the letters j, n, h, l or z, it results in a frequently used symbol that has otherwise no easy way to input.

Input: `\textipa{\*j \*n \*h \*l \*z}`
Output: ɟ ɲ ħ ɬ ɮ

Thirdly, when this macro is followed by letters other than the above cases, they are turned into the symbols of the default text font. This is useful in the IPA environment to select symbols temporarily from the normal text font.

Input: `\textipa{\*A dOg, \*B k\ae{}t,`
`ma\super{\*{214}}}`
Output: A dɔg, B kæt, ma^{214}

The remaining macros `\;`, `\:` and `\!` are used to make small capital symbols, retroflex symbols, and implosives or clicks, respectively.

Input: `\textipa{\;B \;E \;A \;H \;L \;R}`
Output: ʙ ᴇ ᴀ ʜ ʟ ʀ
Input: `\textipa{\:d \:l \:n \:r \:s \:z}`
Output: ɖ ɭ ɳ ɽ ʂ ʐ
Input: `\textipa{\!b \!d \!g \!j \!G \!o}`
Output: ɓ ɗ ɠ ʄ ʛ ʘ

Punctuation marks The following punctuation marks and text symbols that are normally included in the text encoding are also included in the `T3` encoding so that they can be directly input in the IPA environment.

Input: `\textipa{! ’ ( ) * + , - .\ / = ?`
`[ ] ‘}`
Output: ! ’ ( ) * + , - . / = ? [ ] ‘

All the other punctuation marks and text symbols that are not included in `T3` need to be input with a prefix `\*` explained in the last section when they appear in the IPA environment.

Input: `\textipa{\*; \*: \*@ \*\# \*\$`
`\*\& \*\% \*\{ \*\}}`
Output: ; : @ # $ & % { }

Accents and diacritics Table 4 shows how to input accents and diacritics in TIPA with some examples. Here again, there are two kinds of input methods; one for the normal text environment, and the other for the IPA environment.

In the IPA environment, most of the accents and diacritics can be input more easily than in the

[13] Although TIPA fonts do not include the symbols “ and ”, a negative value of kerning is automatically inserted between ‘ and ‘, ’ and ’, so that the same results can be obtained as in the case of the normal text font.

[14] This idea was pointed out by Jörg Knappen.

Input in the normal text environment	*Input in the IPA environment*	*Output*
`\'a`	`\'a`	á
`\"a`	`\"a`	ä
`\ a`	`\~a`	ã
`\r{a}`	`\r{a}`	å
`\textsyllabic{m}`	`\s{m}`	m̩
`\textsubumlaut{a}`	`\"*a`	a̤
`\textsubtilde{a}`	`\~*a`	a̰
`\textsubring{a}`	`\r*a`	ḁ
`\textdotacute{e}`	`\.'e`	ė́
`\textgravedot{e}`	`` \`.e ``	ė̀
`\textacutemacron{a}`	`\'=a`	ā́
`\textcircumdot{a}`	`\^.a`	ȧ̂
`\texttildedot{a}`	`\~.a`	ȧ̃
`\textbrevemacron{a}`	`\u=a`	ā̆

Table 4: Examples of inputting accents

Input in the normal text environment	*Input in the IPA environment*	*Output*
`\textsubbridge{t}`	`\\|[t`	t̪
`\textinvsubbridge{t}`	`\\|]t`	t̺
`\textsublhalfring{a}`	`\\|(a`	a̜
`\textsubrhalfring{a}`	`\\|)a`	a̹
`\textroundcap{k}`	`\\|c{k}`	k̑
`\textsubplus{o}`	`\\|+o`	o̟
`\textraising{e}`	`\\|'e`	e̝
`\textlowering{e}`	`` \\|`e ``	e̞
`\textadvancing{o}`	`\\|<o`	o̘
`\textretracting{a}`	`\\|>a`	a̙
`\textovercross{e}`	`\\|x{e}`	e̽
`\textsubw{k}`	`\\|w{k}`	k̫
`\textseagull{t}`	`\\|m{t}`	t̼

Table 5: Examples of the accent prefix `\|`

normal text environment, especially in the cases of subscript symbols that are normally placed over a symbol and in the cases of combined accents, as shown in the table.

As can be seen by the above examples, most of the accents that are normally placed over a symbol can be placed under a symbol by adding an `*` to the corresponding accent command in the IPA environment.

The advantage of IPA environment is further exemplified by the all-purpose accent `\|`, which is used as a macro prefix to provide shortcut inputs for the diacritics that otherwise have to be input by lengthy macro names. Table 5 shows examples of such accents. Note that the macro `\|` is also 'dangerous' in that it has been already defined as a math symbol of LaTeX. So if you want to preserve the original meaning of this macro, declare '`safe`' option at the preamble.

Finally, examples of words with complex accents that are input in the IPA environment are shown below.

Input:
```
\textipa{*\|c{k}\r*mt\'om
*bhr\'=at\=er}
```
Output: *k̑m̩̥tóm *bhrā́tēr

For a full list of accents and diacritics, see Appendix A

Superscript symbols In the normal text environment, superscript symbols can be input by a macro called `\textsuperscript`, which has been newly introduced in the recent version of LaTeX 2ε. This macro takes one argument which can be either a symbol or a string of symbols, and can be nested.

Since the name of this macro is too long, TIPA provides an abbreviated form of this macro called `\super`.

Input1:
```
t\textsuperscript h
k\textsuperscript w
a\textsuperscript{bc}
a\textsuperscript{b%
 \textsuperscript{c}}
```
Output1: t^h k^w a^{bc} a^{b^c}

Input2:
```
\textipa{t\super{h} k\super{w}
   a\super{bc} a\super{b\super{c}}}
```
Output2: t^h k^w a^{bc} a^{b^c}

These macros automatically select the correct size of superscript font no matter what size of the text font is used.

Tone letters TIPA provides a flexible system of macros for 'tone letters'. A tone letter is represented by a macro called '`\tone`', which takes one argument consisting of a string of numbers ranging from 1 to 5. These numbers denote pitch levels, 1 being the lowest and 5 the highest. Within this range, any combination is allowed and there is no limit in the length of combination.

As an example of the usage of tone letter macro, the four tones of Chinese are show below.

Input:
```
\tone{55}ma ``mother'',
\tone{35}ma ``hemp'',
\tone{214}ma ``horse'',
\tone{51}ma ``scold''
```
Output: ˥ma "mother", ˧˥ma "hemp",
˨˩˦ma "horse", ˥˩ma "scold"

Roman	
`\textipa{f@"nEtIks}`	fəˈnɛtɪks
Slanted	
`\textipa{\slshape f@"nEtIks}`	*fəˈnɛtɪks* or
`\textipa{\textsl{f@"nEtIks}`	*fəˈnɛtɪks* or
`\textsl{\textipa{f@"nEtIks}`	*fəˈnɛtɪks*
Bold extended	
`\textipa{\bfseries f@"nEtIks}`	**fəˈnɛtɪks** or
`\textipa{\textbf{f@"nEtIks}`	**fəˈnɛtɪks** or
`\textbf{\textipa{f@"nEtIks}`	**fəˈnɛtɪks**
Sans Serif	
`\textipa{\sffamily f@"nEtIks}`	fəˈnɛtɪks or
`\textipa{\textsf{f@"nEtIks}`	fəˈnɛtɪks or
`\textsf{\textipa{f@"nEtIks}`	fəˈnɛtɪks

Table 6: Examples of font switching

How easy to input phonetic symbols?

Let us briefly estimate here how much easy (or difficult) to input phonetic symbols with TIPA in terms of the number of keystrokes.

The following table shows statistics for all the phonetic symbols that appear in the '93 version of IPA chart (diacritics and symbols for suprasegmentals excluded). It is assumed here that each symbol is input within the IPA environment and the `safe` option is not specified.

keystrokes	*number*	*examples*
1	65	a, b, ə, ɑ, β, etc.
2	2	ø, ‖
3	30	æ, ʈ, ʙ, ɓ, etc.
5	1	ç
more than 5	7	ɞ, ʡ, ǂ, ɰ, etc.

As is shown in the table, about 92% of the symbols can be input within three keystrokes.

Changing font styles

This version of TIPA includes five styles of fonts, i.e. roman, slanted, bold, bold extended and sans serif. These styles can be switched in much the same way as in the normal text fonts (see table 6).

The bold fonts are usually not used within the standard LaTeX class packages so that if you want to use them, it is necessary to use low-level font selection commands of LaTeX 2_ε.

Input: `{\fontseries{b}\selectfont`
`abcdefg \textipa{ABCDEFG}}`
Output: **abcdefg ɑβɕðɛɸɣ**

Note also that slanting of TIPA symbols should correctly work even in the cases of combined accents and in the cases of symbols made up by macros.

Input: `\textsl{\textipa{\'{\"{\u*{e}}}}}`
Output: *ë̯́*
Input: `\textsl{\textdoublebaresh}`
Output: *ʃ̸* (This symbol is composed by a macro.)

Internal commands

Some of the internal commands of TIPA are defined without the letter `@` in order to allow a user to extend the capability of TIPA.

`\ipabar` Some TIPA symbols such as `\textbarb` ƀ, `\textcrtwo` ƻ are defined by using an internal macro command `\ipabar`. This command is useful when you want to make barred or crossed symbols not defined in TIPA.

This command requires the following five parameters to control the position of the bar.

- `#1` the symbol to be barred
- `#2` the height of the bar (in dimen)
- `#3` bar width
- `#4` left kern added to the bar
- `#5` right kern added to the bar

Parameters `#3`, `#4`, `#5` are to be given in a scaling factor to the width of the symbol, which is equal to 1 if the bar has the same width with the symbol in question. For example, the following command states a barred b (ƀ) of which the bar position in the y-coordinate is `.5ex` and the width of the bar is slightly larger than that of the letter b.

```
% Barred B
\newcommand\textbarb{%
  \ipabar{{\tipaencoding b}}%
    {.5ex}{1.1}{}{}}
```

Note that the parameters `#4` and `#5` can be left blank if the value is equal to 0.

And the next example declares a barred c (ꞓ) of which the bar width is a little more than half as large as the letter c and it has the same size of kerning at the right.

```
% Barred C
\newcommand\textbarc{%
  \ipabar{{\tipaencoding c}}%
    {.5ex}{.55}{}{.55}}
```

More complex examples with the `\ipabar` command are found in `T3enc.def`.

`\tipaloweraccent`, `\tipaupperaccent` These two commands are used in the definitions of TIPA accents and diacritics. They are special forms of the commands `\loweraccent` and `\upperaccent` that are defined in `exaccent.sty`. The difference between the commands with the prefix `tipa` and the ones without it is that the former commands select accents from a T3 encoded font while the latter ones do so from the current text font.

These commands take two parameters, the code of the accent (in decimal, octal or hexadecimal number) and the symbol to be accented, as shown below.

Input: `\tipaupperaccent{0}{a}`
Output: a̦

Optionally, these commands can take a extra parameter to adjust the vertical position of the accent. Such an adjustment is sometimes necessary in the definition of a nested accent. The next example shows TIPA's definition of the 'Circumflex Dot Accent' (e.g. ȧ̂).

```
% Circumflex Dot Accent
\newcommand\textcircumdot[1]%
  {\tipaupperaccent[-.2ex]{2}%
  {\tipaupperaccent[-.1ex]{10}{#1}}}
```

This definition states that a dot accent is placed over a symbol thereby reducing the vertical distance between the symbol and the dot by `.1ex` and a circumflex accent is placed over the dot and the distance between the two accents is reduced by `.2ex`.

If you want to make a combined accent not included in TIPA, you can do so fairly easily by using these two commands together with the optional parameter. For more examples of these commands, see `tipa.sty` and `extraipa.sty`.

`\tipaLoweraccent`, `\tipaUpperaccent` These two commands differ from the two commands explaind above in that the first parameter should be a symbol (or any other things, typically an `\hbox`), rather than the code of the accent. They are special cases of the commands `\Loweraccent` and `\Upperaccent` and the difference between the two pairs of commands is the same as before.

The next example makes a schwa an accent.

Input:
```
\tipaUpperaccent[.2ex]%
    {\lower.8ex\hbox{%
      \textipa{\super@}}}{a}
```
Output: åᷪ

Acknowledgments

First of all, many thanks are due to the co-authors of TSIPA, Kobayashi Hajime and Shirakawa Shun. Kobayashi Hajime was the main font designer of TSIPA. Shirakawa Shun worked very hard in deciding encoding, checking the shapes of symbols and writing the Japanese version of document. TIPA was impossible without TSIPA.

I would like to thank also Jörg Knappen whose insightful comments helped greatly in many ways the development of TIPA. I was also helped and encouraged by Christina Thiele, Martin Haase, Kirk Sullivan and many other members of the `ling-tex` mailing list.

At the last stage of the development of TIPA Frank Mittelbach gave me precious comments on how to incorporate various TIPA commands into the NFSS. I would like to thank also Barbara Beeton who kindly read over the preliminary draft of this document and gave me useful comments.

References

[1] Martin J. Ball, John Esling, and Craig Dickson. VoQS: Voice Quality Symbols. 1994, 1994.

[2] John Esling. Computer coding of the IPA: Supplementary report. *Journal of the International Phonetic Association*, 20(1):22–26, 1990.

[3] John H. Esling and Harry Gaylord. Computer codes for phonetic symbols. *Journal of the International Phonetic Association*, 23(2):83–97, 1993.

[4] ICPLA. extIPA Symbols for Disorderd Speech. 1994, 1994.

[5] IPA. *The Principles of the International Phonetic Association*, 1949.

[6] IPA. Report on the 1989 Kiel Convention. *Journal of the International Phonetic Association*, 19(2):67–80, 1989.

[7] IPA. Further report on the 1989 Kiel Convention. *Journal of the International Phonetic Association*, 20(2):22–24, 1990.

[8] IPA. Council actions on revisions of the IPA. *Journal of the International Phonetic Association*, 23(1):32–34, 1993.

[9] Geoffrey K. Pullum and William A. Ladusaw. *Phonetic Symbol Guide*. The University of Chicago Press, 1986.

[10] John C. Wells. Computer-coding the IPA: a proposed extension of SAMPA. Revised draft 1995 04 28, 1995.

Appendix

A A List of TIPA Symbols

For each symbol the following information is shown: (1) the symbol, (2) input method in the normal text environment (and a shortcut method that can be used within the IPA environment in parenthesis), (3) the name of the symbol.

Vowels and Consonants

a	`a`	Lower-case A

ɐ	`\textturna (5)`	Turned A
ɑ	`\textscripta (A)`	Script A
ɒ	`\textturnscripta (6)`	Turned Script A
æ	`\ae`	Ash
ᴀ	`\textsca (\;A)`	Small Capital A[15]
ʌ	`\textturnv (2)`	Turned V[16]
b	`b`	Lower-case B
ь	`\textsoftsign`	Soft Sign
ъ	`\texthardsign`	Hard Sign
ɓ	`\texthtb (\!b)`	Hooktop B
ʙ	`\textscb (\;B)`	Small Capital B
ƀ	`\textcrb`	Crossed B
b̵	`\textbarb`	Barred B
β	`\textbeta (B)`	Beta
c	`c`	Lower-case C
ȼ	`\textbarc`	Barred C
ƈ	`\texthtc`	Hooktop C
č	`\v{c}`	C Wedge
ç	`\c{c}`	C Cedilla
ɕ	`\textctc (C)`	Curly-tail C
ʗ	`\textstretchc`	Stretched C[17]
d	`d`	Lower-case D
đ	`\textcrd`	Crossed D
d̵	`\textbard`	Barred D
ɗ	`\texthtd (\!d)`	Hooktop D
ɖ	`\textrtaild (\:d)`	Right-tail D
ȡ	`\textctd`	Curly-tail D
ʣ	`\textdzlig`	D-Z Ligature
ʥ	`\textdctzlig`	D-Curly-tail Z Ligature
ʤ	`\textdyoghlig`	D-Yogh Ligature
ȡʑ	`\textctdctzlig`	Curly-tail D-Curly-tail Z Ligature
ð	`\dh (D)`	Eth
e	`e`	Lower-case E
ə	`\textschwa (@)`	Schwa
ɚ	`\textrhookschwa`	Right-hook Schwa
ɘ	`\textreve (9)`	Reversed E
ᴇ	`\textsce (\;E)`	Small Capital E
ɛ	`\textepsilon (E)`	Epsilon
ʚ	`\textcloseepsilon`	Closed Epsilon
ɜ	`\textrevepsilon (3)`	Reversed Epsilon
ɝ	`\textrhookrevepsilon`	Right-hook Reversed Epsilon
ɞ	`\textcloserevepsilon`	Closed Reversed Epsilon

f	`f`	Lower-case F
ɡ	`\textg (g)`	Lower-case G
ǥ	`\textbarg`	Barred G
g̵	`\textcrg`	Crossed G
ɠ	`\texthtg (\!g)`	Hooktop G
g	`g (\textg)`	Text G
ɢ	`\textscg (\;G)`	Small Capital G
ʛ	`\texthtscg (\!G)`	Hooktop Small Capital G
ɣ	`\textgamma (G)`	Gamma
ɤ	`\textbabygamma`	Baby Gamma
ɤ	`\textramshorns (7)`	Ram's Horns
h	`h`	Lower-case H
ƕ	`\texthvlig`	H-V Ligature
ħ	`\textcrh`	Crossed H
ɦ	`\texthth (H)`	Hooktop H
ɧ	`\texththeng`	Hooktop Heng
ɥ	`\textturnh (4)`	Turned H
ʜ	`\textsch (\;H)`	Small Capital H
i	`i`	Lower-case I
ı	`\i`	Undotted I
ɨ	`\textbari (1)`	Barred I
ɩ	`\textiota`	Iota
ɿ	`\textlhti`	Left-hooktop I[18]
ʅ	`\textlhtlongi`	Left-hooktop Long I
ʅ̡	`\textvibyi`	Viby I[19]
ʅ̡	`\textraisevibyi`	Raised Viby I
ɪ	`\textsci (I)`	Small Capital I
j	`j`	Lower-case J
ȷ	`\j`	Undotted J
ʝ	`\textctj (J)`	Curly-tail J[20]
ᴊ	`\textscj (\;J)`	Small Capital J
ǰ	`\v{\j}`	J Wedge
ɟ	`\textbardotlessj`	Barred Dotless J
ɟ	`\textObardotlessj`	Old Barred Dotless J
ʄ	`\texthtbardotlessj (\!j)`	Hooktop Barred Dotless J[21]
k	`k`	Lower-case K
ƙ	`\texthtk`	Hooktop K
ʞ	`\textturnk (\*k)`	Turned K
l	`l`	Lower-case L

[15] This symbol is fairly common among Chinese phoneticians.

[16] In *PSG* this symbol is called 'Inverted V' but it is apparently a mistake.

[17] The shape of this symbol differs according to the sources. In *PSG* and recent articles in *JIPA*, it is 'stretched' toward both the ascender and descender regions and the whole shape looks like a thick staple. In the old days, however, it was streched only toward the ascender and the whole shape looked more like a stretched c.

[18] The four symbols ɿ, ʅ, ʅ̡ and ʅ̡ are mainly used among Chinese linguists. These symbols are based on "det svenska landsmålsalfabetet" and introduced to China by Bernhard Karlgren. The original shapes of these symbols were in italic as was always the case with "det svenska landsmålsalfabetet". It seems that the Chinese linguists who wanted to continue to use these symbols in IPA changed their shapes upright.

[19] I call this symbol 'Viby I', based on the following description by Bernhard Karlgren: "Une voyelle très analogue à ʅ̡ se rencontre dans certains dial. suédois; on l'appelle 'i de Viby'."(*Études sur la phonologie chinoise*, 1915–26, p. 295)

[20] In the official IPA charts of '89 and '93, this symbol has a dish serif on top of the stem, rather than the normal sloped serif found in the letter j. I found no reason why it should have a dish serif here, so I changed it to a normal sloped serif.

[21] In *PSG* the shape of this symbol slightly differs. Here I followed the shape found in IPA '89, '93.

ɫ	`\textltilde (|~l)`	L with Tilde
ƚ	`\textbarl`	Barred L
ɬ	`\textbeltl`	Belted L
ɭ	`\textrtaill (\:l)`	Right-tail L
ɮ	`\textlyoghlig`	L-Yogh Ligature
ɮ	`\textOlyoghlig`	Old L-Yogh Ligature
ʟ	`\textscl (\;L)`	Small Capital L
λ	`\textlambda`	Lambda
ƛ	`\textcrlambda`	Crossed Lambda
m	`m`	Lower-case M
ɱ	`\textltailm (M)`	Left-tail M (at right)
ɯ	`\textturnm (W)`	Turned M
ɰ	`\textturnmrleg`	Turned M, Right Leg
n	`n`	Lower-case N
ƞ	`\textnrleg`	N, Right Leg
ñ	`\~n`	N with Tilde
ɲ	`\textltailn`	Left-tail N (at left)
ŋ	`\ng (N)`	Eng
ɳ	`\textrtailn (\:n)`	Right-tail N
ȵ	`\textctn`	Curly-tail N
ɴ	`\textscn (\;N)`	Small Capital N
o	`o`	Lower-case O
ʘ	`\textbullseye (\!o)`	Bull's Eye
ɵ	`\textbaro (8)`	Barred O
ø	`\o`	Slashed O
œ	`\oe`	O-E Ligature
ɶ	`\textscoelig (\OE)`	Small Capital O-E Ligature
ɔ	`\textopeno (O)`	Open O
ᴔ	`\textturncelig`	Turned C(Open O)-E Ligature
ω	`\textomega`	Omega
ꭥ	`\textscomega`	Small Capital Omega
ɷ	`\textcloseomega`	Closed Omega
p	`p`	Lower-case P
ƿ	`\textwynn`	Wynn
þ	`\textthorn (\th)`	Thorn
ƥ	`\texthtp`	Hooktop P
ɸ	`\textphi (F)`	Phi
q	`q`	Lower-case Q
ʠ	`\texthtq`	Hooktop Q
ꞯ	`\textscq (\;Q)`	Small Capital Q[22]
r	`r`	Lower-case R
ɾ	`\textfishhookr (R)`	Fish-hook R
ɼ	`\textlonglegr`	Long-leg R
ɽ	`\textrtailr (\:r)`	Right-tail R
ɹ	`\textturnr (\*r)`	Turned R
ɻ	`\textturnrrtail (\:R)`	Turned R, Right Tail
ɺ	`\textturnlonglegr`	Turned Long-leg R

ʀ	`\textscr (\;R)`	Small Capital R
ʁ	`\textinvscr (K)`	Inverted Small Capital R
s	`s`	Lower-case S
š	`\v{s}`	S Wedge
ʂ	`\textrtails (\:s)`	Right-tail S (at left)
ʃ	`\textesh (S)`	Esh
ʃ	`\textdoublebaresh`	Doube-barred Esh
ʆ	`\textctesh`	Curly-tail Esh
t	`t`	Lower-case T
ƭ	`\texthtt`	Hooktop T
ƫ	`\textlhookt`	Left-hook T
ʈ	`\textrtailt (\:t)`	Right-tail T
ʨ	`\texttctclig`	T-Curly-tail C Ligature
ʦ	`\texttslig`	T-S Ligature
ʧ	`\textteshlig`	T-Esh Ligature
ʇ	`\textturnt (\*t)`	Turned T
ȶ	`\textctt`	Curly-tail T
ȶɕ	`\textcttctclig`	Curly-tail T-Curly-tail C Ligature
θ	`\texttheta (T)`	Theta
u	`u`	Lower-case U
ʉ	`\textbaru (0)`	Barred U
ʊ	`\textupsilon (U)`	Upsilon
ᴜ	`\textscu (\;U)`	Small Capital U
v	`v`	Lower-case V
ʋ	`\textscriptv (V)`	Script V
w	`w`	Lower-case W
ʍ	`\textturnw (\*w)`	Turned W
x	`x`	Lower-case X
χ	`\textchi (X)`	Chi
y	`y`	Lower-case Y
ʎ	`\textturny (L)`	Turned Y
ʏ	`\textscy (Y)`	Small Capital Y
ɥ	`\textvibyy`	Viby Y[23]
z	`z`	Lower-case Z
ȥ	`\textcommatailz`	Comma-tail Z
ž	`\v{z}`	Z Wedge
ʑ	`\textctz`	Curly-tail Z
ƹ	`\textrevyogh`	Reversed Yogh
ʐ	`\textrtailz (\:z)`	Right-tail Z
ʒ	`\textyogh (Z)`	Yogh
ʓ	`\textctyogh`	Curly-tail Yogh
ƻ	`\textcrtwo`	Crossed 2
ʔ	`\textglotstop (P)`	Glottal Stop
ˀ	`\textraiseglotstop`	Superscript Glottal Stop
ʡ	`\textbarglotstop`	Barred Glottal Stop
ʖ	`\textinvglotstop`	Inverted Glottal Stop
ƾ	`\textcrinvglotstop`	Crossed Inverted Glottal Stop
ʕ	`\textrevglotstop (Q)`	Reversed Glottal Stop
ʢ	`\textbarrevglotstop`	Barred Reversed Glottal Stop

[22] Suggested by Prof S. Tsuchida for Austronesian languages in Taiwan. In *PSG* 'Female Sign' and 'Uncrossed Female Sign'(pp. 110–111) are noted for pharyngeal stops, as proposed by Trager (1964). Also I'm not sure about the difference between an epiglottal plosive and a pharyngeal stop.

[23] See explanations in footnote 19.

ǀ	\textpipe (|)	Pipe
ǂ	\textdoublebarpipe	Double-barred Pipe
≠	\textdoublebarslash	Double-barred Slash
ǁ	\textdoublepipe (||)	Double Pipe
!	!	Exclamation Point

Suprasegmentals

ˈ	\textprimstress (")	Vertical Stroke (Superior)
ˌ	\textsecstress ("")	Vertical Stroke (Inferior)
ː	\textlengthmark (:)	Length Mark
ˑ	\texthalflength (;)	Half-length Mark
|	\textvertline	Vertical Line
‖	\textdoublevertline	Double Vertical Line
‿	\textbottomtiebar (\t*{})	Bottom Tie Bar
↘	\textglobfall	Downward Diagonal Arrow
↗	\textglobrise	Upward Diagonal Arrow
↓	\textdownstep	Down Arrow[24]
↑	\textupstep	Up Arrow

Accents and Diacritics

è	\`e	Grave Accent
é	\'e	Acute Accent
ê	\^e	Circumflex Accent
ẽ	\~e	Tilde
ë	\"e	Umlaut
e̋	\H{e}	Double Acute Accent
e̊	\r{e}	Ring
ě	\v{e}	Wedge
ĕ	\u{e}	Breve
ē	\=e	Macron
ė	\.e	Dot
ȩ	\c{e}	Cedille
ę	\textpolhook{e} (\k{e})	Polish Hook (Ogonek Accent)
ȅ	\textdoublegrave{e} (\H*e)	Double Grave Accent
e̖	\textsubgrave{e} (\`*e)	Subscript Grave Accent
e̗	\textsubacute{e} (\'*e)	Subscript Acute Accent
ḙ	\textsubcircum{e} (\^*e)	Subscript Circumflex Accent
g̑	\textroundcap{g} (\\|c{g})	Round Cap
ā́	\textacutemacron{a} (\'=a)	Acute Accent with Macron
a̍	\textvbaraccent{a}	Vertical Bar Accent
a̎	\textdoublevbaraccent{a}	Double Vertical Bar Accent
ė̀	\textgravedot{e} (\`.e)	Grave Dot Accent
ė́	\textdotacute{e} (\'.e)	Dot Acute Accent
ȧ̂	\textcircumdot{a} (\^.a)	Circumflex Dot Accent
ȧ̃	\texttildedot{a} (\~.a)	Tilde Dot Accent
ā̆	\textbrevemacron{a} (\u=a)	Breve Macron Accent
ā̊	\textringmacron{a} (\r=a)	Ring Macron Accent
š́	\textacutewedge{s} (\v's)	Acute Wedge Accent
ȧ̆	\textdotbreve{a}	Dot Breve Accent
t̪	\textsubbridge{t} (\\|[t)	Subscript Bridge
d̺	\textinvsubbridge{d} (\\|]t)	Inverted Subscript Bridge
n̻	\textsubsquare{n}	Subscript Square
o̹	\textsubrhalfring{o} (\\|)o)	Subscript Right Half-ring[25]
o̜	\textsublhalfring{o} (\\|(o)	Subscript Left Half-ring
k̫	\textsubw{k} (\\|w{k})	Subscript W
ğ	\textoverw{g}	Over W
t̼	\textseagull{t} (\\|m{t})	Seagull
e̽	\textovercross{\e} (\\|x{e})	Over-cross
ɔ̟	\textsubplus{\textopeno} (\\|+O)	Subscript Plus[26]
ɛ̝	\textraising{\textepsilon} (\\|'E)	Raising Sign
e̞	\textlowering{e} (\\|`e)	Lowering Sign
u̟	\textadvancing{u} (\\|<u)	Advancing Sign
ə̠	\textretracting{\textschwa} (\\|>@)	Retracting Sign
ḛ	\textsubtilde{e} (\~*e)	Subscript Tilde
e̤	\textsubumlaut{e} (\"*e)	Subscript Umlaut
u̥	\textsubring{u} (\r*u)	Subscript Ring
e̬	\textsubwedge{e} (\v*e)	Subscript Wedge
e̱	\textsubbar{e} (\=*e)	Subscript Bar
ẹ	\textsubdot{e} (\.*e)	Subscript Dot
e̯	\textsubarch{e}	Subscript Arch
m̩	\textsyllabic{m} (\s{m})	Syllabicity Mark
t̴	\textsuperimposetilde{t} (\\|~{t})	Superimposed Tilde
t˺	t\textcorner	Corner
t˹	t\textopencorner	Open Corner
ə˞	\textschwa\rhoticity	Rhoticity
b′	b\textceltpal	Celtic Palatalization Mark
k˂	k\textlptr	Left Pointer
k˃	k\textrptr	Right Pointer
p	p\textrectangle	Rectangle[27]

[24] The shapes of \textdownstep and \textupstep differ according to sources. Here I followed the shapes found in the recent IPA charts.

[25] Diacritics \textsubrhalfring and \textsublhalfring can be placed after a symbol by inputting, for example, [e\textsubrhalfring{}] [e˒].

[26] The diacritics such as \textsubplus, \textraising, \textlowering \textadvancing and \textretracting can be placed after a symbol by inputting [e\textsubplus{}] [e˖], for example.

[27] This symbol is used among Japanese linguists as a diacritical symbol indicating no audible release (IPA ˺), because the symbol ˺ is used to indicate pitch accent in Japanese.

g͡b	`\texttoptiebar{gb}` (`\t{gb}`)	Top Tie Bar
’	`’`	Apostrophe
ʻ	`\textrevapostrophe`	Reversed Apostrophe
.	`.`	Period
ʽ	`\texthooktop`	Hooktop
̢	`\textrthook`	Right Hook
̡	`\textpalhook`	Palatalization Hook
p^h	`p\textsuperscript{h}` (`p\super h`)	Superscript H
k^w	`k\textsuperscript{w}` (`k\super w`)	Superscript W
t^j	`t\textsuperscript{j}` (`t\super j`)	Superscript J
tˠ	`t\textsuperscript{\textgamma}` (`t\super G`)	Superscript Gamma
dˤ	`d\textsuperscript{\textrevglotstop}` (`d\super Q`)	Superscript Reversed Glottal Stop
d^n	`d\textsuperscript{n}` (`d\super n`)	Superscript N
d^l	`d\textsuperscript{l}` (`d\super l`)	Superscript L

Tone letters

The tones illustrated here are only a representative sample of what is possible. For more details see the section entitled "Tone Letters" (page 107).

˥	`\tone{55}`	Extra High Tone
˦	`\tone{44}`	High Tone
˧	`\tone{33}`	Mid Tone
˨	`\tone{22}`	Low Tone
˩	`\tone{11}`	Extra Low Tone
˥˩	`\tone{51}`	Falling Tone
˩˥	`\tone{15}`	Rising Tone
˦˥	`\tone{45}`	High Rising Tone
˩˨	`\tone{12}`	Low Rising Tone
˦˥˦	`\tone{454}`	High Rising Falling Tone

Diacritics for extIPA, VoQS

In order to use diacritics listed in this section, it is necessary to specify the option '`extra`' at the preamble (See the section entitled "Other options" on page 105). Note also that some of the diacritics are defined by using symbols from fonts other than TIPA so that they may not look quite satisfactory and/or may not be slanted (e.g. `\whistle{s}` s͎).

s͍	`\spreadlips{s}`	Left Right Arrow
v͆	`\overbridge{v}`	Overbridge
n̪͆	`\bibridge{n}`	Bibridge
t͇	`\subdoublebar{t}`	Subscript Double Bar
f͈	`\subdoublevert{f}`	Subscript Double Vertical Line
v˺	`\subcorner{v}`	Subscript Corner
s͎	`\whistle{s}`	Up Arrow
θ͢s	`\sliding{\ipa{Ts}}`	Right Arrow
m̃	`\crtilde{m}`	Crossed tilde
ȧ̃	`\dottedtilde{a}`	Dotted Tilde
s̃̃	`\doubletilde{s}`	Double Tilde
(n̥)	`\partvoiceless{n}`	Parenthesis + Ring
(n̥	`\inipartvoiceless{n}`	Parenthesis + Ring
n̥)	`\finpartvoiceless{n}`	Parenthesis + Ring
(s̬)	`\partvoice{s}`	Parenthesis + Subwedge
(s̬	`\inipartvoice{s}`	Parenthesis + Subwedge
s̬)	`\finpartvoice{s}`	Parenthesis + Subwedge
J˱	`\sublptr{J}`	Subscript Left Pointer
J˲	`\subrptr{J}`	Subscript Right Pointer

B B Symbols not included in TIPA

There are about 40 symbols that appear in *PSG* but are not included or defined in TIPA (ordinary capital letters, Greek letters and punctuation marks excluded). Most of such symbols are the ones that have been proposed by someone but never or rarely been followed by other people.

Some of such symbols can be realized by writing appropriate macros, while some others cannot be realized without resorting to the Metafont.

This section discusses these problems by classifying such symbols into three categories, as shown below.

1. Symbols that can be realized by TeX's macro level and/or by using symbols from other fonts.
2. Symbols that can be imitated by TeX's macro level and/or by using symbols from other fonts (but may not look quite satisfactory).
3. Symbols that cannot be realized at all, without creating a new font.

The following table shows symbols that belong to the first category. For each symbol, an example of input method and its output is also given. Note that barred or crossed symbols can be easily made by TIPA's `\ipabar` macro.

Script Lowe-case F	`{\itshape f}`	*f*
Barred Small Capital I	`\ipabar{\textsci}{.5ex}{1.1}{}{}`	ᵻ
Barred J	`\ipabar{j}{.5ex}{1.1}{}{}`	ɟ
Crossed K	`\ipabar{k}{1.2ex}{.6}{}{.4}`	ꝁ
Barred Open O	`\ipabar{\textopeno}{.5ex}{.6}{.4}{}`	ɔ̵
Barred Small Capital Omega	`\ipabar{\textscomega}{.5ex}{1.1}{}{}`	ᵿ
Barred P		

	`\ipabar{p}{.5ex}{1.1}{}{}`	ᵽ
Half-barred U	`\ipabar{u}{.5ex}{.5}{}{.5}`	ʉ
Barred Small Capital U	`\ipabar{\textscu}{.5ex}{1.1}{}{}`	ᵾ
Null Sign	`$\emptyset$`	∅
Double Slash	`/\kern-.25em/`	//
Triple Slash	`/\kern-.25em/\kern-.25em/`	///
Pointer (Upward)	`k\super{\tiny$\wedge$}`	$\mathrm{k}^{\wedge}$
Pointer (Downward)	`k\super{\tiny$\vee$}`	$\mathrm{k}^{\vee}$
Superscript Arrow	`k\super{\super{$\leftarrow$}}`	$\mathrm{k}^{\leftarrow}$

Symbols that belong to the second category are shown below. Note that slashed symbols can be in fact easily made by a macro. For example, a slashd b i.e. b̸ can be made by `\ipaclap{b}{/}`. The reason why slashed symbols are not included in TIPA is as follows: first, a simple overlapping of a symbol and a slash does not always result in a good shape, and secondly, it doesn't seem significant to devise fine-tuned macros for symbols which were created essentialy for typewriters.

Right-hook A	ᶏ
Slashed B	b̸
Slashed C	c̸
Slashed D	d̸
Small Capital Delta	Δ
Right-hook E	ᶒ
Right-hook Epsilon	ᶓ
Small Capital F	ꜰ
Female Sign	
Uncrossed Female Sign	
Right-hook Open O	
Slashed U	u̸
Slashed W	w̸

And finally, symbols that cannot be realized at all are as follows.

- Reversed Turned Script A
- A-O Ligature
- Inverted Small Capital A
- Small Capital A-O Ligature
- D with Upper-left Hook
- Hooktop H with Rightward Tail
- Heng
- Hooktop J
- Front-bar N
- Inverted Lower-case Omega
- Reversed Esh with Top Loop
- T with Upper Left Hook
- Turned Small Capital U
- Bent-tail Yogh
- Turned 2
- Turned 3

C C TIPA encoding (T3)

	'0	*'1*	*'2*	*'3*	*'4*	*'5*	*'6*	*'7*
'00x	,	´	ˆ	˜	¨	˝	˚	ˇ
'01x	˘	¯	˙	̡	̢	˵	ˋ	ˊ
'02x	⁀	⊓	⊔	▫	˒	˓	̼	ͫ
'03x	×	ı	ȷ	₊	˔	˕	⊣	⊢
'04x	ʹ	!	ˈ	˞	˛	ͅ	~	’
'05x	(	)	*	+	,	-	.	/
'06x	ʉ	ɨ	ʌ	ɜ	ɥ	ɐ	ɒ	ɤ
'07x	ɵ	ɘ	ː	ˑ	‿	=	⁀	?
'10x	ə	ɑ	β	ɕ	ð	ɛ	ɸ	ɣ
'11x	ɦ	ɪ	ʝ	ʁ	ʎ	ɱ	ŋ	ɔ
'12x	ʔ	ʕ	ɾ	ʃ	θ	ʊ	ʋ	ɯ
'13x	χ	ʏ	ʒ	[	˓	]	˺	˹
'14x	ʻ	a	b	c	d	e	f	g
'15x	h	i	j	k	l	m	n	o
'16x	p	q	r	s	t	u	v	w
'17x	x	y	z	‖	|	ǂ	˷	ˌ
'20x	_	ˎ	⟍	\	\	ˏ	⟋	/
'21x	/	_	ˎ	ˎ	\	\	ˏ	ˏ
'22x	/	/	|	‖	↓	↑	↗	↘
'23x	ˏ	ˎ	ˆ	˜	ˈ	ʺ	ˋ·	·ˊ
'24x	ƀ	đ	ᶑ	ᴇ	ɡ	ı	ɿ	ʅ
'25x	ᴊ	ʞ	ƚ	λ	ɮ	ᶇ	ƞ	æ
'26x	ω	ꭥ	ꞯ	ʆ	ƫ	ȶ	ʦ	ʮ
'27x	ʓ	ᶓ	ь	ъ	ˀ	˂	˃	
'30x	ᴀ	ƈ	ʗ	ʥ	ɚ	ʚ	ɞ	ɝ
'31x	ɤ	ʛ	ɧ	ʜ	ɩ	ɟ	ƙ	ʟ
'32x	ɮ	ɷ	ꞵ	ʠ	ɼ	ɺ	ʄ	ɶ
'33x	ʇ	ʧ	ᴜ	ʖ	ʡ	ʢ	ȥ	ƥ
'34x	ʙ	ɓ	ɗ	ɖ	ɠ	ɢ	æ	ç
'35x	ħ	ɟ	ʄ	ł	ɬ	ɭ	ɰ	ɳ
'36x	ɴ	ɲ	ʘ	ɽ	ɹ	ɻ	ʀ	œ
'37x	ø	ʂ	ʈ	ʍ	ʐ	ʑ	þ	ɧ

Computer Modern Typefaces as the Multiple Master Fonts

A.S. Berdnikov
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
berd@ianin.spb.su

O.A. Grineva
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
olga@ianin.spb.su

Introduction

Several years ago Adobe Inc. announced its new Multiple Master font format which enabled one to vary smoothly the font characteristics (say, *weight* from light to black, *width* from condensed to expanded, etc.) and create a unique font which suites the User's demands. Like many other "new" inventions in computer-assisted typography,[1] the roots of this idea can be found inside TEX — namely, in the Computer Modern font family created by D. Knuth in 1977 – 1985 [1]. These fonts are parametrized using 62 (!!!) parameters most of which are independent. It can be seen easily that it exceeds the flexibility of *any* multiple master font which has been created up to now or even *will be* created by somebody in future.

The METAFONT source code and the parameters for the *Computer Modern* series were created using the advice of such professional font designers as Hermann Zapf, Matthew Carter, Charles Bigelow and others. The METAFONT source code is designed so that the parameter files are separated from the main source code; all the parameters together with the relations between them are totally documented [1], and the font variations (provided that they have a name different from the original CM fonts) are encouraged by the author. The parametric representation of the parameters for canonical Computer Modern typefaces created by John Sauter and Karl Berry (SAUTER fonts [2]), and, on a different basis, by Jörg Knappen and Norbert Schwarz in *European Computer Modern* typefaces [3], enables one to vary smoothly the font size in a wide range without loosing high quality of the resulting fonts. All these facts provide the basis of the LATEX macros MFF.STY, which enables one to treat *Computer Modern typefaces* like multiple master fonts.

The MFF.STY macros follow the ideas implemented in the MFPIC package and allows specification of new fonts dynamically within a LATEX document without dealing with the details of METAFONT programming and without manual manipulations of *each* of 62 parameters used in METAFONT source files. Like MFPIC, the first pass of LATEX creates the METAFONT source file (substituting font `dummy` instead of the user-defined font), the METAFONT source file is processed by METAFONT, and at the second pass the generated font is used to format the document properly.

The user can vary the font shape smoothly between CMR, CMBX, CMSL, CMSS, CMTT and CMFF font families, specify the *weight*, *width*, *height* and *contrast* of the output font independently, and in addition he/she can play with the character characteristics so that the resulting output does not look like the canonical *Computer Modern* typefaces at all. Although originally the package was created for internal purposes to facilitate the investigation of the possibilities hidden inside Computer Modern source code, it can be useful for professional typographic purposes too.

Font series with the arbitrary design size

Suppose that there is a smooth approximation for *Computer Modern Roman* (CMR) which enables one to calculate the METAFONT font parameters for an *arbitrtary* design size even with weak TEX arithmetical capabilities. The desired font size is specified as the input parameter, all the internal calculations of the font parameters are performed by TEX, and the result is a METAFONT *ready-to-run* font header file for a new font. When this new font header file is processed by METAFONT, it can

[1] For example, *Microsoft Word* 6.0 was announced as the *first* program which enables to mark some place in the text by a special marker and then to refer to its position in a form: "see page . . . ".

be used in TEX documents like other generic TEX fonts.

This operation is performed by the command

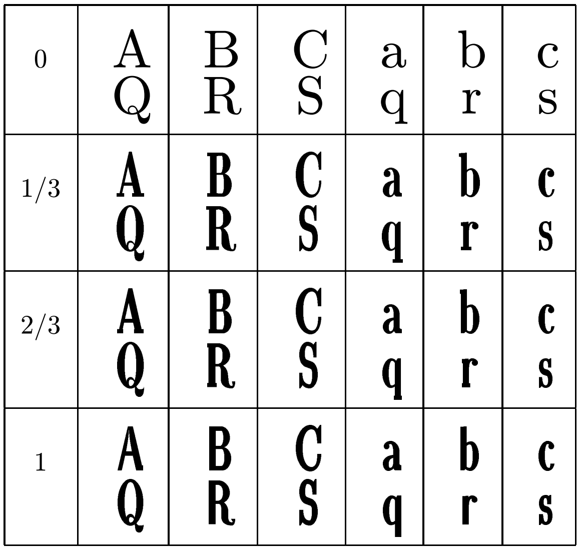
\FontMFF[*fntscaled*]{*fntcmd*}{*filename*}{*size*}

The command *fntcmd* will switch to the desired font as is done by the LATEX commands `\bf`, `\sl`, `\sf`, etc. The file *filename*`.mf` will contain the METAFONT source for a new font (please, *never* use the font name which is already used in CM or DC or some other standard fonts!). The parameter *size* specifies the design size of the new font, and the optional parameter *fntscaled* specifies the additional scaling of this font in LATEX document. Using the commands described in the following sections the user can vary the shape of the font characters in a wide range.

It is interesting to compare the possibilities of this simplest form of parametrization of CMR fonts and the PostScript vector fonts. The nearly-proportional changing of the font dimensions with respect to the magnification parameter is the analog of the linear scaling of the PostScript fonts. The non-linear relationship of the inter-character spacing from the font size imitates the *tracking* mechanism implemented in PostScript fonts (which is not taken into account in most cases by text processors). The fact that the ratio height/width is a non-constant (and non-linear) function of the font size is a serious advantage of these pseudo-CMR fonts in comparison with the linearly scaled PostScript fonts since it enables one to make the font proportions more suitable for the human eye (it is well known that for good eye recognition, small letters are to be more expanded and have greater inter-character spacing).

There are at least *two* ready-to-use font approximations available from CTAN. The first one is the METAFONT SAUTER font package [2]. It uses the smooth functions composed from constant, linear and quadratic pieces which are constructed so that for *canonical* font sizes they produce nearly the same `*.mf` files as the ones used by the original *Computer Modern* typefaces. Although the latest version of SAUTER is dated 1992, and in 1995 the parameters of *Computer Modern* fonts were again slightly changed, it seems still to be the most reliable source of the fonts with intermediate design sizes.

The other approximation is realized in DC and TC fonts by by Jörg Knappen and Norbert Schwarz [3]. It is based on cubic splines — Lagrange cubic splines or canonical cubic splines — using the parameters of *Computer Modern* typefaces as the base points. Although generally piecewise-cubic functions produce good quality approximations, it is not so with the data extracted from *Computer Modern* METAFONT files. The plots of the parameters, with respect to the design size, are “noisy” functions with some abrupt jumps since these parameters were selected manually to optimize the font shape, not the mathematical plots. As a result the cubic smooth approximations obey parasitic local minima and maxima and do not work far outside the range of design sizes used as the base data points. The DC and TC fonts with intermediate font sizes are visually good even with these “mathematical” defects, but for some specific reasons these defects could work badly when implemented in MFF.STY.

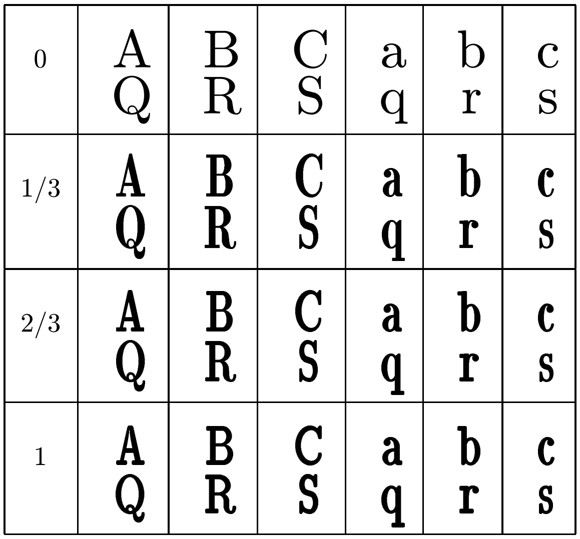


0	A Q	B R	C S	a q	b r	c s
1/3	A Q	B R	C S	a q	b r	c s
2/3	A Q	B R	C S	a q	b r	c s
1	A Q	B R	C S	a q	b r	c s

Figure 1: CMSS series

The first version of MFF.STY was based on piecewise-linear and piecewise-cubic (Lagrange splines) functions using *Computer Modern* typefaces as the reference data. To eliminate the parasitic local minima and maxima, some data points were slightly changed, and new data points were added to guarantee a good behaviour of the approximating expressions outside the range `5pt – 17.28pt`. The current version of MFF.STY is based on the SAUTER-type approximation with some modifications (especially for `cmff` and `cmfib` fonts).[2] As an option the piecewise-linear and the piecewise-cubic approximations based on DC data could be be included in futher versions of MFF.STY while the variated *Computer Modern* data used in the intermediate versions becames more-or-less obsolete.

[2] The reason to use SAUTER was the following: it is not a good idea to modify voluntary the original *Computer Modern* parameters.



0	A Q	B R	C S	a q	b r	c s
1/3	A Q	B R	C S	a q	b r	c s
2/3	A Q	B R	C S	a q	b r	c s
1	A Q	B R	C S	a q	b r	c s

Figure 2: CMTT series

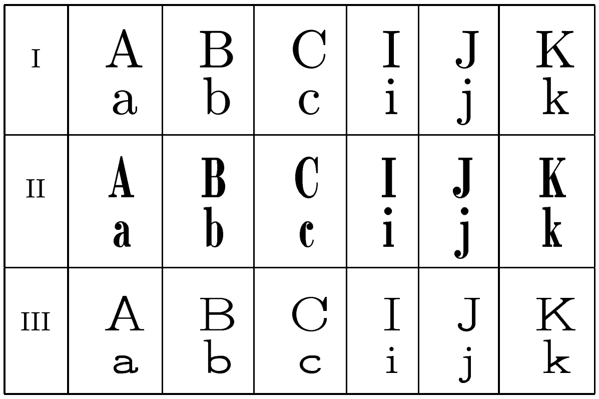
Mixture of independent fonts

The *Computer Modern* fonts roman, **bold**, *slanted*, sans serif, `typewriter`, funny, dunhill, and quotation use the same METAFONT programs but with different values for font parameters. For each font series it is possible to construct the smooth approximations (similar to CMR font approximation mentioned in the previous section) which enables creation of a METAFONT header file with arbitrary design size.

Suppose that there *are* such smooth approximations, and it is possible to calculate for some fixed design size the parameters of *legal* CMR, CMBX, CMTT and CMSS fonts. All these fonts are generated by METAFONT without errors, and it can be supposed that the *weighted sum* of the parameters corresponding to these fonts also can be processed by METAFONT without error messages — at least we can expect it with good probability.

The CMTT fonts have nearly rectangular serifs, nearly no contrast between thin and thick lines, and are a little compressed in the vertical direction. The CMSS fonts have no serifs at all, their width is less than that of CMR fonts, and, although they also have no contrast, the thickness of their lines is greater as compared with CMTT. Other fonts have their own specific features, but in spite of this fact they can be "added" together — at least mathematically. The resulting font is no longer CMR, CMBX, CMTT, etc., but is something intermediate with a unique shape.

The CMBX fonts can be subdivided (to some extend artificially) into *two* independent font series — one for "boldness" (i.e., *weight*) and one for "extension" (i.e., *width*). It makes the total scheme more closely related to the NFSS realized in LaTeX2$_\epsilon$. So in MFF.STY the CMBX font sequence is decomposed into CMB′ (fonts which are as bold as CMBX and as wide as CMR, but which are different from the standard CMB10 font) and CMX (fonts which are as wide as CMBX and as bold as CMR).



I	A a	B b	C c	I i	J j	K k
II	A a	B b	C c	I i	J j	K k
III	A a	B b	C c	I i	J j	K k

Figure 3: Font Modifications

The mixture of fonts is performed by the command `\MFFcompose`$\{\alpha_1\}\{\alpha_2\}\ldots\{\alpha_6\}$ where $\alpha_1 - \alpha_6$ are some numerical values. The value α_1 corresponds to CMB, α_2 to CMX, α_3 to CMSS (sans serif font), α_4 to CMTT (typewriter font), α_5 to CMFIB ("Fibonacci" font), and α_6 to CMFF (funny font). If some parameter has the numerical value p_{cmr} for CMR font, p_{cmb} for CMB font with the same design size, etc., the *mixture* value is calculated as

$$\begin{aligned} p_* \quad = \quad & p_{cmr} + \alpha_1\,(p_{cmb} - p_{cmr}) + \alpha_2\,(p_{cmx} - p_{cmr}) \\ & +\alpha_3\,(p_{cmss} - p_{cmr}) + \alpha_4\,(p_{cmtt} - p_{cmr}) \\ & +\alpha_5\,(p_{cmfib} - p_{cmr}) + \alpha_6\,(p_{cmff} - p_{cmr}) \end{aligned}$$

It enables one, for example, to make the font "less bold" than CMR or "more bold" and "more extended" than CMBX by assignment of values which are less than 0 or greater than 1, and to create the "mutant" combinations of nearly incompatible font families. In Fig. 1 the result of a mixture of `cmr10` and `cmss10` is shown, and in Fig. 2 a similar series is constructed for `cmr10` and `cmtt10`.

Manual font modification

In addition to the weighted mixture of font ingredients it is possible to vary some font parameters which are responsible for the specific effects.

The inclination of the characters depends on the single font parameter `slant#`, and its value can be set explicitly as a ratio $\Delta y/\Delta x$ or as the inclination angle (specified in degrees).

The *width* of the CMR font can be varied by mixing with the CMX font, but it also can be scaled explicitly by some factor specified by the user. Although this scaling skips some fine tuning of the font parameters, it can be advantageous to use it in some situations. For example, if we deal with CMTT fonts or CMSS fonts, mixing with CMX results also in some variation of the character shapes which can have an undesirable effect.

Similarly, the *weight*, i.e., the "boldness" of the characters, which can be controlled by mixing with CMB, can be defined also as a direct scaling of the font parameters which are responsible for this effect. The weight of "thin lines" and "thick lines" can be scaled independently, and in addition the user can specify explicitly the *contrast*— i.e., the ratio between thick and thin strokes of the characters.

The *height* of the vertical elements of the characters can be varied by the user with greater flexibility. That is, it is possible to scale independently

- the general height and the depth of the characters;
- the height of the capital characters, brackets, digits, etc., and the ascenders of the characters like 'b', 't';
- the depth of commas and the ascenders of the characters like 'Q', 'y';
- the height of digits and the position of the horizontal bar for mathematical signs like +, −.

If several height factors are specified, their effect is combined. For example, the font cmdunh10 can be derived *exactly* from cmr10 by proper specification of all these factors.

Finally, the special scaling factors can be used to produce the special effects:

- scale the fine connection between thin and thick lines in 'h', 'm', 'n';
- scale the thickness of sharp corners in letters 'A', 'V', 'w';
- scale the diameters of dots in 'i', ':' and bulbs in 'a', 'c';
- scale the curvature of the serif footnotes.

and the logical flags can control the level of ligatures and to switch on/off *square dots*, *sans serif* mode, *monospace* mode, etc.

Fig. 3 demonstrates the examples of such modifications:

- font (I) is the standard cmr10 scaled at 1.8 times,
- font (II) differs from cmr10 by scaling the width by 0.8, the height by 1.2 and the width of bold lines by 1.5;
- font (III) differs from cmr10 by scaling the width by 1.2, the height by 0.8, the ascenders and descenders by 1.25, the width of bold lines by 0.8 and the width of thin lines by 2.4.

All the commands which allow performance of these modifications are described in MFF.STY manual in details.

Automatic check of font parameters

Each specification of MFF.STY parameters produces a unique font which belongs to the CMR/MF (Computer Modern Roman Master Font) font family and with a unique name specified by the user. Not all of these fonts are too pleasant, and not each variation of the parameters result to a font which is well-distinguished from the others. But at least it is an interesting toy for font maniacs.

There is a list of mutual relations between font parameters which are assumed implicitly in METAFONT programs for *Computer Modern* typefaces [1]. Although in reality most METAFONT source files *violate* these conditions, it is safer if the font parameters calculated by MFF.STY satisfy them. The command \MFFcheck sets the mode when these conditions are checked and the variable values are corrected if necessary. Nethertheless, several interesting effects can be achieved only without such automatic correction. The mode of automatic checking can be switched off by the command \MFFcheck.

Switch to other font classes

The LaTeX 2ε NFSS classifies TeX font families in a way which is different from the logical structure of METAFONT programs. That is, the *italic* and SMALL CAPS are at the same family roman, together with **bold** and *slanted* fonts, although they are produced by a different driver files. Similarly, roman, typewriter and sans serif fonts are different families while they are generated by the same METAFONT script and their parameters can be varied so that one family is smoothly converted to another family.

In MFF.STY, there is no sharp boundary between roman, **bold**, *slanted*, typewriter, sans serif, quotation, funny and dunhill fonts — each font is smoothly converted to another one, while *italic* and SMALL CAPS fonts are quite different. The MFF.STY macros assign different *classes* to these fonts to distinguish such differences from the *font families* used in NFSS. The following font classes can be used:

CMR Computer Modern Roman;

CMTI Computer Modern Text Italic;

CMCSC Computer Modern Small Caps;

CMRZ *CMZ* Computer Modern Roman/Cyrillic by N. Glonti and A. Samarin;

CMRIZ *CMZ* Computer Modern Text Italic/Cyrillic;

CMCCSC *CMZ* Computer Modern Small Caps/Cyrillic;

LHR *LH* Computer Modern Roman/Cyrillic by O. Lapko and A. Khodulev;

LHTI *LH* Computer Modern Text Italic/Cyrillic;

LHCSC *LH* Computer Modern Small Caps/Cyrillic.

The interface for DC fonts [3] is under development.

The set of font classes can be extended easily when the METAFONT program is based on the same set of parameters as *Computer Modern* fonts: The only thing to do is to specify the macro which writes the `font_identifier` value and the operator `generate` with the corresponding file name.

Acknowledgements

The authors would like to express their warmest thanks to Kees van der Laan for organization and realization of the *Euro-Bus* project which enabled the Russian delegation to take part in *EuroTEX-95*, and to Ph.Taylor for his activity in breaking down the barriers between West and East.

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07-30-007).

References

[1] Donald E. Knuth. *Computer Modern Typefaces*, (*Computers & Typesetting* series). Addison-Wesley, 1986.

[2] John Sauter. *Building Computer Modern fonts. TUGboat*, **7** (1986), pp. 151 – 152.

[3] Jörg Knappen. *The release 1.2 of the Cork encoded DC fonts and the text companion symbol fonts.* Proceedings of the 9th EuroTEX Conference, Arnhem, 1995.

[4] A. Khodulev and I. Mahovaya. *On TEX experience in MIR Publishers.* Proceedings of the 7th EuroTEX Conference, Prague, 1992.

[5] O. Lapko. *MAKEFONT as a part of CyrTUG – EmTEX package.* Proceedings of the 8th EuroTEX Conference, Gdańsk, 1994.

VFComb 1.3 — the program which simplifies the virtual font management

A.S. Berdnikov
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
`berd@ianin.spb.su`

S.B. Turtia
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
`turtia@ianin.spb.su`

The MS DOS program *VFComb* enables one to simplify the design of the virtual fonts.[1][1] Its main purpose was to facilitate the integration of CM fonts with Cyrillic LL fonts created by O. Lapko and A. Khodulev [2, 3] but it can be used for other applications too. It uses the information from `.tfm` files (converted to ASCII form by *TFtoPL*) and the ASCII data files created by the User on its input, and produces the `.vpl` file on its output (the `.vpl` file can be converted later to the virtual font using *VPtoVF*). The characteristic feature of the program is that it can assemble the ligature tables and metric information from various fonts and combine it with the user-defined metric information and ligature/kerning data. *VFComb* supports the full syntax of `.pl` files and `.vpl` files as it was defined by D.E. Knuth and adds new commands like symbolic variables or conditional operators, which simplifies the creation and the debugging of the virtual fonts.

The description of the previous version 1.2 (which is the first version distributed far outside the home computer) can be found in [4]. This version has only the Russian manual which prevents its wide distribution among TEX community. The current version *has* the English manual, but except this "new feature" a lot of additional improvements are added. The main features of the program are described below while the complete information can be found inside the manual.

The program will be put on the CTAN archives following *TUG'96* together with the source code and will be available to TEX community on a freeware basis. Since this MS DOS program is written on *Borland Pascal* and uses some specific features of this language, it is hardly portable to any other platform "as is", but it is not too difficult to transfer it to portable ANSI C (volunteers are welcome).

Virtual fonts for TEX formats with national alphabets

Although everything which can be done by *VFComb* could be realized also by explicit usage of PL and VPL file syntax (as well as everything which can be done manually by `.pl` and `.vpl` files can be done with *VFComb*), some typical operations with the virtual fonts are performed with its help *easier* than by manual editing of `.pl` and `.vpl` The typical problem of this type is the adaptation of standard TEX formats to national alphabets — this problem is especially important for Cyrillic alphabets since most Cyrillic letters *cannot* be created as the combination of the Latin (English) letters with some accents.

The standard solution of this problem is to combine the English part taken from Computer Modern family with the national fonts which extend the Computer Modern family and which contain in the upper part of ASCII table (codes 128 – 255) the national symbols. The best way how to do it is to create the *virtual font* whose lower part refers to original CM fonts, and upper part refers to the national fonts — it is just the way which was recommended by D. Knuth.[1] The advantage of this approach is that it is possible to keep the changes in CM fonts and in national fonts separately, and in addition, it is possible to economize disk space since it is not necessary to keep *two* copies of each Computer Modern character — one as the original CM font which is necessary for original TEX formats, and the second one as the lower part in the combined national font.

The combination of the lower part of one font and the upper part of another font, or even the joining all the characters from one font and all

[1] It is assumed that you are familiar with the virtual fonts. If it is not so, it is highly recommended that you read [1] before proceeding further.

the characters from another font (provided that *no* character code is encountered twice) can be done by *VFComb* using few commands. If some characters are to be discarded from the font or moved to the different positions of the ASCII table, it does not makes the command script more complicated. The resulting virtual font contains proper metric information for each character borrowed from the source metric information and the correct mapping of the characters into individual real fonts.

The similar operation can be performed also by the program *TFMerge* (IHEP TEXware, Protvino), but *VFComb* performs additional operations. That is, in addition to joining metric and ligature/kerning information from each font into one virtual font, it is necessary to add *cross*-ligature and *cross*-kerning information for the pairs of characters taken from different fonts. *VFComb* enables one to add metric, ligature and kerning data taken from its script file (which makes the original script a little bit more complicated). The important feature is that this additional data can contain *variables* and *logical structures*, from which one can generate the whole CM family of the virtual typefaces with national characters using just the same pseudo-program written on *VFComb* command language.

Except the operations described above, *VFComb* is capable of performing the following operations if it is specified by the user in its script:

- discard the ligature tables of some real fonts;
- include in the virtual font the full ligature table of the real font;
- include in the virtual font only those characters which are declared explicitly in user-defined data, and discard the elements of the ligature tables which correspond to non-included characters of the real font;
- automatically add to the virtual fonts the characters which are not included explicitly by the user but which are joined with the already included characters through ligature table data, or by specifications `NEXTLARGER` and `VARCHAR`.

These features allow creation of the desired virtual fonts for national alphabets with less effort and with more reliability than by manual manipulations with `.pl` and `.vpl` files.

Virtual fonts for colored printing

Another problem is the application of the virtual fonts to multicolored printing. Suppose that it is necessary to print text where different characters have different colors. From TEX-compiler's point of view it means that the characters with different colors are assigned to different fonts, and it is a task for the `dvi` driver to decide how to print these fonts in desired colors.

The colored printing is collected from the overlapped sheets where each sheet of text or graphics is printed by individual monocolor passes. To make the templates for monocolor printing it is necessary to organize the output of the `.dvi` file so that in one pass only yellow characters are printed, in another pass only blue characters are printed, etc., while the characters which have the green color are to be printed twice — in blue as well as in yellow. The easiest way to teach `dvi` driver how to do it is to create different subdirectories with virtual fonts — one subdirectory for each elementary color. The virtual font files placed in the subdirectory for yellow printing (which corresponds to the yellow fonts) will refer to the actual `*.pk` files if and only if the yellow color is assigned to this character — otherwise it will refer to *empty* character. The subdirectories for other colors are organized similarly. As soon as the yellow printing is performed, the `dvi` driver is configured so that it takes the virtual fonts from the "yellow" subdirectory, and for the output in other colors a corresponding reconfiguration of the `dvi` driver is performed.

If the mapping of the empty characters into the `dummy` font is performed, it results to the wrong behaviour of the `dvi` driver: the characters in the `dummy` font have zero size, and this means that the next character after the empty character is shifted to the left (as compared with the desired behaviour) a the distance equal to the width of the skipped character. To prevent this effect it is necessary to insert into the virtual font the explicit `dvi` commands which move the current output position to the right by the distance representing the width of the skipped character. This operation is performed by *VFComb* by a single command: the user assigns the attribute `NULLCHARACTER` to the corresponding real font, and for the characters of this font the *empty* mapping will be performed instead of mapping the real font characters.

Substitution of CM fonts instead of PostScript fonts for *DVI* Viewers

The next problem where the usage of the virtual fonts is advantageous is the visualization of the document which was compiled using PostScript fonts. Generally, the screen viewer *cannot* process the PostScript characters, and it is necessary to remap the PostScript font characters into some `.pk` font which *can* be displayed by the viewer — say, some typefaces from the Computer Modern family.

Such remapping can be performed using virtual font mechanism, but if it is done without special precautions the screen view can be far from the printed output. The reason is that CM characters have a width different from the PostScript font (the fact that they have a different graphical image is not so essential). As in the previous case, the screen output will be shifted to the left on the distance which is the difference between the width of the PostScript character and the CM character if no special precautions are taken. To make the correct output, it is necessary to add to the virtual font the explicit `dvi` commands which correct the current output position.

To make corresponding virtual font automatically, *VFComb* allows the user to specify for the real fonts *two* `.pl` files with the metric information: the first one is for the nominal characters which are used by TeX to compile the `.dvi` file (in our case it is the PostScript `.afm` file converted to `.tfm` format), and the second one is for the real characters which are used when the `.dvi` file is displayed (or printed). If such information is specified by the user, the commands to correct properly the current output position are inserted into the virtual font.

This operation works if both fonts have the same coding scheme – namely, the characters used by TeX and the characters used by `dvi` viewer have the same code value. If not, the operation of remapping inside an already-mapped font is required, and this could be very complicated and result in a very complicated scheme of virtual font generation. To solve this problem, it is assumed that the correct metric information for the "true" font (i.e., for the font used in compilation of the `.dvi` file), is already available. The special operators in *VFComb* enable the user to load this information and to correct the proper character width.

Other features

The other features incorporated in *VFComb* 1.3 are:

- improved syntax for *VFComb* commands;
- improved logical operators:
- implementation of string variables in *VFComb* script files;
- specification of variable values in command line among other parameters;
- specification of the names of the real and virtual fonts inside *VFComb* scripts instead of command line;
- correction of some bugs, including the obligatory conversion to uppercase all input characters together with the font names;
- automatic computation of the metric information for the characters composed from user-defined `dvi` commands.

All of these improvements ensure easier use of the program. For example, after implementation of the new features, the generation of the virtual fonts for the LL/LH family (used in the CyrTUG Cyrillic version of TeX/LaTeX/AMS-TeX) is performed using just *one* script file of *VFComb* instead multiple header files (see the example below).

Comparison with *FontInst*

There is another package for manipulating with virtual fonts — namely, *FontInst* by Alan Jeffrey. Although there are many similar features between *FontInst* and *VFComb*, these tools are designed to solve different tasks.

The main purpose of *FontInst* is to create new font families for LaTeX 2_ε using existing PostScript fonts. *FontInst* contains high-level operators which enable one to perform this task in several commands, and it is written totally in TeX, which guarantees its high portability. In addition, *FontInst* contains special TeX macro commands which enable the user to do nearly anything with virtual fonts, without direct editing of `.vpl` files; but, in this case the "program" written in *FontInst* commands may be comparable in length to the `.vpl` file.

VFComb is designad to solve different problems — it was designed mainly to simplify the integration of new alphabets into TeX so that the Latin part of Computer Modern Typefaces is left unchanged. It *cannot* read `.afm` files and it *cannot* create `.fd` files. If you wish to use it to create virtual fonts for PostScript, you can do so, but you need some additional utilities (`afm2tfm`) and many more manual operations than you need with *FontInst*.

Similarly, you can make virtual fonts like that created by *VFComb* using *FontInst* as well, but this will require manual manipulations which are comparable to the direct editing of `.vpl` files. Although one can expect in future more convergence between *FontInst* and *VFComb*, currently these programs do not intersect in this respect. If you wish to create virtual fonts for font families derived from PostScript, please use *FontInst*, and you will economize a lot of your time. If you wish to make virtual fonts which solves any of the problems described in this paper, *VFComb* may be more suitable.

Example

The syntax of *VFComb* commands is similar to that of `.vpl` files. It means that it is more suitable for computers than for people. The following example is extracted from the file `lhfonts.tbf` used to generate the full family of Cyrillic virtual fonts demonstrates the *VFComb* syntax.

Suppose that the program *VFComb* is called as

```
vfcomb lhfonts.tbf font=lhr10
```

which means that the script file `lhfonts.tbf` is used as a source of *VFComb* commands, and that the user asks to generate the virtual font `lhr10` which is a combination of the *Latin part* taken from `cmr10` and the *Cyrillic part* taken from `llr10`. The fact that the virtual font has the name `lhr10` and that it is composed from `cmr10` and `llr10` is described inside the script file `lhfonts.tbf`. Actually the command line shown above specifies only the fact that the string variable `font` is defined with the initial value `lhr10`, before the script file `lhfonts.tbf` is processed.

The head of the file `lhfonts.tbf` contains the commands

```
(IF-DEF font)
    (VARIABLE (STRING FONT @V font))
(ELSE)
    (MESSAGE Variable FONT is not defined)
    (HALTPGM)
(ENDIF)
```

These commands check that the user does not forget to specify the variable `font=...` at the command line, and assign its value to the string variable `FONT` (uppercase and lowercase letters are distinguished by variable names, and the prefix `@V` means the value of the string variable).

Similarly, the commands

```
(IF-DEF type)
     (VARIABLE (STRING TYPE @V type))
(ELSE)
     (VARIABLE (STRING TYPE NEW))
     (MESSAGE TYPE = NEW is assumed)
(ENDIF)
```

analyzes the contents of the variable `type` if it is specified at the command line, and assigns the default value `NEW` if there is no expression `type=...` at the command line.

The commands

```
(VTITLE CyrTUG freeware LH font family)
(OUTPUT @V FONT)
(HEADER (FONT D 1))
(MAPFONT D 0 (LOWPART))
(MAPFONT D 1 (HIGHPART))
```

specify the title of the virtual font and the name of the output virtual font. The *header* of the virtual font is similar to that of the font `D 1` if there are no other commands which modify the contents of the header. The commands `MAPFONT` state that two fonts are used to create the virtual fonts where the characters 0 – 127 are taken from one font and the characters 128 – 255 are taken from the other font.

The actual names of the real fonts '`0`' and '`1`' are specified after the following analysis of the contents of the variable `FONT`:

```
  (IFS-CASE @F @V FONT)
  (CASE LHR)
     (MAPFONT D 0 (FONTNAME @+ CMR @P @V FONT))
     (MAPFONT D 1 (FONTNAME @+ LLR @P @V FONT))
     (HEADER (FAMILY LHR))
     (VARIABLE (BYTE FLKERN 0))
     (BREAK)
  (CASE LHTI)
    (MAPFONT D 0 (FONTNAME @+ CMTI @P @V FONT))
    (MAPFONT D 1 (FONTNAME @+ LLTI @P @V FONT))
    (HEADER (FAMILY LHTI))
    (VARIABLE (BYTE FLKERN 1))
    (BREAK)
......
(ELSE)
   (MESSAGE Unknown font family @F @V FONT)
   (HALTPGM)
(ENDIF)
```

Here the prefix commands `@F @V FONT` extract the non-digital component of the font name which is analysed by the `CASE` operators (note that the comparison of text strings by the operator `IFS-CASE` does not distinguish between uppercase and lowercase letters). If the non-digital font component is equal to `LHR` (as it is in our case for `font=lhr10`), the first font gets the name `CMR...`, and the second font gets the name `LLR...` where the dots are substituted by the font design size: the prefix `@P @V FONT` extracts the value '`10`' from `lhr10` (like `@F @V FONT` extracts '`lhr`' from the same value), and the prefix `@+` performs the concatenation of two strings. In the same command block, the header field `FAMILY` gets the value `LHR`, and the byte variable `FLKERN` gets the value 0 (it is used later to construct the additional entries for ligature table which corresponds to the ligature/kerning data for the characters taken from different real fonts). The font names `LHTI...` are analyzed similarly, and analogous commands (skipped here) are specified for every legal font family.

The following commands specify the symbolic names for some Cyrillic letters which are used later in the ligature tables. Note the the coding scheme specified here depends on the value of the expression `type=...` specified in the command line:

```
  (IFS-EQ @V TYPE OLD)
     (VARIABLE
        (BYTE CYR_open_quote  D 243)
        (BYTE CYR_close_quote D 244)
     (BYTE CYR_Number      D 245)
   )
(ELSE) (COMMENT TYPE = NEW)
   (VARIABLE
      (BYTE CYR_open_quote  D 250)
      (BYTE CYR_close_quote D 251)
      (BYTE CYR_Number      D 252)
   )
(ENDIF)
(VARIABLE
  (BYTE CYR_GHE D 131)
  (BYTE CYR_ghe D 163)
  (BYTE CYR_ER  D 144)
  (BYTE CYR_er  D 224)
  ......
)
```

The following commands analyze the value of the variable FONT and assign the value for the real variable u# and for the logical variable monospace, which are used to construct the ligature data for the pairs of characters taken from different fonts:

```
(IFS-CASE @V FONT)
(CASE LHR5)
    (COMMENT Font LHR5)
    (VAR
       (REAL u# A/ A/ R 12.5 R 36 R 5)
       (BYTE monospace D 0)
    )
    (BREAK)
(CASE LHR6)
    (COMMENT Font LHR6)
    (VAR
       (REAL u# A/ A/ R 14 R 36 R 6)
       (BYTE monospace D 0)
    )
    (BREAK)
...
(ENDIF)
```

The arithmetic expressions in these commands are constructed using the "*Polish notation*" structure. That is, the prefix A/ is the division of the two arguments which follow it (A+ is addition, A- is subtraction, A* is multiplication and R is a real number), and each of these arguments can be the arithmetic expression starting with an arithmetic prefix as well. For example, the value of the variable u# for the font LHR5 is equal to A/ A/ R 12.5 R 36 R 5 $= ((12.5/36)/5) = 0.069444444444$.

Finally, the commands which specify the additional ligature data for the pairs of characters taken from different fonts are added:

```
(IF-EQ V monospace D 0)
(IF-CASE V FLKERN)
(CASE D 0) (COMMENT KERN Roman)
   (VARIABLE
      (REAL k#   A* R -0.5 V u#)
      (REAL kk#  A* R -1.5 V u#)
      (REAL kkk# A* R -2.0 V u#)
      ......
   )
   (LIGTABLE
      (LABEL V CYR_GHE)
      (KRN C . V kk#)
      (KRN C , V kk#)
      (KRN C : V kk#)
      (KRN C ; V kk#)
      (STOP)
   )
   (LIGTABLE
      (LABEL V CYR_ER)
      (KRN C . V kk#)
      .....
   )
   .....
   (BREAK)

(CASE D 1) (COMMENT KERN Italic)
   ......
(ENDIF)
(ENDIF)
```

The syntax of these commands is similar to that of .vpl files except that the variable values are used (some variables are calculated depending on the variable u# defined above) and the logical operators analyze what data should be included depending on the current values of the variables FLKERN and monospace.

Acknowledgements

All new improvements of *VFComb* (except the English manual) are the result of the contacts and discussions which were held during the *EuroTEX-95* meeting. So we would like to thank Kees van der Laan for his giant efforts to organise the visit to the *EuroTEX-95* by the delegation from Russia and for his patient attention to Russian colleagues before, during and after the *EuroTEX-95*.

It is not so easy to recall *all* participants of this conference whose opinions made an impact on the preparation of the new version of *VFComb*. Among other persons we would like to thank Phil Taylor and S. Znamensky for their valuable suggestions which led to improvements in the program. We would like also to thank O.A. Lapko, S.A. Strelkov and I.A. Makhovaya for their efforts spent on the Cyrillic TEX project which actually inspired our work.

This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07-30-007).

References

[1] D. Knuth, "Virtual Fonts: More Fun for Grand

Wizards", TUGBoat **11** (1993), No. 1, pp.13–23.

[2] A. Khodulev, I. Mahovaya. "On TEX experience of MIR Publishers", Proceedings of the 7th EuroTEX Conference, Prague, 1992.

[3] O. Lapko. "MAKEFONT as a part of CyrTUG–EmTEX package", Proceedings of the 8th EuroTEX Conference, Gdańsk, 1994.

[4] A.S. Berdnikov, S.B. Turtia. "*VFComb* — a program for design of virtual fonts", Proceedings of the 9th EuroTEX Conference, Arnhem, 1995.

ΩTimes and ΩHelvetica Fonts Under Development: Step One

Yannis Haralambous
Atelier Fluxus Virus, 187, rue Nationale, F-59 800 Lille, France
Yannis.Haralambous@univ-lille1.fr

John Plaice
Département d'informatique, Université Laval, Ste-Foy (Québec) Canada G1K 7P4
John.Plaice@ift.ulaval.ca

The Truth Is Out There
— Chris CARTER, *The X-Files* (1993)

Introduction

ΩTimes and ΩHelvetica will be public domain virtual Times- and Helvetica-like fonts based upon real PostScript fonts, which we call "Glyph Containers". They will contain all necessary characters for typesetting efficiently (that is, with TEX quality) in all languages and systems using the Latin, Greek, Cyrillic, Arabic, Hebrew and Tifinagh alphabets and their derivatives. All Unicode characters will be covered, although the set of glyphs of our Ω fonts will not be limited to these; after all, our goal is high-quality typography, which requires more glyphs than would be required for mere information interchange.

Other alphabets will follow (the obvious first candidates are Coptic, Armenian and Georgian) as well as mathematical symbols, dingbats, etc.

Why PostScript instead of METAFONT? METAFONT is the ultimate tool for font development. Extending Computer Modern fonts to the Unicode encoding is still one of our goals. We have started developing a set of fonts we call "Unicode Computer Modern" fonts, using techniques such as Virtual METAFONT (virtual fonts are created directly by METAFONT using the `gftotxt` utility for text output). Nevertheless, we realized that the task of writing METAFONT code for some thousands of characters (including code for typewriter style) is a tremendous task, which will take several years.

So we have decided to take a small break from METAFONTing, and to develop in a limited time period PostScript fonts that will cover a maximum number of languages and will give the TEX community a good reason to switch to Ω.

Why Times and Helvetica? First of all because, after Computer Modern, they are the most widely used fonts in the TEX community. Many journals and publishers request that their texts be typeset in Times; Helvetica (especially the bold series) is often used as a titling font. Like Computer Modern, Times is a very neutral font that can be used in a wide range of documents, ranging from poetry to technical documentation...

It would surely be more fun to prepare a Bembo- or Stempel Garamond-like font for the serifs part and a Gill Sans- or Univers-like one for the sans-serifs part; but these can hardly be used in the scientific/technical area, and that's perhaps where TEX (and hence potentially Ω) is used the most.

When will the Ω fonts be finished? The development of ΩTimes and ΩHelvetica fonts is divided into four steps:

1. Drawing of PostScript outlines and packaging of Glyph Container fonts.
2. Development of virtual code, based on the real fonts of step one.
3. Kerning of virtual fonts.
4. Development of LATEX code and Ω Translation Processes necessary for the use of these fonts.

For the time being (June 1996) we have done the biggest part of step one, and this is what we present in this paper. We hope to have finished with steps two, three and four before the next teTEX CD-ROM in December 1996.

We want your support! Please keep in mind:

> The choice and shapes of glyphs presented in this paper are only a first attempt. We need *your* feedback to improve them, so that *you* can use them efficiently.

In the tables we present only Times family and medium series fonts (except in the case of Tifinagh, which is also presented in the Helvetica family). Up-to-date tables of the remaining fonts can be consulted on our Ω WWW server

`http://www.ens.fr/omega`

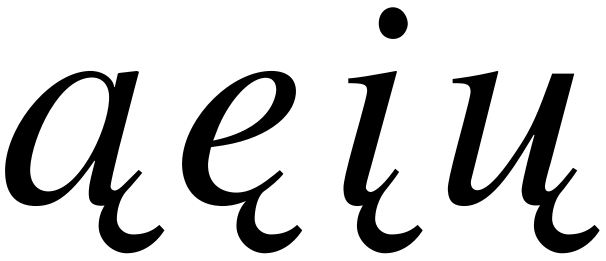


Figure 1: Italic-style letters with ogonek (Polish and Lithuanian)

You can also retrieve the PostScript code from the same address, or from

`ftp://ftp.ens.fr/pub/tex/yannis/omega`

General remarks on the fonts

To prevent confusion, the word "font" in this section is meant in the sense of the PostScript Type 1 font structure; and not of TeX text or math fonts: the fonts we describe in this paper will *never* be used directly for typesetting. Their *raison d'être* is to provide glyphs for the virtual Unicode+Typography Ω fonts which we will develop in steps two–four. Hence, there is no need to look in the tables for a 'é': this character will be assembled by the virtual font, using the glyphs of letter 'e' and of the acute accent.

The same stands for the letter 'c' with cedilla: it can be assembled out of the two corresponding glyphs; however, this is *not* true for letters with ogonek: the shape of the ogonek changes while it gets attached to the letter; that is why you find letters with ogonek in the Glyph Containers and not letters with cedilla. In Fig. 1 the reader can see examples of letters in italic style, carrying an ogonek accent.

In Fig. 2 the reader can find the general structure of the fonts:[1] On the left, the 16-bit Unicode+Typography virtual font, on the right a certain number of Glyph Containers, that is 8-bit PostScript fonts.

The reader will notice that a certain number of glyphs are repeated in the different Glyph Containers. This is because we want to minimize the number of Glyph Containers used for a single-alphabet text. For example, accents for all fonts are stored in Glyph Container "Common"; theoretically, to produce an acute-accented Latin letter and an acute-accented Cyrillic letter, one would use three Glyph Containers: one for the accent, and one for each alphabet. To avoid this, we store all accents relevant to a given alphabet, in the alphabet's Glyph Container. The same method is used for shapes that are similar in the different alphabets: Latin, Greek and Cyrillic alphabets share the letter 'A', Latin, IPA and Cyrillic alphabets share the letter 'a'.[2]

The "Common" Glyph Container

The "Common" Glyph Container, shown in Table 1, contains glyphs that will be used potentially in conjunction with all alphabets. These are described in the following subsections.

Punctuation, digits, editorial marks Special care has been taken to distinguish between "typographical" punctuation and "typewriter/computer terminal-derived" one: compare the typographical double quotes "" and the straight 'ASCII' ones ".

This table covers all Unicode punctuation marks from the ASCII and ISO 8859-1 tables as well as from the GENERAL PUNCTUATION table (`0x2010`–`0x2046`). We have not included a few punctuation marks specific to a single alphabet: Arabic asterisk, inverted comma and semicolon, Hebrew colon, Greek upper dot. These will be found in the corresponding Glyph Containers.

In Fig. 3 the reader can see how regular curly braces have been transformed into SQUARE BRACKETS WITH QUILL, by simply reflecting the central part of the brace.

Commonly adopted Latin alphabet derived symbols Symbols like © use Latin alphabet letters but are used in many non-Latin-alphabet based languages. Symbol № is an even stranger example: although the glyph 'N' does not exist in Cyrillic, this symbol is used mainly in Cyrillic-alphabet languages.

[1] In the figure, the reader will notice real font "Adobe Zapf Dingbats". In fact, the glyphs of this font have become Unicode characters (`0x2701`–`0x27be`) and we see no reason to redraw them since this font is widely available. Hence the virtual 16-bit font will also point to the standard Adobe Zapf Dingbats font.

[2] We will use a totally different approach when `dvips` and Adobe Acrobat are able to use 16-bit PostScript fonts. Instead of having many 'small' PostScript fonts and one 'big' virtual font, we will use a single 'big' PostScript font, in Unicode+Typography encoding. In that font, accented letters and repeated identical shapes will be obtained by the PostScript font technique of *composite characters*. This will allow Acrobat users to select and copy Unicode-encoded text directly from the document window.

Since the two techniques (16-bit + 8-bit real, vs. 16-bit composite real) will share the same `.tfm` metrics for characters, it will be possible to convert `.dvi` files from one format to the other, so that files obtained today by the first method will be "Acrobat-ready" later, whenever Acrobat switches to Unicode (hopefully soon!).

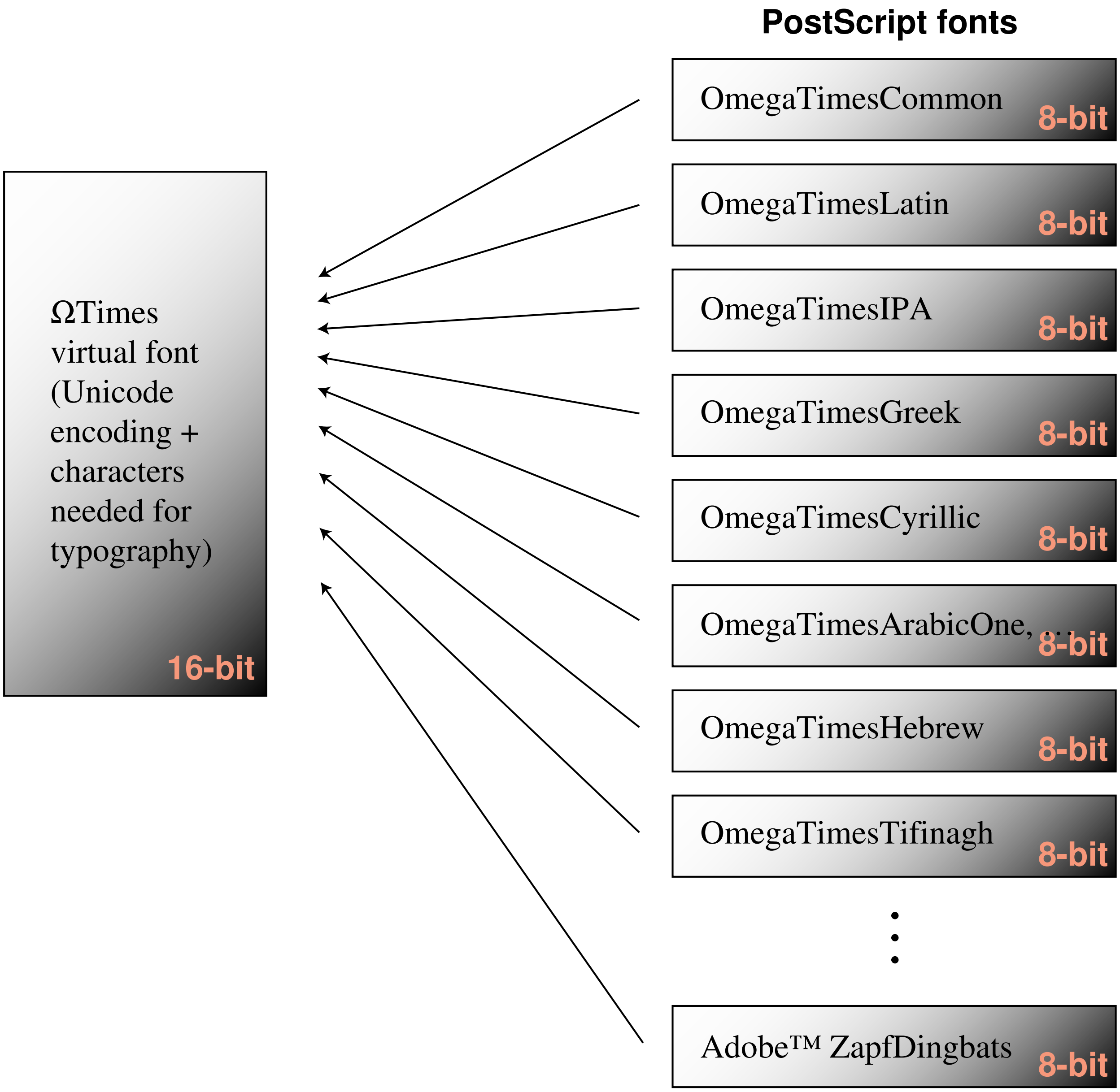


Figure 2: General structure of the ΩTimes fonts (idem for ΩHelvetica)

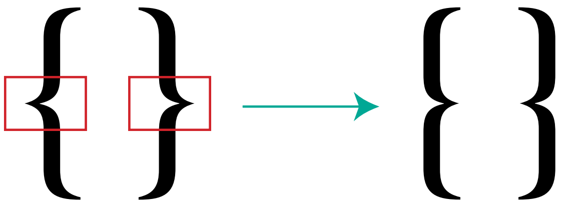


Figure 3: The LEFT and RIGHT SQUARE BRACKETS WITH QUILL were drawn by reflecting the central part of regular curly braces

In Fig. 4 the reader can see our small tribute to the GNU foundation: the “copyleft” symbol. We hope that this character will soon be included in the LETTER-LIKE SYMBOLS section of Unicode.

—Q. Why not use the `\reflectbox` macro to reflect the “copyright” glyph into a “copyleft” one?

—A. Because we want to treat “copyleft” as a separate character in the virtual font, which may be searched inside a `.dvi` or PDF file. In other cases, such as ᗺ, used in the AᗺBA logo and the cobar construction in Algebraic Topology, one can

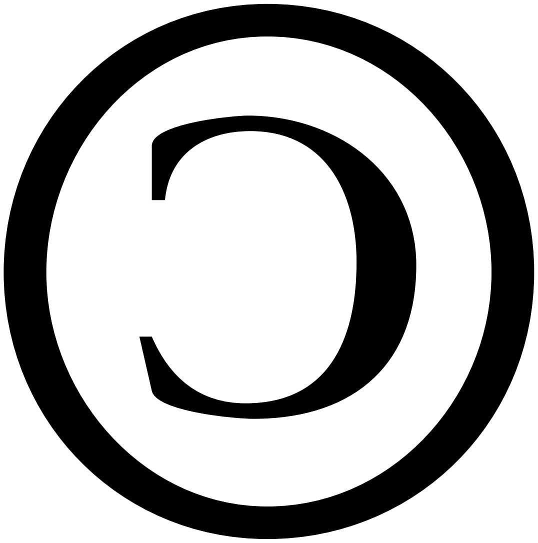


Figure 4: The GNU "copyleft" symbol

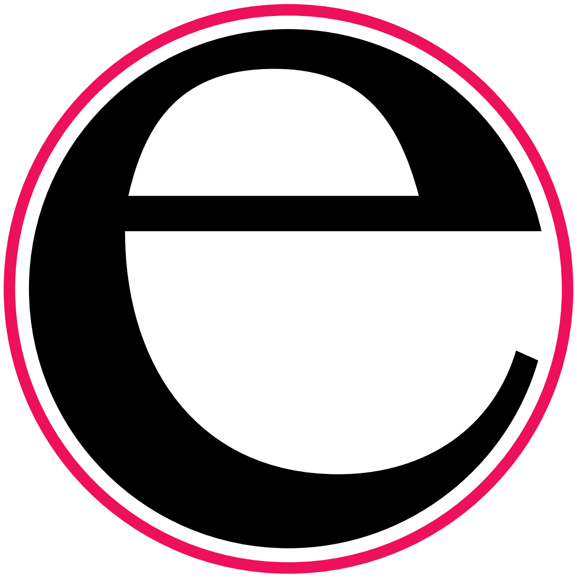


Figure 5: The ESTIMATED SYMBOL was drawn inside a perfect circle

easily use PostScript-manipulation macros without damage.

Finally, in Fig. 5 the reader can see how the ESTIMATED SYMBOL fits inside a perfect circle (in red, for readers of a color version of this paper).

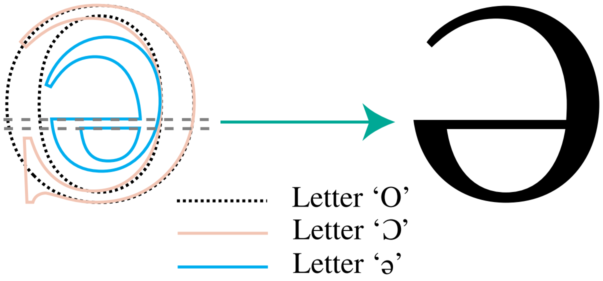


Figure 6: How shapes 'C', 'O' and 'e' were used for the design of the LATIN and CYRILLIC CAPITAL LETTER SHWA

Currency symbols Among currencies having proprietary symbols, there are strong and weaker ones. The strong ones have made it into ISO 8859-1 (you-know-who made it even into ASCII itself, and is used by a well-known typesetting language to enter math mode...), the remaining ones have found their home in the CURRENCY SYMBOLS section of Unicode.

We have included them all (even the Thai currency symbol ฿, which looks suspiciously Latin) in the "Common" Glyph Container. Note that the symbol ₣ for French Franc is virtually unknown in France...

Diacritics The zone `0xa0`–`0xe2` of the "Common" Glyph Container is dedicated to combining diacritics. These diacritics are supposed to be useful for more than one alphabet; whenever a diacritic belongs specifically to one alphabet, it has been included only in the corresponding Glyph Container (this is the case, for example, of Vietnamese double accents, Greek spirit+accent combinations, and Slavonic accents). Thanks to the diacritics in the "Common" Glyph Container we will be able to construct all LATIN EXTENDED ADDITIONAL Unicode characters (`0x1e00`–`0x1ef9`) virtually, by combining them with letters from the "Latin" Glyph Container. This Unicode region covers Welsh, Vietnamese and transcriptions of Indic and other languages.

The "Latin" Glyph Container

The "Latin" Glyph Container, shown in Table 2, contains glyphs of letters for Latin alphabet languages. These are described in the following subsections.

Letters for Latin alphabet languages All glyphs necessary to typeset Western and Central European, Nordic, Baltic, African languages, Vietnamese and Zhuang.

Some African characters are derived from the International Phonetic Alphabet. It is a fascinating challenge to design uppercase and italic-style forms for these characters (for example, see in Fig. 6 the

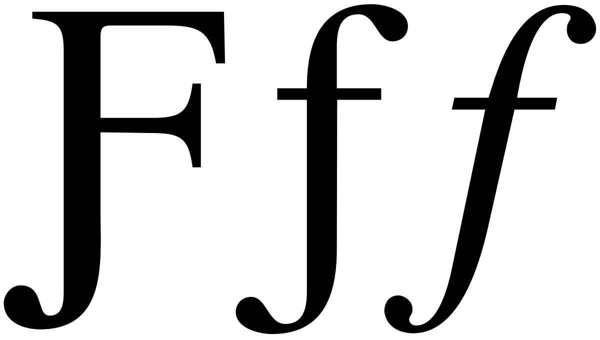


Figure 7: The African letter F WITH HOOK in roman upper- and lowercase, as well as in italic lowercase form

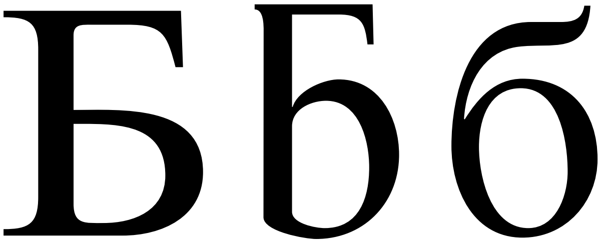


Figure 8: CAPITAL LETTER B WITH TOPBAR (which shares the same glyph as CYRILLIC CAPITAL LETTER BE) SMALL LETTER B WITH TOPBAR and CYRILLIC SMALL LETTER BE

uppercase version of letter ə letter which derives from the phonetic SHWA, and in Fig. 7 the straight uppercase and straight lowercase versions of an African letter that becomes a standard *f* in italic style).

Sometimes, although a Greek form (or a form derived from Greek) is used for the lowercase, the uppercase does not follow the Greek model: for example, the uppercase of African letters ɛ and ɣ (the former is 100% Greek, while the latter looks more like a "phonetic gamma") are Ɛ and Ɣ.

Inversely, sometimes a Greek form is used for the uppercase only: Σ is the uppercase form of the integral-like ʃ, a character derived from the IPA, for which one can hardly imagine an obvious uppercase form — taking a Greek letter for that purpose is the easiest solution.

It is quite interesting how the lowercase of the Latin letter Ƃ differs from that of the Cyrillic letter sharing the same glyph (see Fig. 8).

Special attention has been paid to the notorious Dutch ligature 'ij', in italic Times lowercase form and in Helvetica uppercase form (see Fig. 9).

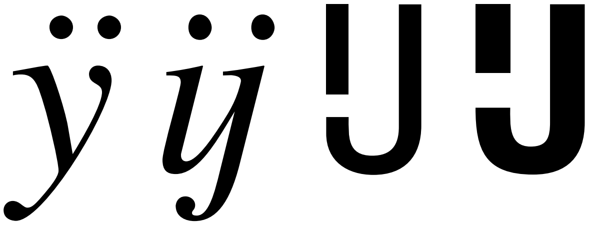


Figure 9: Italic letter 'y' with umlaut accent followed by the Dutch ligature 'ij' in Times italic, Helvetica medium uppercase and Helvetica bold uppercase form

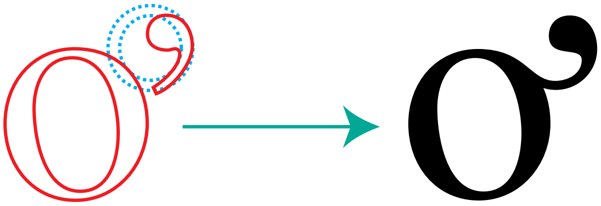


Figure 10: The Vietnamese SMALL LETTER O WITH HORN was designed using the glyphs of the ring accent, the apostrophe and letter 'o'

In the former case, we have connected the two letters, creating an intentional confusion with a 'y' letter with umlaut accent. In Fig. 9 the reader can compare these two constructions. On the other hand, the shape of the uppercase sans serif 'IJ' is derived from that of the letter 'U': this is common Dutch titling practice.

The characters Ꝁ and ꝁ are used in Breton. They are not (yet) included in Unicode, and have been brought to our attention by Jacques André.

Finally, in Fig. 10 the reader can see how the "horn" of Vietnamese letters has been designed, using graphical elements from the font: in this case, the ring accent and the apostrophe.

Ligatures Besides the "standard five" ligatures ff, fi, fl, ffi, ffl, we have included ligatures for the case where:

- the second or third letter is an 'i' with ogonek: fį, ffį, useful in Lithuanian;
- the second or third letter is a stroked 'l' : fł, ffł, useful in Polish;
- the second or third letter is a 'j': fj, ffj;
- instead of an 'i' one has an 'ij' ligature: fij, ffij, useful in Dutch;
- instead of 'f' one has a long 's': ſſ, ſi, ſl, ſſi, ſſl. We haven't (yet) included any 'f' + 'long s' combinations.

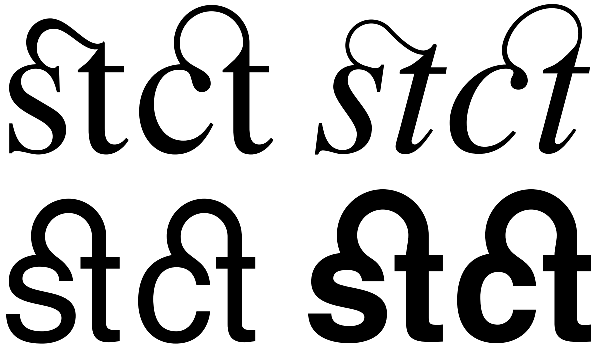


Figure 11: French ligatures ‘st’ and ‘ct’ in Times and Helvetica fonts

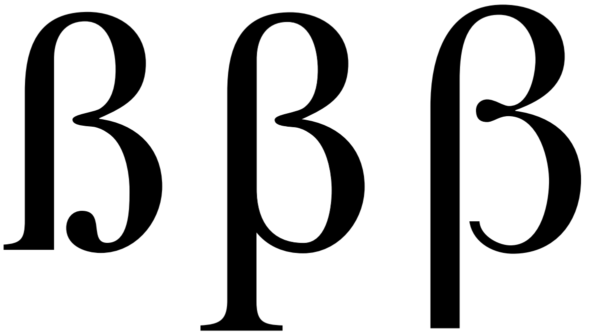


Figure 13: Three closely-related glyphs: German ‘sz’, IPA ‘beta’ (not a Unicode character), Greek lowercase ‘beta’

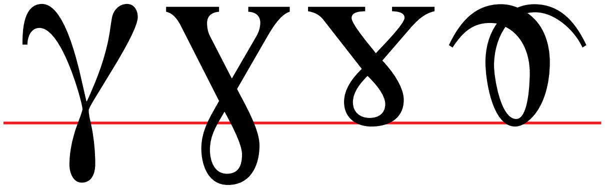


Figure 14: Different types of ‘gamma’: the first one from the Greek alphabet, the others from the IPA

Finally we couldn’t resist the temptation of making “French” ligatures ﬆ, ct, of course, both in Times and Helvetica (!) font families. These ligatures are well-known because of their use (in the Garamond typeface) in the Pléiade book collection. One can argue about their reason for being in Times (and especially in Helvetica) style, which has absolutely no historical background... Consider them an experiment, and trust us for not making them automatic in default text mode!

Diacritics Accents `0xf8`–`0xff` are used for Vietnamese only. Diacritics occupying positions `0xe8`–`0xf6` are shared with the “Common” Glyph Container.

The “IPA” Glyph Container

The “IPA” Glyph Container, shown in Table 3, contains a collection of glyphs needed for the International Phonetic Alphabet. They have been found in different sources: Unicode encoding, literature on phonetics (in particular we covered the complete table of characters of the French classic *Initiation à la phonétique*, by J.-M.-C. Thomas, L. Bouquiaux and F. Cloarec-Heiss, PUF, 1976). Only a small number of these characters are contained in the Unicode encoding. We would be **grateful** for any feedback from scholars on additional characters or corrections of the existing ones.

Since IPA is so... international, we have assumed that one can encounter phonetic insertions in text written in any language. Hence, we have included all glyphs needed, including lowercase Latin alphabet letters, and small capitals — the latter being included only whenever these are significantly different from their lowercase counterparts: including, for example, small caps ‘s’, ‘x’, ‘z’ would be useless since they are indistinguishable from lowercase ‘s’, ‘x’, ‘z’.

This point deserves some explanation: one should not confuse *small capitals used for text* and *IPA small capitals*. The former are a stylistic enhancement of text; they appear only in words entirely in small capitals style; their height is *not* equal to the x-height of the font, generally they are slightly higher. The latter are phonetic characters used together with authentic lowercase letters: they *must* have exactly the same height.

In Fig. 12 the reader can see some small caps we have designed for the ΩTimes IPA Glyph Container. It may not be obvious from the figure, but stroke widths of the small caps are exactly the same as the ones of lowercase letters (see lowercase letter ‘a’ to compare, as well as glyph ‘v’, which is used to represent both a lowercase and a small caps ‘v’). On the second line of the figure, one can see the same letters obtained as ‘fake’ small capitals.

We had a lot of fun designing these characters. In some cases, they just had to look a little different from their Greek counterparts: for example, in Fig. 13 the reader can see how the ‘phonetic beta’ has been inspired by the German ‘sz’, and not by the ‘real’ Greek beta; in Fig. 14 we present a collection

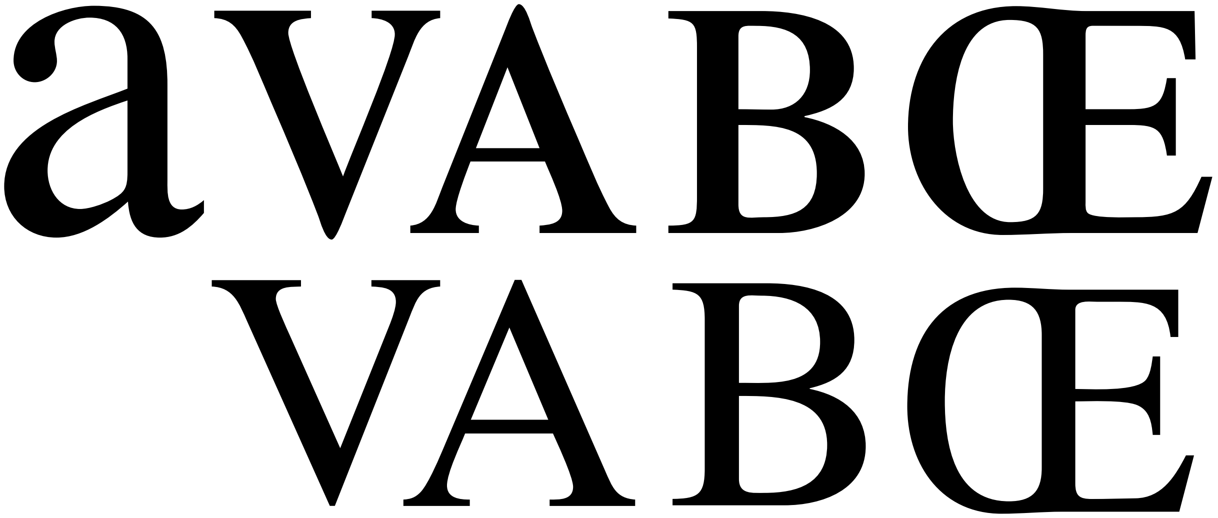


Figure 12: On the first line, lowercase letters ‘a’ and ‘v’ and specially designed IPA small capitals; on the second line, uppercase letters reduced 68%

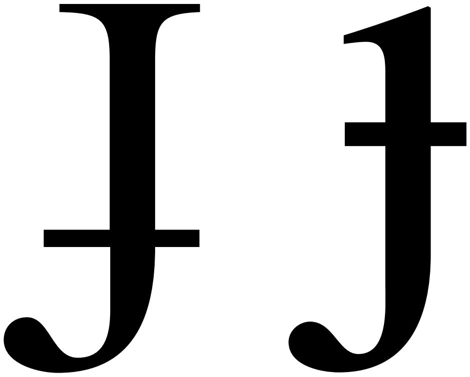


Figure 15: Two IPA symbols of different origin: inverted letter ‘f’ and dotless ‘j’ with stroke

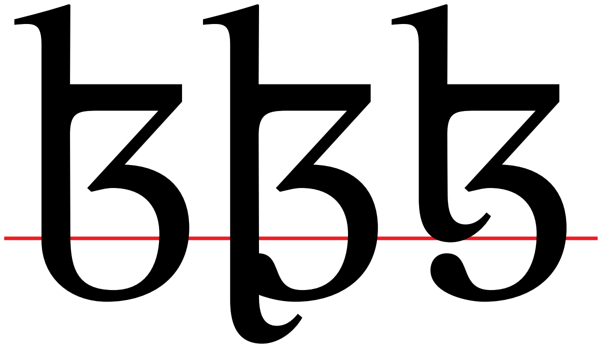


Figure 16: Which one is the best ‘l with retroflex hook+ezh’ ligature? (Three proposals)

of gamma-like glyphs, as well as the ‘real’ Greek letter gamma.

In some cases we were not very sure about the origin of some IPA characters and made several attempts: for example, in Fig. 15 one can see two symbols with a superficial resemblance: an inverted ‘f’ and a dotless stroked ‘j’. Which one is used in phonetics? The choice is left to the user.

Finally there was one case where we had to solve a *real* design problem: the one of the ‘l with retroflex hook’ and ‘ezh’ ligature (not a Unicode character). In most real-life examples we had the opportunity to see, the letter ‘ezh’ was sadly squeezed so that its tail remains higher than the retroflex hook of the ‘l’; we find it bad typographical practice to squeeze the ‘ezh’ and propose three solutions (only one of which of course will survive in the final Glyph Container). In Fig. 16 the reader can see: (a) the tail of ‘ezh’ merged with the retroflex hook of the ‘l’, (b) the retroflex hook of the ‘l’ continuing deep enough for it to be seen under the tail of ‘ezh’, (c) the retroflex hook rising higher so that it fits between the tail of ‘ezh’ and the baseline. In cases (a) and (c) the risk may be that the ‘l with retroflex hook’ be taken for an ‘ordinary l’ (compare ɭʒ, ɭʒ and ɭʒ); in case (b) we are going too deep under the baseline…

Anyhow, we expect your feedback to resolve this issue.

The “Greek” Glyph Container

The “Greek” Glyph Container, shown in Table 4, contains glyphs needed for the Greek language, ancient and modern. The problem with most

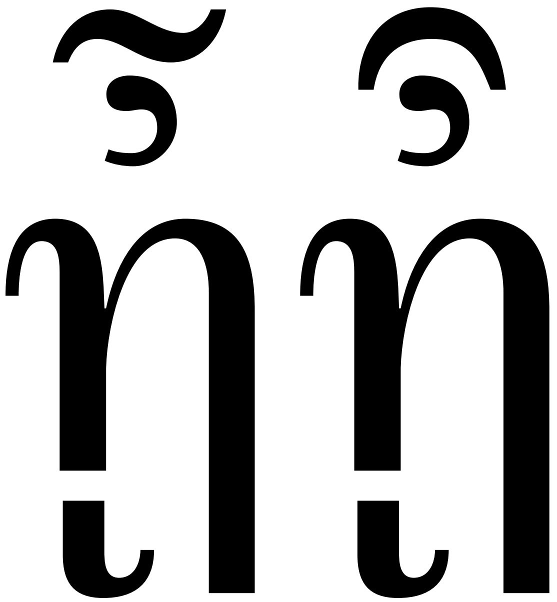


Figure 17: Two forms of Greek circumflex accent: the first used in modern Greek typesetting, the second one used in a number of scholarly typefaces

commercial Greek fonts is that they are either made for modern Greek use, or for ancient Greek use by non-Greek scholars. There is a third possibility: fonts made for ancient Greek use by Greek scholars.

In this Glyph Container we have tried to satisfy all three categories of users. Of course, this font can be used both for monotonic and polytonic text (there is a straight monotonic accent, while the acute one can be used as well). But we have gone even farther, by including two versions of the circumflex accent, shown in Fig. 17: the tilde-like one, used in Greece, and the cap-like one used in scholarly Western editions.[3]

Faithful to the first *TUGboat* paper by one of the authors (Haralambous and Thull, 1989), we have included the inverted iota with circumflex accent, found in certain 19th-century editions of modern Greek (see Fig. 18).

Version 1.0 of the Unicode standard contained uppercase versions of Greek numerals, just as roman numerals were provided in upper- and lowercase; in ISO 10646 these characters were removed, apparently by decision of the Greek delegation. We find this action absurd (there are many examples of uppercase numerals in literature) and have of course included the relevant glyphs in the Glyph Container.

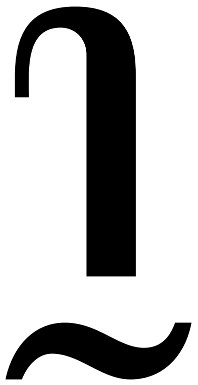


Figure 18: The inverted 'iota with circumflex' (not a Unicode character) used in 19th-century modern Greek printing

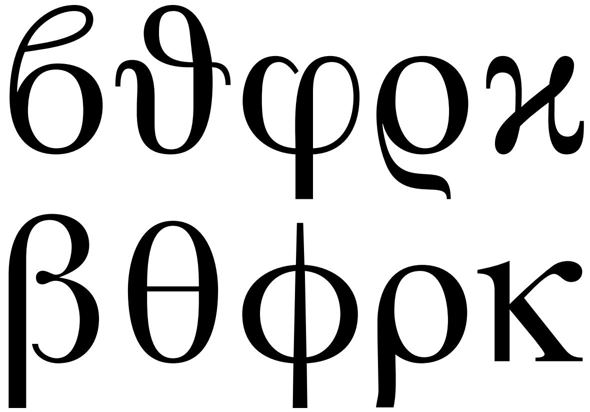


Figure 20: The different variant forms of Greek letters: beta, theta, phi, rho, kappa

In fact, we have included all known variants of Greek numerals: stigma, digamma, qoppa and sampi (see Fig. 19, next page), and the upper and lower numeral signs.

Greek letters are also very common in mathematics and physics. Some variant forms are used for different purposes in these fields (rho with curved or straight tail, open/closed phi, open/closed theta, curly or straight kappa). To these we add a variant form used in regular text: the initial and medial beta (β vs. ϐ). All variant forms of lowercase letters are shown in Fig. 20.

There is also a variant form of the letter sigma: the so-called "lunate" sigma. This character is used in some scholarly editions to avoid the distinction between ordinary medial and final sigmas. In Fig. 21 we compare it with Latin letter 'c': the lowercase of lunate sigma has no bulb and the upper one, no serif

[3] This brings us to a nice typographical joke: the Greek "circumflex" accent is either "tilde"-like or "cap"-like, but never actually… "circumflex"-like!

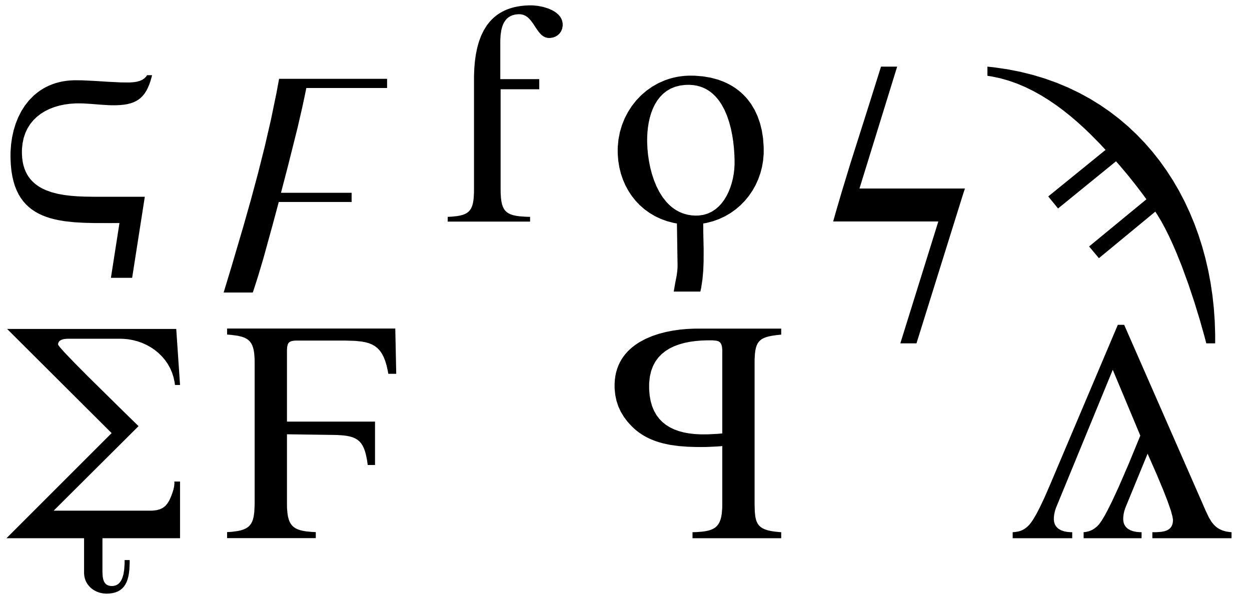


Figure 19: Collection of glyphs used for Greek numerals; on the first line: lowercase stigma, digamma (2 variants), qoppa (2 variants), sampi; on the second line: uppercase stigma, digamma, qoppa, sampi

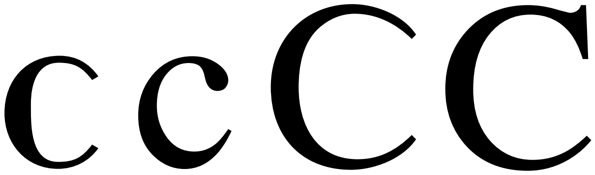


Figure 21: The glyphs of Greek lunate sigma and Latin letter 'c' (in upper- and lowercase)

(and hence the Greek and Latin letters are identical in the Helvetica family).

The reader may ask why on positions `0x80`–`0x8d` and `0xa0`–`0xad` of the Glyph Container table we have accented letters, while on positions `0x90`–`0x9d` there are only accents and no letters. The answer is very simple: accents on Greek letters sometimes change shape according to the width of the letter. Greek vowels can be roughly divided into three width classes: narrow ones (the iota), wide ones (the omega) and medium ones (all the remaining). On row `8` and `a` we have placed accents for narrow and wide letters, respectively; on row `9` we have placed the ordinary accents. And since rows `8` and `a` have been made for unique vowels, we have prefered to include complete accent+letter combinations (letters being aliases, of course) so that the virtual font doesn't need to make the construction.

The same reason justifies positions `0x5e` and `0x5f`: the dieresis (dialytika, in Greek) does not have the same width, depending on whether it is placed on an iota or an upsilon.

Finally, in the table there are also the specifically Greek guillemets (round ones), the upper dot, and the two forms of "subscript" iota: for lowercase letters (ypogegrammeni) and for uppercase ones (prosgegrammeni).

The "Cyrillic" Glyph Container

The "Cyrillic" Glyph Container, shown in Table 5, contains glyphs needed for all languages using the Cyrillic alphabet, whether European or Asian. Characters for pre-Lenin Russian (fita, izhitsa, yat) have also been included, as well as "modernized" versions of Slavonic characters. As in the case of the Latin 'st', 'ct' ligatures, one may argue the necessity of modernizing Slavonic script, especially when it comes to drawing the Helvetica version. It happens that these characters have been included in Unicode (like the Coptic ones), and this is already reason enough to draw the font; it is up to the user to actually adopt the "modern" font, or use beautiful traditional Slavonic typefaces.

What was fascinating when designing Slavonic letters was their relation to Greek ones. In Fig. 22 we compare Slavonic and Greek Xi and Psi: at a first glance, Slavonic Xi bears a resemblance to Greek lowercase xi, it is quite surprising that

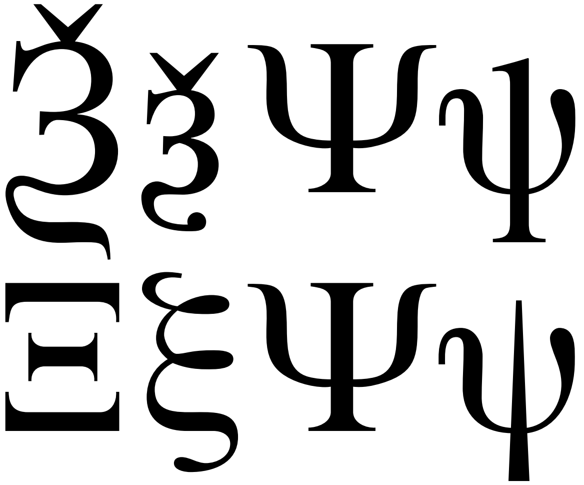


Figure 22: Comparing Slavonic letters Xi and Psi with the corresponding Greek ones

the Slavonic xi is reflected with respect to the Greek one. Uppercase Psi letters are identical, and lowercase Slavonic psi has a heavier stem (with serif, while Greek lowercase letters *never* have serifs). Uppercase Slavonic Omega Ѡ is a magnified lowercase omega (where we have placed a serif on the central stem). For the lower part of the Slavonic Yus Ѫ we have used an inverted Psi: this is a small designer's secret allowing better integration of the letter in the Times (resp. Helvetica) font family.

Another design initiative of ours was to use graphic elements from the Serbian letter Ђ ђ, for the Asian Cyrillic Ҕ ҕ, Ҧ ҧ, Ӄ ӄ. We expect user feedback to validate or deny this choice.

The "Arabic" Glyph Container

The "Arabic" Glyph Container, shown in Tables 6–8 contains the glyphs that are necessary to typeset in any language using the Arabic alphabet.[4] The design of these Ω Arabic fonts doesn't actually have much in common the Times and Helvetica families; in fact, we have used popular modern designs, which can be used both for technical and literary text, and which allow easy readability in small sizes. Fat and thin stroke width, ascender height and descender depth have all been calculated with respect to the corresponding parameters of the Latin/Greek/Cyrillic fonts.

In Fig. 23 the reader can see the metrical relationship between the Latin, Arabic and Hebrew fonts: we want these characters to fit with one other, so that multilingual texts using the three scripts will produce typographically acceptable results.

Concerning diacritics we have included all vowels, and combinations with shadda and hamza, as well as some special cases: madda, wesla, vertical fatha, vertical fatha + shadda, and other diacritics used in Arabic spellings of African languages, Kurdish, Baluchi, Kashmiri, Uigur and Kazakh.[5]

Esthetic Ligatures We have included a very small number of esthetic ligatures (fewer than 150): ba-like letters followed by a final noon-like, initial fa-like letters followed by a ya-like, an initial lam-meem ligature, and the llah ligature with and without vertical fatha + shadda. We are not convinced that heavy ligaturing of these fonts, in the manner of traditional Naskhi fonts, would be esthetically judicious. Nevertheless, we are open to any suggestion on possible ligatures that might be added.

The "Tifinagh" Glyph Container

The "Tifinagh" Glyph Container, shown in Tables 9 and 10, contains the glyphs needed for typesetting the Tamazight (alias Berber) language. A complete description of the Berber TeX system developed by Haralambous [2].

The glyphs shown on the tables warrant some explanation. Tifinagh script has always been written in a non-serif style. On table 9 we show a "Helvetica" version of the script, in the sense that everything has been done to bring these glyphs closer to the Latin/Cyrillic/Greek Helvetica types. This approach is quite safe, and — in all modesty — the result should not surprise any speaker of Berber.

On the other hand, table 10 shows a 100% experimental font! The main idea was to say: "What would Tifinagh letters look like today if they had followed the same evolution as Latin ones?" See Fig. 24 for a closer comparison of some Tifinagh letters in both Helvetica and Times styles. Admittedly, the result seems weird, and a lot of corrections have to be made before these drawings actually can be called Tifinagh glyphs... Nevertheless, the Berber language is being typeset more and more in its traditional script, often together with other scripts. As a result, the problem of homogenization with Western typographical traditions must be faced; the style of the Times typeface is one of the first challenges.

[4] We have also included undotted versions of letters ba, fa, qaf, for typesetting of old manuscripts.

[5] We have *not* included diacritics used for the Qur'ān, as we believe that these should be printed using very specific traditional typefaces; nevertheless, the diacritics provided are sufficient for excerpts from the Qur'ān.

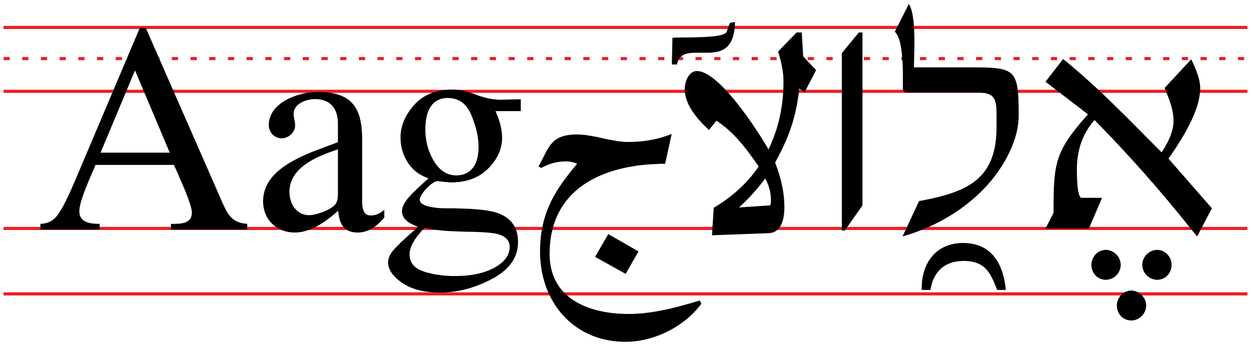


Figure 23: Capital height, x-height, baseline and descender depth of Latin, Arabic and Hebrew letters

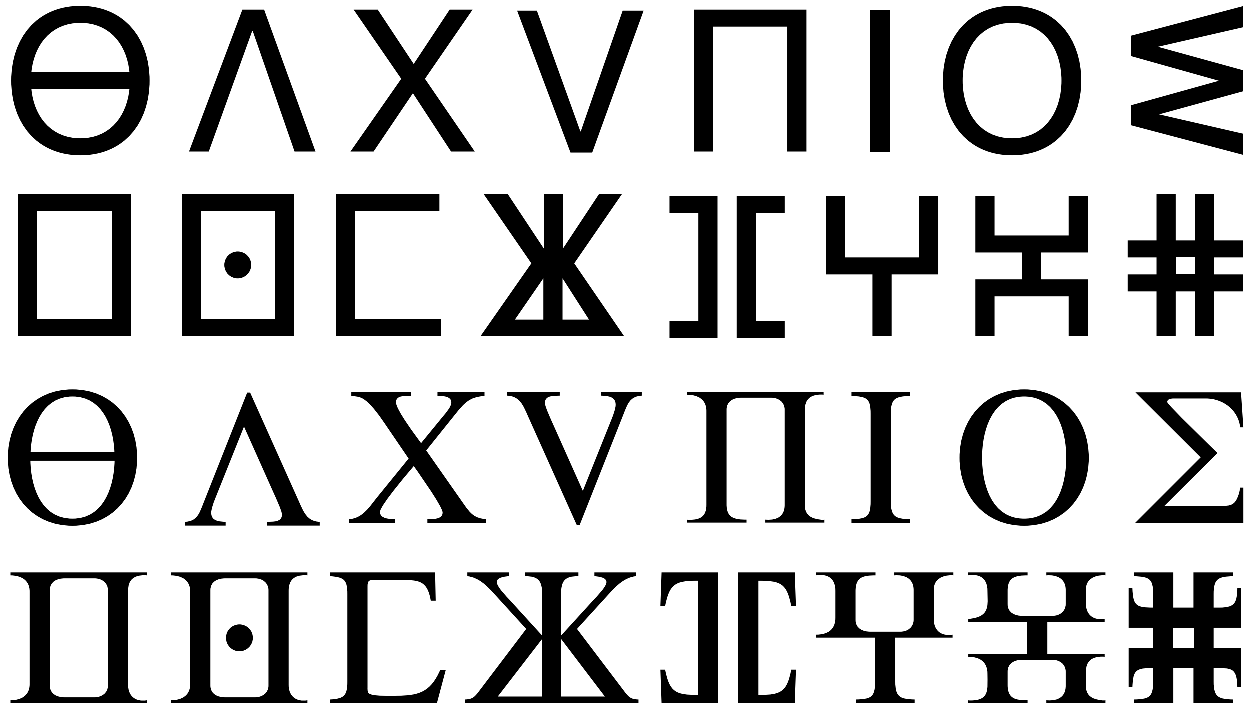


Figure 24: On line 1: Tifinagh letters in Helvetica style, with shapes identical to Greek and Latin ones; on line 2: idem. but shapes become stranger; on lines 3 and 4: the same letters, in Times style, using "evolutional logic"

The "Hebrew" Glyph Container

The "Hebrew" Glyph Container, shown in Table 11, contains the glyphs needed for Hebrew, Yiddish (both in the Russian or American YIVO spelling) and Ladino. The font design is based on a — very popular in Israel — modern Hebrew typeface. Once again we have adapted the stroke widths and character dimensions of Latin/Greek/Cyrillic glyphs. In Fig. 23, the reader can compare the size and weight of Hebrew letters with those of Latin and Arabic.[6]

We have *not* included Masoretic signs in the font, because we believe that the Bible should be typeset in traditional "square" fonts. Nevertheless, we have included letters with dagesh which appear only in a few isolated positions in the Bible, as well as the inverted nun. We have also included two

[6] It should be noted that the height of Hebrew letters is exactly equal to the half distance between that of upper- and lowercase Latin letters.

different forms of the lamed-aleph ligature (with and without left stroke), used in older texts.

References

[1] Haralambous, Yannis and Klaus Thull. "Typesetting modern Greek with 128-character codes." *TUGboat* 10,3 (1989), pages 354—359.

[2] Haralambous, Yannis. *Un système TEX berbère*. Actes de la table ronde internationale « Phonologie et notation usuelle dans le domaine berbère ». Paris: Institut National des Langues et Civilisations Orientales, 1993. [Forthcoming in the *Cahiers GUTenberg* thematic issue on Semitic scripts.]

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x		!	"	#	$	%	&	'
	(	)	*	+	,	-	.	/
"3x	0	1	2	3	4	5	6	7
	8	9	:	;	<	=	>	?
"4x	@	©	🄯	®	℗	¢	£	¤
	§	¶	¦	º	ª	¬	†	‡
"5x	‰	‱	′	‵	‸	⁁	⁂	▪
	⟦	⟧	℮	[	\	]	^	_
"6x	{	}	※	‿	⁀	␣	¡	¿
	‘	’	‛	‚	“	”	‟	„
"7x	«	»	‹	›	–	—	◦	•
	‣	\	№	{	|	}	~	
"8x	¥	₠	₡	₢	₣	₤	₥	₦
	₱	₩	₵	℔	℞	℟	℣	฿
"9x	Ⅎ	‽						
"Ax	̀	́	̂	̃	̄	̅	̆	̇
	̈	̉	̊	̋	̌	̍	̎	̏
"Bx	̐	̑	̒	̓	̔	̕	̖	̗
	̘	̙	̚	̛	̜	̝	̞	̟
"Cx	̠	̡	̢	̣	̤	̥	̦	̧
	̨	̩	̪	̫	̬	̭	̮	̯
"Dx	̰	̱	̲	̳	̴	̵	̶	̷
	̸	̹	̺	̻	̼	̽	̾	̿
"Ex	͡	͠						
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 1: Tentative `OmegaTimesCommon` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x		Æ	Đ	Ø	Þ	Ą	ß	Ę
	Ħ	Į	ı	Ĳ	Ł	Ŋ	Œ	Ŧ
"3x		æ	ð	ø	þ	ą	đ	ę
	ħ	į	ȷ	ĳ	ł	ŋ	œ	ŧ
"4x	Ų	A	B	C	D	E	F	G
	H	I	J	K	L	M	N	O
"5x	P	Q	R	S	T	U	V	W
	X	Y	Z	ĸ	Ơ	Ư	Ꝁ	
"6x	ų	a	b	c	d	e	f	g
	h	i	j	k	l	m	n	o
"7x	p	q	r	s	t	u	v	w
	x	y	z	ſ	ơ	ư	ꝁ	
"8x	Ɓ	Ƃ	Ƅ	Ɔ	Ƈ	Ɗ	Ƌ	Ƹ
	Ǝ	Ə	Ɛ	Ƒ	Ɠ	Ɣ	Ƕ	Ɨ
"9x	Ƙ	ƚ	Ɯ	Ɲ	ƞ	Ɵ	Ƣ	Ƥ
	Ƨ	Ʃ	ƪ	Ƭ	Ʈ	Ʊ	Ʋ	Ƴ
"Ax	ƀ	ƃ	ƅ	ɔ	ƈ	ɗ	ƌ	ƍ
	ɜ	ə	ɛ	ƒ	ɠ	ɣ	ɩ	ɨ
"Bx	ƙ	ƛ	ɯ	ɲ	Ʀ	ɵ	ƣ	ƥ
	ƨ	ʃ	ƫ	ƭ	ʈ	ʊ	ʋ	ƴ
"Cx	Ƶ	Ʒ	Ƹ	ƺ	Ƽ	ƾ	ǂ	Ǥ
	Ȣ				ff	fi	fl	ffi
"Dx	ffl	fį	ffį	fł	ffł	fj	ffj	fij
	ffij	ſſ	ſi	ſl	ſſi	ſſl	ſt	ct
"Ex	ƶ	ʒ	ƹ	ƻ	ƽ	ƿ	¸	ǥ
	`	´	ˆ	˜	¯	˘	˙	¨
"Fx	ˀ	˚	˝	ˇ	˵	⁀	.	’
	ˆ́	ˆ̀	ˆ̉	ˆ̃	˘́	˘̀	˘̉	˘̃
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 2: Tentative `OmegaTimesLatin` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x	ɑ	ɒ	ɐ	ɑ˞	ɓ	ƃ	ƀ	b̵
	b̸	β	ɔ	ɔ˞	ƈ	ç	c̸	ɕ
"3x	ɖ	ᶁ	đ	ɗ	ʤ	δ	ʣ	ʥ
	đ	d̵	d̸	ð	ə	ɘ	ɚ	ę
"4x	ɛ	A	B	ɜ	D	E	F	G
	H	I	J	K	L	M	N	Œ
"5x	P	Q	R	ɝ	T	U	ɛ˞	ʃ
	ʄ	Y	ɸ	φ	ɸ	φ̵	ɸ̵	Ɉ
"6x	ſ	a	b	c	d	e	f	g
	h	i	j	k	l	m	n	o
"7x	p	q	r	s	t	u	v	w
	x	y	z	ɡ	ɣ	γ	ɤ	
"8x	ɷ	ɷɷ	ʛ	ǥ	ɠ	ħ	ɦ	ɥ
	ɧ	ʮ	ꞕ	ɦ	ƕ	ɨ	ɩ	ᵻ
"9x	ɟ	ȷ	ʝ	j	ʞ	ᴋ	ᶄ	ɫ
	ɭ	ʎ	ɮ	l	ɬ	ɮ	ʎ̸	ʎʒ
"Ax	⅂	ł	ᶅ	ƚ	ƛ	λ	ɱ	ɯ
	ɰ	hm	ɳ	ɲ	ŋ	n̥	n̥̥	ɳ̊
"Bx	ᶇ	ƞ	ɲ	ơ	ǫ	ǫ	ᶈ	ƥ
	ƥ	p̬	ᵽ	ʠ	ɋ	ɽ	ɹ	ʁ
"Cx	ɾ̢	ɺ	ɻ	ɼ	ʅ	ɺ	ɾ	ʂ
	ʃ	ʅ	ɿ	ʆ	ꞩ	σ	ʈ	ƫ
"Dx	ŧ	ɵ	ʦ	ʧ	ɟ	ʨ	θ	ʉ
	Ω	ɥ̸	ᵿ	ʋ	ʌ	ⱴ	ʍ	ɷ
"Ex	ω	χ	ᶍ	ʐ	ʒ	ƶ	ʓ	ʑ
	ʓ	ƨ	ʔ	ʕ	ʖ	ȣ	ʗ	⊙
"Fx	ɞ	ϖ	Ω	ʡ	ʢ	ƻ	ꝏ	ɮ
	ɮ							
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 3: Tentative OmegaTimesIPA Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"3x	ϒ	ϐ	ϑ	φ	ϱ	ϰ	ϲ	ϖ
	ι̰	ϙ		·	«	»	ͺ	ɪ
"4x		Α	Β	C	Δ	Ε	Φ	Γ
	Η	Ι	Θ	Κ	Λ	Μ	Ν	Ο
"5x	Π	Χ	Ρ	Σ	Τ	ϒ	F	Ω
	Ξ	Ψ	Ζ	Σ̦	Ϟ	Ϡ	Ϊ	Ϋ
"6x		α	β	ς	δ	ε	ϕ	γ
	η	ι	θ	κ	λ	μ	ν	ο
"7x	π	χ	ρ	σ	τ	υ	ϝ	ω
	ξ	ψ	ζ	ς	ϟ	ϡ	f	
"8x	ἱ	ἰ	ὶ	ί	ῖ	ι̑	ἳ	ἵ
	ἷ	ἱ̑	ἲ	ἴ	ἶ	ἰ̑	ʹ	ˏ
"9x	῾	᾿	`	´	~	̑	῝	῞
	῟	῾̑	῍	῎	῏	᾿̑	'	
"Ax	ὡ	ὠ	ὼ	ώ	ῶ	ω̑	ὣ	ὥ
	ὧ	ὡ̑	ὢ	ὤ	ὦ	ὠ̑		
"Bx	ϊ	ῒ	ΐ	ῗ	ϊ̑	ϋ	ῢ	ΰ
	ῧ	ϋ̑	ῥ	ῤ	ϱ̔	ϱ̓		
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 4: Tentative OmegaTimesGreek Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x		Ђ	Є	Ѕ	І	Ј	Љ	Њ
	Ћ	Щ	Џ	Ѡ	Ѥ	Ѧ	Ѩ	Ѫ
"3x	ı	ђ	є	ѕ	і	ј	љ	њ
	ћ	щ	џ	ѡ	ѥ	ѧ	ѩ	ѫ
"4x	Я	А	Б	Ц	Д	Е	Ф	Г
	Ю	И	Ж	К	Л	М	Н	О
"5x	П	Ч	Р	С	Т	У	В	Ш
	Х	Ы	З	Ъ	Ѣ	Ь	Э	
"6x	я	а	б	ц	д	е	ф	г
	ю	и	ж	к	л	м	н	о
"7x	ԥ	ч	р	с	т	у	в	ш
	х	ы	з	ъ	ѣ	ь	э	
"8x	Ғ	Ѭ	Ѯ	Ѱ	Ѳ	Ѵ	Ҁ	Ѽ
	Ѿ	Ҫ	Ґ	Ғ	Ҕ	Җ	Қ	Ҟ
"9x	҂	ѭ	ѯ	ѱ	ѳ	ѵ	ҁ	ѽ
	ѿ	ҫ	ґ	ғ	ҕ	җ	қ	ҟ
"Ax	Ҝ	Ҡ	Ӊ	Ҥ	Ԥ	Ҩ	Ҭ	Ү
	Ұ	Ҳ	Ҵ	Ҷ	Ҹ	Һ	Ҽ	Ӄ
"Bx	ҝ	ҡ	ӊ	ҥ	ԥ	ҩ	ҭ	ү
	ұ	ҳ	ҵ	ҷ	ҹ	һ	ҽ	ӄ
"Cx	Ӈ	Ӌ	Ӕ	Ә	Ӡ	Ө	Ҙ	Ҿ
	Ç							
"Dx	ӈ	ӌ	ӕ	ә	ӡ	ө	ҙ	ҿ
	ç							
"Ex	ˏ	¨	˵	˘	˝	ˉ	҃	~
	҅	҆	ᴗ					
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 5: Tentative `OmegaTimesCyrillic` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x	*	،	؛	؟	-	٪	؉	؊
"3x	٠	١	٢	٣	٤	٥	٦	٧
	٨	٩					ﷲّ	ﷲ
"4x	َ		ُ	ْ	ً	ٍ	ٌ	ٰ
	َّ	ِّ	ُّ	ّ	ًّ	ٍّ	ٌّ	ّٰ
"5x	ٔ َ	ٔ	ٔ ُ	ء	ٔ ً	ٔ ٍ	ٔ ٌ	ـٔ
	¯	^	ˇ	ٕ ِ	ٕ	ٕ ٍ		
"6x	ء	آ	ﺂ	أ	ﺄ	ؤ	ﺆ	إ
	ﺈ	ﺉ	ﺋ	ﺌ	ﺊ	ﺍ	ﺎ	ﺏ
"7x	ﺑ	ﺒ	ﺐ	ﺓ	ﺔ	ﺕ	ﺗ	ﺘ
	ﺖ	ﺙ	ﺛ	ﺜ	ﺚ	ﺝ	ﺟ	
"8x	ﺠ	ﺞ	ﺡ	ﺣ	ﺤ	ﺢ	ﺥ	ﺧ
	ﺨ	ﺦ	ﺩ	ﺪ	ﺫ	ﺬ	ﺭ	ﺮ
"9x	ﺯ	ﺰ	ﺱ	ﺳ	ﺴ	ﺲ	ﺵ	ﺷ
	ﺸ	ﺶ	ﺹ	ﺻ	ﺼ	ﺺ	ﺽ	ﺿ
"Ax	ﻀ	ﺾ	ﻁ	ﻃ	ﻄ	ﻂ	ﻅ	ﻇ
	ﻈ	ﻆ	ﻉ	ﻋ	ﻌ	ﻊ	ﻍ	ﻏ
"Bx	ﻐ	ﻎ	ﻑ	ﻓ	ﻔ	ﻒ	ﻕ	ﻗ
	ﻘ	ﻖ	ﻙ	ﻛ	ﻜ	ﻚ	ﻝ	ﻟ
"Cx	ﻠ	ﻞ	ﻵ	ﻶ	ﻷ	ﻸ	ﻹ	ﻺ
	ﻻ	ﻼ	ﻡ	ﻣ	ﻤ	ﻢ	ﻥ	ﻧ
"Dx	ﻨ	ﻦ	ﻩ	ﻫ	ﻬ	ﻬ	ﻪ	ﻭ
	ﻮ	ﻯ	ﻳ	ﻴ	ﻰ	ﻱ	ﻲ	ﻠ
"Ex	ٮ	ٮـ	ـٮـ	ـٮ	ں	ـں	ڡ	ڡـ
	ـڡـ	ـڡ	ٯ	ـٯ	ڤ	ڤـ	ـڤـ	ـڤ
"Fx	ٹ	ٹـ	ـٹـ	ـٹ	ٽ	ٽـ	ـٽـ	ـٽ
	پ	پـ	ـپـ	ـپ	ټ	ټـ	ـټـ	ـټ
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 6: Tentative `OmegaTimesArabicOne` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x	ٺ	ٺـ	ـٺـ	ـٺ	ٻ	ٻـ	ـٻـ	ـٻ
	ٿ	ٿـ	ـٿـ	ـٿ	ڀ	ڀـ	ـڀـ	ـڀ
"3x	ځ	ځـ	ـځـ	ـځ	ڂ	ڂـ	ـڂـ	ـڂ
	ڃ	ڃـ	ـڃـ	ـڃ	ڄ	ڄـ	ـڄـ	ـڄ
"4x	څ	څـ	ـڅـ	ـڅ	چ	چـ	ـچـ	ـچ
	ڇ	ڇـ	ـڇـ	ـڇ	ڈ	ـڈ	ډ	ـډ
"5x	ڊ	ـڊ	ڈ	ـڈ	ڌ	ـڌ	ڍ	ـڍ
	ڎ	ـڎ	ڏ	ـڏ	ڐ	ـڐ	ڑ	ـڑ
"6x	ړ	ـړ	ڔ	ـڔ	ڕ	ـڕ	ڒ	ـڒ
	ڗ	ـڗ	ژ	ـژ	ڙ	ـڙ	ښ	ښـ
"7x	ـښـ	ـښ	ڛ	ڛـ	ـڛـ	ـڛ	ڜ	ڜـ
	ـڜـ	ـڜ	ڝ	ڝـ	ـڝـ	ـڝ	⌒	
"8x	ڞ	ڞـ	ـڞـ	ـڞ	ڟ	ڟـ	ـڟـ	ـڟ
	ط	طـ	ـطـ	ـط	ڠ	ڠـ	ـڠـ	ـڠ
"9x	ڢ	ڢـ	ـڢـ	ـڢ	ڣ	ڣـ	ـڣـ	ـڣ
	ڥ	ڥـ	ـڥـ	ـڥ	ڦ	ڦـ	ـڦـ	ـڦ
"Ax	ڧ	ـڧ	ڨ	ـڨ	کـ	ـکـ	ک	ـک
	ڪ	ـڪ	ګ	ګـ	ـګـ	ـګ	ك	كـ
"Bx	ـكـ	ـك	ڭ	ڭـ	ـڭـ	ـڭ	ڮ	ڮـ
	ـڮـ	ـڮ	گ	گـ	ـگـ	ـگ	ڰ	ڰـ
"Cx	ـڰـ	ـڰ	ڴ	ڴـ	ـڴـ	ـڴ	ڲ	ڲـ
	ـڲـ	ـڲ	ڳ	ڳـ	ـڳـ	ـڳ	ڱ	ڱـ
"Dx	ـڱـ	ـڱ	ڵ	ڵـ	ـڵـ	ـڵ	ڵا	ـڵا
	ڶ	ڶـ	ـڶـ	ـڶ	ڶا	ـڶا	ڷ	ڷـ
"Ex	ـڷـ	ـڷ	ڷا	ـڷا	ڻ	ـڻ	ڼ	ڼـ
	ـڼـ	ـڼ	ڽ	ـڽ	ـہـ	ہـ	ۂ	ـۂ
"Fx	ـۀ	ـة	ۄ	ـۄ	ۅ	ـۅ	ی	ـی
	ې	ـې	ۑ	ـۑ	ے	ـے	ۓ	ـۓ
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 7: Tentative OmegaTimesArabicTwo Glyph Container Table (June 10, 1996).

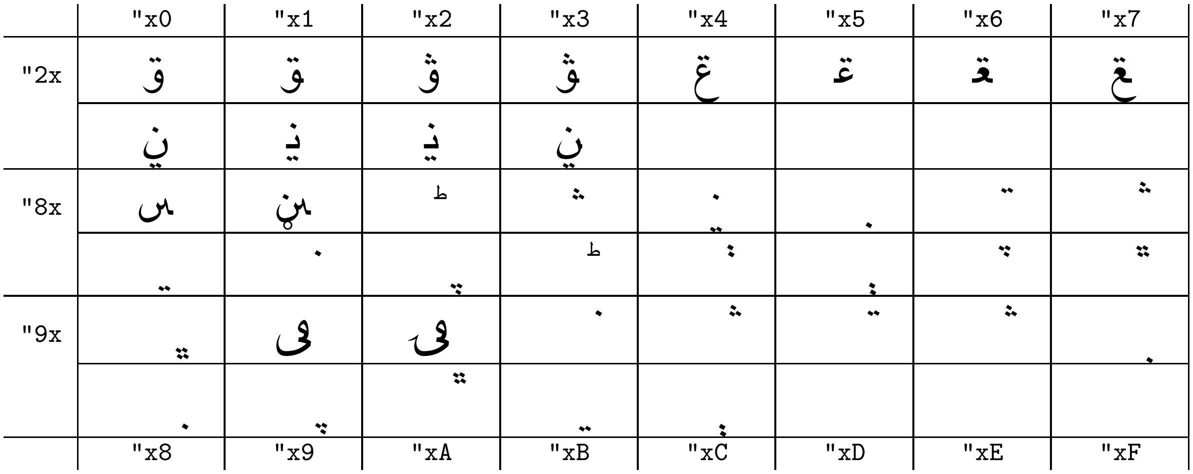

"x0
"x1
"x2
"x3
"x4
"x5
"x6
"x7
"2x
"8x
"9x
"x8
"x9
"xA
"xB
"xC
"xD
"xE
"xF

Table 8: Tentative `OmegaTimesArabicThree` Glyph Container Table (June 10, 1996).

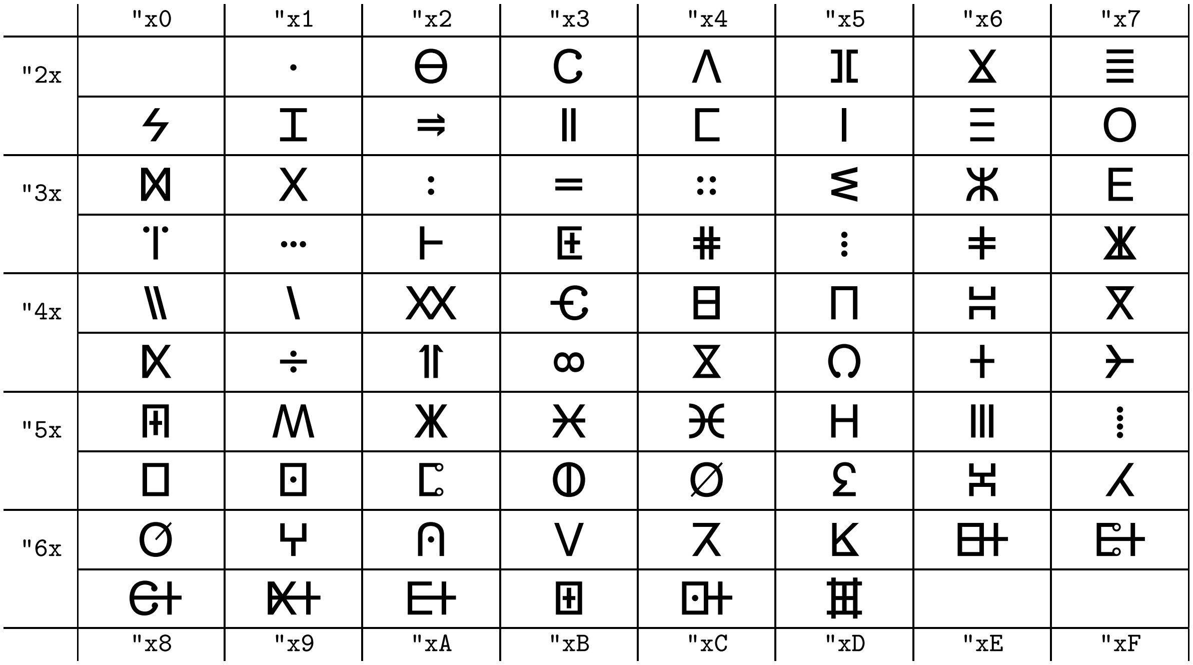

"x0
"x1
"x2
"x3
"x4
"x5
"x6
"x7
"2x
"3x
"4x
"5x
"6x
"x8
"x9
"xA
"xB
"xC
"xD
"xE
"xF

Table 9: Tentative `OmegaHelveticaTifinagh` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x		·	Θ	Ͼ	Λ	ƆC	X	≣
	ϟ	Ɪ	⇉	][	⊏	I	Ξ	O
"3x	Ⲙ	X	:	=	∷	Σ	Ж	E
	Ï	⋯	Ⱶ	Ɇ	⋕	⋮	ǂ	Ж
"4x	⑊	⧵	XX	Ꞓ	Ħ	Π	Ⲏ	X
	Ꝁ	÷	⥣	∞	X	Ω	ⵜ	ⵤ
"5x	Ⱨ	ΛΛ	Ж	Ӿ	ЭЄ	H	Ⅲ	⁝
	∏	⊡	Ⳑ	Φ	Ø	Ƹ	ⵅ	ʎ
"6x	Ø	Ψ	⩀	V	Ӽ	K	Ħⵜ	Ⳑⵜ
	Ͼⵜ	Ꝁⵜ	Eⵜ	Ⅲ	⊡ⵜ	Ⅲ		
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 10: Tentative `OmegaTimesTifinagh` Glyph Container Table (June 10, 1996).

	"x0	"x1	"x2	"x3	"x4	"x5	"x6	"x7
"2x		׳	״		ל̯	ע̯	בֿ	פֿ
	װ	ױ	ײ	ײַ	?	־	׃	
"3x	ַ	ִ	ֵ	ֶ	ָ	ֹ	ֻ	ְ
	ֲ	ֱ	ֳ		ﬞ	אּ	ﭏ	ﬡ
"4x	א	ב	ג	ד	ה	ו	ז	ח
	ט	י	ך	כ	ל	ם	מ	ן
"5x	נ	ס	ע	ף	פ	ץ	צ	ק
	ר	ש	ת	שׁ	שׂ			
"6x	אּ	בּ	גּ	דּ	הּ	וּ	זּ	חּ
	טּ	יּ	ךּ	כּ	לּ	םּ	מּ	ןּ
"7x	נּ	סּ	עּ		פּ		צּ	קּ
	רּ	שּ	תּ	שּׁ	שּׂ	וֹ	אֹ	
	"x8	"x9	"xA	"xB	"xC	"xD	"xE	"xF

Table 11: Tentative `OmegaTimesHebrew` Glyph Container Table (June 10, 1996).

Extending TEX for Unicode

Richard J. Kinch
6994 Pebble Beach Ct
Lake Worth FL 33467 USA
Telephone (561) 966-8400
FAX (561) 966-0962
kinch@holonet.net
URL: http://styx.ios.com/~kinch

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, 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	AMS Euler Big Operators
eufb	AMS Euler Fraktur Bold†
eufm	AMS Euler Fraktur
eur	AMS Euler
eus	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	AMS symbol set A
msbm	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 ...)

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		

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:
  - 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 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

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 $\cup$ mre $\cup$ 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*)

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.

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

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			•	•	•	•	•	•

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

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

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

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

  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. 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

[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	Lʼ
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 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 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

[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

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 Ω-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 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.

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

- 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! Ω-Babel." *TUGboat* 16,4 (1995), pages 418–427.

[5] Haralambous, Yannis. "Ω, 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. "Ω + 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 Ω 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 Ω 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.

Russian Encoding Plurality Problem and a New Cyrillic Font Set

L.N. Znamenskaya and S.V. Znamenskii
Krasnoyarsk State University, Svobodnyi prospekt 79, 660041 Krasnoyarsk, Russia
znamensk@ipsun.ras.ru

Abstract

To run TEX with cyrillic in network is a problem. Various widespread Cyrillic coding tables under DOS, UNIX and other OS are incompartible. The ASCII Russian text imported from a different system usually become completely unreadable. The new set of fonts, TDS and some other tools give a solution of the problem for the east-European Cyrillic typsertting users.

TEX has become the one of the best known means of communication between scientific people. To solve the problem of plural incompatible Russian TEX systems, the Russian Foundation for Basic Research (RFBR) proposed the idea of creation of a standard non-commercial Russian TEX distribution. Therefore, half a year ago the new "Russian TEX" project was begim under RFBR support. An important feature of the project is to determine the best system which is able to work in a LAN, with various client platforms and operating systems.

The new TDS (TEX Directory Structure) standard gives us the perfect base for a such system. The problem we find here is specific for the Cyrillic-based languages. It is the Russian encoding plurality problem. For example there exists several widely-used Russian coding tables under UNIX. Even Microsoft® uses completely different coding tables for Russian text under DOS and Windows® on the same PC. At the same time, in different directories on CTAN, we can find METAFONT sources for Cyrillic fonts with the same name cmrz10 but with different Russian letter"A" character codes.

Fonts

The first thing we have had to do was to select an available Cyrillic extended standard TEX font set and fix new names in order to reflect a coding table in the name of font. As soon as we found the CyrTUG LH fonts not to be available for non-commercial RFBR distribution for free, we asked N. Glonty and A. Samarin for a permission to use their fonts, as they are the first and the most widely used TEX fonts in Russia. After a period of a month and a half, we received the very kind and grateful permission to use or modify the fonts or their sources for RFBR distribution and we appreciate very much such a generous solution. Unfortunately, we could not wait so long and and at this point in time, the development of the new Russian extension of a CM TEX font family was at the kerning stage. It so happened that we obtained the extra Russian extension of CM font family.

We tried to realise the following aims in this new font set:

- to keep the original CM font sources unchangeable to input by extension sources in order to provide appropriate Latin text when typesetting using the new fonts;
- to make text and letters more habitual for the Russian eye, keeping the traditional CM fonts peculiarity;
- to make letter darkness in text more uniform;
- to make all CM source based fonts, including concrete available for Russian typesetting;
- to avoid possible low-resolution font-creation errors causing problems while using automatic font generation; and
- to lay the foundation for future support of all Cyrillic-based alphabets of the Russian people.

We used CM macros, fragments of CM codes and a bit of cmcyr code. The acroLH font family has been used just for comparison in the first stage.

When the new fonts were almost ready it was decided to compare their typesetting quality with the one of the best sources of widely distributed fonts — the Samarin and Glonty Cyrillic fonts. A large mathematical paper has been printed at 10 and 12 points on a 600dpi HP LaserJet4 printer, the same text in two copies printed with different font sets. There was a blank page in each copy for experts to write their opinion. The RFBR experts (physicists and mathematicians) compared the two, and determined that the both Russian font families are of the same good quality.

What should we do with the new fonts names? The first idea was to use the fontname scheme. In this way, we made the name of extended 8-bit font much too different from the name of corresponding standard 7-bit CM font. As a result users would have a problems while adopting new styles and using the TEX primitive font selection commands. To reduce such problems we decided to create a font name from RF (Russian Font + Russian Foundation); to use the third char (digit) in the name to point to the coding table, and to end by using the same char sequence as that used by the corresponding CM font. One can see the examples on tables.

The empty boxes in font tables will be filled by other Cyrillic letters in next version of fonts. It is impossible to support all Cyrillic-based languages by the same 8-bit coding table — the number of different letters is more than 256. The project is working on a coding table which would allow typesetting on more than sixty Cyrillic-based languages with the use of accents or virtual fonts or `\charsubdef`. The list of languages to be supported in such a way contains all of the Cyrillic-based languages of Russia.

Russian encoding plurality problem

We need to support the typical situation of an entire TEX file system residing on a server, with clients working under different operation systems using various Russian encodings. The main problem is to select the appropriate procedure for inputting TEX files with *any* encoding.

Our way to solve this problem is to create an executable which would recognize the Cyrillic coding of a file in the correct way, and then recode it automatically to conform to the local coding.

Why not? Anybody who reads Russian can easily convert the text in the right coding from the same text in the wrong coding. But as soon as we try to look more carefully at the problem, we see the multiple problems.

The coding tables one-to-one correspondence as a part of problem If a binary file is occasionally to be recoded as Cyrillic, it is useful to have the capability of recovering an accidently-converted file. The networking forse problem to be more difficult: multiple convertions must preserve the original information. We cannot see a way to solve this problem without the additional difficulty of creating a proper conversion algorithm. It is natural to preserve the ASCII first 128 positions of code table. In the last 128 positions, we have to put one-to-one correspondence between each set of coding tables in a consistent way.

Unfortunately, this is not possible. The set of symbols in this part of the coding tables differs very much from one table to other. Therefore we have to permit the Rchar to change meaning during conversions. We try to considerably decrease the set of possible meaning changes. The desirable solution is to split the set of all possible char meanings of the 128 equivalence classes such that any conversion can change the symbol meaning only inside its equivalence class. This is also impossible. Some of the meanings will necessarily be found in different classes and the best thing we can do is to use the less valuable meanings for such a mess. You can see the summary of a various available information on Cyrillic coding tables [1]–[8] and our proposals on the one-to-one table correspondence in a huge table bellow. In this table the numbers 0, 1, 2, 3, 4, 5 respectively denote ISO8859-5, CP1251, PC866, -8, MacOS, and PC855.

The problem of other Cyrillic languages There are more then 60 Cyrillic-based languages and some of them still have not settled coding tables. Most of the files contains a lot of the non-text commands. There is a lot of software which puts a non-ASCII chars into file and the program has to distinguish, as far as is possible, the right Cyrillic words from the combinations of such symbols.

We therefore cannot use only the char set information of the file to discover the coding table of document. Another problem we see is that some coding tables use the same char set. As we need to get a right solution for a short file, it is also inadequate just to count the number of each letters appearing in text. A more precise instrument would be to count the number of each combinations of two letters appearing in the document.

This effective approach require more them 128 kilobytes of memory for an intermediate data storage. The natural algorithm to perform a proper statistical analysis of this data includes multiple computing of logarithms and is not fast enough – especially on a PC. How to find a way to get the acceptable result in a simple and fast way?

The next idea was to select two sets of possible strings of length 2: the set, A, of frequently-appearing Cyrillic text bicharacter strings and a set, U, of commonly unused Cyrillic text bicharacter strings. The executable counts the numbers N_A and N_U of strings from A and U, respectively, appearing in the file. The number $C = \frac{N_A - N_U}{N_A + N_U}$ will show if this file looks as Cyrillic text or not. Such a number can

where	THE MEANING
23	BOX DRAWINGS DOWN SINGLE AND RIGHT DOUBLE
0145	CYRILLIC CAPITAL LETTER DJE
23	RIGHT HALF BLOCK
0145	CYRILLIC CAPITAL LETTER GJE
23	BOX DRAWINGS DOWN SINGLE AND LEFT DOUBLE
0145	CYRILLIC CAPITAL LETTER DZE
23	LEFT HALF BLOCK
0145	CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I
3	TOP HALF INTEGRAL
01245	CYRILLIC CAPITAL LETTER YI
23	BOX DRAWINGS UP SINGLE AND RIGHT DOUBLE
0145	CYRILLIC CAPITAL LETTER JE
23	BOX DRAWINGS UP DOUBLE AND RIGHT SINGLE
0145	CYRILLIC CAPITAL LETTER LJE
23	BOX DRAWINGS UP SINGLE AND LEFT DOUBLE
0145	CYRILLIC CAPITAL LETTER NJE
23	BOX DRAWINGS UP DOUBLE AND LEFT SINGLE
0145	CYRILLIC CAPITAL LETTER TSHE
23	BULLET OPERATOR
0145	CYRILLIC CAPITAL LETTER KJE
23	BOX DRAWINGS VERTICAL SINGLE AND RIGHT DOUBLE
0145	CYRILLIC CAPITAL LETTER DZHE
23	BOX DRAWINGS VERTICAL DOUBLE AND RIGHT SINGLE
0145	CYRILLIC SMALL LETTER DJE
23	BOX DRAWINGS VERTICAL SINGLE AND LEFT DOUBLE
0145	CYRILLIC SMALL LETTER GJE
23	BOX DRAWINGS VERTICAL DOUBLE AND LEFT SINGLE
0145	CYRILLIC SMALL LETTER DZE
23	BOX DRAWINGS DOWN SINGLE AND HORIZONTAL DOUBLE
0145	CYRILLIC SMALL LETTER BYELORUSSIAN-UKRAINIAN I
3	BOTTOM HALF INTEGRAL
01245	CYRILLIC SMALL LETTER YI
23	BOX DRAWINGS DOWN DOUBLE AND HORIZONTAL SINGLE
0145	CYRILLIC SMALL LETTER JE
23	BOX DRAWINGS UP SINGLE AND HORIZONTAL DOUBLE
0145	CYRILLIC SMALL LETTER LJE
23	BOX DRAWINGS UP DOUBLE AND HORIZONTAL SINGLE
0145	CYRILLIC SMALL LETTER NJE
23	BOX DRAWINGS VERTICAL SINGLE AND HORIZONTAL DOUBLE
0145	CYRILLIC SMALL LETTER TSHE
23	BOX DRAWINGS VERTICAL DOUBLE AND HORIZONTAL SINGLE
0145	CYRILLIC SMALL LETTER KJE
23	FULL BLOCK *)
0145	CYRILLIC SMALL LETTER DZHE
235	BOX DRAWINGS LIGHT VERTICAL AND RIGHT
14	CYRILLIC CAPITAL LETTER GHE WITH UPTURN
235	BOX DRAWINGS LIGHT VERTICAL AND LEFT
14	CYRILLIC SMALL LETTER GHE WITH UPTURN

Table 1: the non-russian letters
*) for this coding table

WHERE	THE MEANING
235	BOX DRAWINGS LIGHT UP AND RIGHT
14	LEFT SINGLE QUOTATION MARK
235	BOX DRAWINGS LIGHT UP AND LEFT
14	RIGHT SINGLE QUOTATION MARK
235	BOX DRAWINGS DOUBLE UP AND LEFT
14	LEFT DOUBLE QUOTATION MARK
235	BOX DRAWINGS DOUBLE UP AND RIGHT
14	RIGHT DOUBLE QUOTATION MARK
235	BOX DRAWINGS DOUBLE DOWN AND LEFT
14	DOUBLE LOW-9 QUOTATION MARK
4	POUND SIGN
235	BOX DRAWINGS LIGHT DOWN AND RIGHT
1	SINGLE LOW-9 QUOTATION MARK

Table 2: the symbols look more-or-less like left/right coma quotation

WHERE	THE MEANING
3	GREATER-THAN OR EQUAL TO *)
01245	CYRILLIC CAPITAL LETTER UKRAINIAN IE
3	DIVISION SIGN *)
01245	CYRILLIC CAPITAL LETTER SHORT U
3	LESS-THAN OR EQUAL TO *)
01245	CYRILLIC SMALL LETTER UKRAINIAN IE
3	ALMOST EQUAL TO *)
01245	CYRILLIC SMALL LETTER SHORT U

Table 3: pc855/pc866 splittings
*) for this coding table

WHERE	THE MEANING
23	BOX DRAWINGS DOWN DOUBLE AND LEFT SINGLE
145	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
23	BOX DRAWINGS DOWN DOUBLE AND RIGHT SINGLE
145	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
4	LESS-THAN OR EQUAL TO *)
235	BOX DRAWINGS DOUBLE VERTICAL AND LEFT
1	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
4	GREATER-THAN OR EQUAL TO *)
235	BOX DRAWINGS DOUBLE VERTICAL AND RIGHT
1	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK

Table 4: the symbols look more-or-less like left/right angle quotation
*) for this coding table

WHERE	THE MEANING
3	SUPERSCRIPT TWO
01245	NUMERO SIGN
23	MIDDLE DOT *)
0145	SECTION SIGN
25	LOWER HALF BLOCK *)
134	COPYRIGHT SIGN
235	BOX DRAWINGS LIGHT VERTICAL
14	NOT SIGN
235	BOX DRAWINGS LIGHT VERTICAL AND HORIZONTAL
14	REGISTERED SIGN
235	BOX DRAWINGS LIGHT DOWN AND LEFT
14	PLUS-MINUS SIGN
235	BOX DRAWINGS DOUBLE HORIZONTAL
14	MICRO SIGN
235	BOX DRAWINGS DOUBLE VERTICAL
14	PILCROW SIGN
235	BOX DRAWINGS LIGHT DOWN AND HORIZONTAL
14	EN DASH
235	BOX DRAWINGS LIGHT UP AND HORIZONTAL
14	EM DASH
235	BOX DRAWINGS LIGHT HORIZONTAL
14	DAGGER
235	BOX DRAWINGS DOUBLE DOWN AND RIGHT
14	BULLET
235	LIGHT SHADE
14	HORIZONTAL ELLIPSIS
235	BOX DRAWINGS DOUBLE DOWN AND HORIZONTAL
14	TRADE MARK SIGN
4	NOT EQUAL TO
235	BOX DRAWINGS DOUBLE UP AND HORIZONTAL
1	DOUBLE DAGGER
4	INFINITY
235	BOX DRAWINGS DOUBLE VERTICAL AND HORIZONTAL
1	NOT USED
4	INCREMENT
235	UPPER HALF BLOCK
1	PER MILLE SIGN
012345	NO-BREAK SPACE
4	DIVISION SIGN *)
235	MEDIUM SHADE
1	BROKEN BAR
4	LATIN SMALL LETTER F WITH HOOK
235	DARK SHADE
1	MIDDLE DOT *)
4	ALMOST EQUAL TO *)
235	BLACK SQUARE
1	NOT USED
5	FULL BLOCK *)
1234	DEGREE SIGN
234	SQUARE ROOT
015	SOFT HYPHEN
3	LOWER HALF BLOCK *)
1245	CURRENCY SIGN

Table 5: other symbols
*) for some coding tables

be computed for each known coding table and the largest value must point to the right coding table. It seems to be fast, easy and effective because the most frequently used conjunctions of two characters (less then 5% of all conjunctions) gives more then 50% of bicharacter substrings in Russian text and approximately half of all possible conjunctions which are practically never used in Russian. The “only” problem remaining is to select the sets A and U properly.

How we selected A and U A great help for us was the unique Gilyarovskii and Grivnin book [9] with the text samples on most of the languages. We had to turn the samples into computer files in order to count biletter appearance numbers. A new problem then arose: what should we do with non-Russian letters?

There are no fixed coding tables for most of the languages. We also do not know about any other attempts to use a Russian keyboard and special TEX commands for typesetting of most of the Cyrillic languages of Russia, Mongolia and Alaska. For each of the languages which use non-Russian letters, we have made two files: the first file has char representation of non-Russian letters mostly according to the tables above, and the second file has more-or-less better readable Russian letter sequences following the slash char (such as /ЪК for “K as in beak” or /КЦ for “K as in desk” or /ЛЬ for Љ or /Ц for Џ) and maximal usage of the standard TEX accent control sequences. For the Russian language, we used three different subject topics and a dictionary with 51924 words. Each of the other languages was represented by a single file. We obtained 109 files for 64 languages.

We cannot be certain other people will use the same codes or sequences for non-Russian letters. Therefore, while counting the biletter strings for each file we assign all letters with unknown codes to a group, identify all ASCII non-letters and assign them to another group and assign all Latin letters unusable by Cyrillic text to a separate group. After, counting we selected biletter strings which did not appeared in files. They composed the set U with 695 elements.

The selection of set A was more difficult. After several attempts to select it we got the following algorithm. For each couple of letters and each file, the logarithm of ‘relative frequence’ was computed. To avoid infinity we had zero frequences changed to a small non-zero value, as if this biletter string appears once in a file twice as long. Then we found the sums over all the files and used them for selection. The most frequent 314 couples consist of only Russian letters and almost each word contains at least one of such biletter strings. We had to avoid the effects of possible usage of other TEX names for non-Russian letters, or other coding tables which may correlate only to the Russian part of our coding table. Therefore we used only 306 of these couples without the biletter strings which our special notations for non-russian letters could produce.

In this way, the Cyrillic coding recognition algorithm was finished.

Availability

The METAFONT sources of RF font family and sources of cyrillic coding recognition algorithm will be available from RFBR TEX server via anonymous ftp: `ftp.tex.math.ru`.

Acknowledgements

This work was inspired and supported by Russian Foundation for Basic Research, grant 96-07-89406.

References

[1] A. Chernov. Registration of a Cyrillic Character Set. RFC 1489, RELCOM Development Team, July 1993.

[2] J. Reynolds, J. Postel. Assigned Numbers. RFC 1700, USC/Information Sciences Institute, October 1994.

[3] T.Greenwood, J. H. Jenkins. ISO 8859-5 (1988) to Unicode. Unicode Inc. January 1995.

[4] M. Siugnard, L. Hoerth. cp1251_WinCyrillic to Unicode table. Unicode Inc. March 1995.

[5] M. Siugnard, L. Hoerth. cp10007_MacCyrillic to Unicode table. Unicode Inc. March 1995.

[6] M. Siugnard, L. Hoerth. cp855_DOSCyrillic to Unicode table. Unicode Inc. March 1995.

[7] M. Siugnard, L. Hoerth. cp866_DOSCyrillicRussian to Unicode table. Unicode Inc. March 1995.

[8] P. Edberg. MacOS_Ukrainian [to Unicode]. Unicode Inc. April 1995.

[9] Р.С. Гиляровский, В.С. Гривнин. Определитель языков мира по письменности. Изд-е третье, исправленное и дополненное. М.: Наука, 1964.

Cyrillic TEX files: interplatform portability

Peter A. Ovchenkov
Russia, Moscow 125047, Miusskaya sq. 4, M. V. Keldysh Institute of Applied Mathematics
ptr@ASoft.MSK.SU

Abstract

We consider the problems with regard to Cyrillic encoding while working with different operatining systems. The proposed solutions can simplify the administration and support of TEX on different platforms. The related problem of encoding Cyrillic TEX fonts is also discussed.

Preliminary notes

A number of new Cyrillic encodings have appeared recently. Maybe that's all? But no — the process needs further consideration. Why, indeed, ... is Bill Gates better than me? I would like to suggest trying Cyrillic encoding for TEX fonts. If you don't like it, there are almost as many TEX fonts as Cyrillic encodings. If you multiply these two numbers, you can see fantastic potential for encodings suggestions.

I would also like to talk about the native Russian typesetting, not translitaration. The examples I will use demonstrate only Russian letters; I do not have enough experience to talk about letters specific to other Cyrillic alphabets.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x																
1x																
2x	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x	А	Б	В	Г	Д	И	Ж	З	И	Й	К	Л	М	Н	О	П
9x	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
ax	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
bx																
cx																
dx																
ex	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
fx	Ё	ё														

Figure 1: Cyrillic encoding "alternative" (code page 866).

Problem

First of all, for historical reasons, many Cyrillic encodings were developed for different types of computers (and different operating systems). To simplify, I will speak only about computers with 8 bits/byte and with Latin characters encoded as US-ASCII. For such machines, at least four cyrillic encodings exist. For computers running DOS, the most widely used encoding is the so-called alternative encoding (fig. 1). Its popularity here is because the pseudographic symbols have the same codes as those for extended ASCII. For MS Windows, the original Microsoft encoding "honored Bill Gates" (fig. 2). On UNIX machines two encodings coexist — KOI-8 (fig. 3) and ISO 8859-5 (fig. 4). ISO 8859-5 is the official standard (GOST). Macintoshes use the ISO 8859-5 standard. That's enough without Cyrillic in EBCDIC-like encodings.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x																
1x																
2x	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x																
9x																
ax									Ё							
bx									ё	№						
cx	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
dx	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
ex	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
fx	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Figure 2: Cyrillic encoding in MS Windows (code page 1251).

Let us consider the following problem: you need to process TEX files on computers with different Cyrillic encodings. Don't worry about your `.tex` files: (i) there are enough transcoder programs, and (ii) there are enough program tools (like text editors, etc.) that are outside TEX's control. But

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x																
1x																
2x	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x																
9x																
ax				ё												
bx				Ё												
cx	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
dx	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
ex	Ю	Б	В	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
fx	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	Ч	Ш	Э	Щ	Ч	Ъ

Figure 3: Cyrillic encoding KOI8.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x																
1x																
2x	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8x																
9x																
ax		Ё														
bx	А	Б	В	Г	Д	И	Ж	З	И	Й	К	Л	М	Н	О	П
cx	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
dx	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
ex	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
fx	№	ё														

Figure 4: Cyrillic encoding ISO 8859-5.

what about `.dvi` files and style files? Oh, this is the chief problem for the TEX administrator in a heterogeneous network. He (or she) must:

1. define as letters (catcode 11) Cyrillic letters for all encodings in usage (during format generation)
2. transcode styles with Cyrillic styles (this would preclude sharing LATEX styles via an electronic network)
3. replace fonts with virtual fonts

The first two operations are simple, but if you forget to convert a style to a Cyrillic style, for example, you discover unusual output: and in some cases, the mistake may be very difficult to locate.

Remapping fonts into virtual fonts is also not without problems: for example, what do you do with the kerning between punctuation (codes less than 127) and Cyrillic letters (codes greater than 128)? The use of 8-bit full fonts leads to incompatibility of `.dvi` files prepared on different platforms and dramatically increases the number of fonts.

Input Flow Transformation

Let us consider a hypothetical 8-bit font: the first half is Latin symbols (symbols with codes 00_h–79_h), punctuation, and digits, etc., and the second half contains Cyrillic letters. Let define the font encoding as E_f. For TEX (the binary executable program) we can define a transformation κ, such that $E_e \xrightarrow{\kappa} E_i$, where E_e is a character encoding that TEX "sees" as input flow (i.e., "native" encoding of a (soft)hardware system), and E_i — TEX internal encoding.

If we define $E_i \equiv E_f$, we can use the same fonts during editing and previewing on one machine type, and printing on another one. (This is obvious for systems with Latin typesettings, but not for Cyrillic!) We will had that a `.dvi` file (created on a PC) can be converted by dvips on SPARCstation without problems.

In the input flow you can refer directly to a symbol in "internal" encoding (in E_i representation). This fact allows one to create style files that can be used on different platforms simultaneously.

Input Flow Transformation in emTEX In emTEX, Eberhard Mattes suggests a simple way to set up κ transformation. This is the creation of TEX Code Page to then include this information in the precompiled format file.

The transformation is described in the `.mtc` file. Below is the beginning of the file for the alternative — experimental (compare figures 1 and 5) encoding:

```
%
% alt_abs.mtc
%
^^80 ^^c0
^^81 ^^c1
^^82 ^^c2
^^83 ^^c3
^^84 ^^c4
^^85 ^^c5
^^86 ^^c6
^^87 ^^c7
^^88 ^^c8
^^89 ^^c9
^^8a ^^ca
^^8b ^^cb
^^8c ^^cc
^^8d ^^cd
^^8e ^^ce
^^8f ^^cf
```

Here, the first column represents the character code that TEX see as input flow, and second is the one inside TEX. In the second column you can write TEX commands instead of character codes.

Next the `.mtc` file is compiled into the `.tcp`file:

```
C:\EMTEX\DATA> maketcp -c -8 alt_abs.mtc
```

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x	ˋ	´	ˆ	˜	¨	˝	˚	ˇ	˘	¯	˙	¸	˛	‚	‹	›
1x	“	”	„	«	»	–	—		₀	ı	ȷ	ff	fi	fl	ffi	ffl
2x	␣	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	˜	-
8x						Ё										
9x																§
ax						ё										
bx											№			¡	¿	£
cx	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
dx	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
ex	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
fx	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Figure 5: Experimental font encoding ($E_i \equiv E_f$).

The code table is turned on while the format file is being created:

```
C:\EMTEX\LATEX\BASE> latex -i -8 \
   -c alt_abs -mt15000 -mp65500 latex.ltx
```

There are some other options that you can find in the emTEX documentation.

Input flow transformation in UNIX-like systems On UNIX-like systems, TEX is traditionally compiled from `WEB` sources. In many cases this is done via `CWEB`. Here it seems there is no simple way, as there is for emTEX. One is forced to program; but for UNIX you always can find a C-compiler (at least).

While TEX is translated from `WEB` to C, there are applied patches from the file `ctex.ch`. You can create your own variant of this file with minor additions.

First we remap the input flow (the example below illustrates the remapping from ISO 8859-5 encoding into the experimental one – see figures 4, 5):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [2.23] Allow any character as input.
%        (Remapping by ptr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@x
for i:=0 to @’37 do xchr[i]:=’ ’;
for i:=@’177 to @’377 do xchr[i]:=’ ’;
@y
for i:=0 to @’37 do xchr[i]:=chr(i);
for i:=@’177 to @’237 do xchr[i]:=chr(i);
for i:=@’240 to @’357 do
                        xchr[i+16]:=chr(i);
for i:=@’360 to @’377 do
                        xchr[i-64]:=chr(i);
xchr[@’205]:=chr(@’241);{\Yo}
xchr[@’245]:=chr(@’361);{\yo}
@z
```

Substitute printable symbols for the non-printable ones, (i.e., TEX substitutes on the terminal display, and xwrites in the `.log` file:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% [?.??] Modify characters cannot be
%        printed -- Ptr.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@x
@<Character |k| cannot be printed@>=
  (k<" ")or(k>"~")
@y
@<Character |k| cannot be printed@>=
  (k<" ")or((k>"~")and(k<@’300)
           and(k<>@’270)and(k<>@’271))
@z
```

After that you can build TEX in the normal way.

Fonts Encoding

But there is one small thing: now we have as many font encodings as machine ones. Now’s the time to find a uniform Cyrillic font encoding.

The problem connected with letters of national alphabets in Europe was solved in part with unification of TEX fonts. The arguments, pro and contra, connected with the Cork encoding scheme, you can find in various *TUGboats*, or in the *LATEX Companion*.

There is no more room for Cyrillic in the Cork encoding. That’s evident. But the experience of `dc`-fonts is good (in my opinion). So I think that a good solution is encoding similar to that in figure 5. The first half is equivalent to the first half of the Cork encoding, and Cyrillic is placed in the second half. Why is there no ISO- or alternative-like encoding here? This is due to the LATEX 2$_\varepsilon$ requirement for the difference between codes for upper- and lowercase of the same letter to be 32, and a sequence of letters beginning with code 128 or 192. There is no evident preference for any machine encoding of Cyrillic, but with an encoding as seen in figure 5 we can avoid some of the problems with LATEX 2$_\varepsilon$.

Cyrillic in Style Files

TEX allows direct references on internal TEX encoding (`^^hh`, `hh` is a pair of hex digits). This fact can be used for writing files for the precompiled format or for styles: such files depend only upon font encoding, but not upon the Cyrillic encoding on computer.

For precompiled format files we can write definitions for Russian letters and then use this file on every computer:

```
\def\letter#1#2{%
    \catcode‘#1=11\catcode‘#2=11%
    \uccode`#1=`#1\lccode`#1=`#2%
    \uccode`#2=`#1\lccode`#2=`#2%
}
\language=\l@russian
\letter{^^c0}{^^e0}%
\letter{^^c1}{^^e1}%
\letter{^^c2}{^^e2}%
\letter{^^c3}{^^e3}%
```

Or, we can do the same with styles:

```
\addto\captionsrussian{%
  \def\prefacename{%
^^cf^^f0^^e5^^e4^^e0^^f1^^eb^^ee^^e2%
^^e8^^e5}%
  \def\refname{%
^^cb^^e8^^f2^^e5^^f0^^e0^^f2^^f3^^f0^^e0}
  \def\abstractname{%
^^c0^^ed^^ed^^ee^^f2^^e0^^f6^^e8^^ff}%
  \def\bibname{%
^^c1^^e8^^e1^^e8^^eb^^e8^^ee^^e3^^f0%
^^e0^^f4^^e8^^ff}%
  \def\chaptername{^^c3^^eb^^e0^^e2^^e0}%
  \def\appendixname{%
^^cf^^f0^^e8^^eb^^ee^^e6^^e5^^ed^^e8^^e5}
  \def\contentsname{%
^^d1^^ee^^e4^^e5^^f0^^e6^^e0^^ed^^e8^^e5}
  \def\listfigurename{%
^^d1^^ef^^e8^^f1^^ee^^ea %
^^f0^^e8^^f1^^f3^^ed^^ea^^ee^^e2}%
  \def\listtablename{%
^^d1^^ef^^e8^^f1^^ee^^ea %
^^f2^^e0^^e1^^eb^^e8^^f6}%
  \def\indexname{^^c8^^ed^^e4^^e5^^ea^^f1}
  \def\figurename{%
^^d0^^e8^^f1^^f3^^ed^^ee^^ea}%
  \def\tablename{%
^^d2^^e0^^e1^^eb^^e8^^f6^^e0}%
  \def\partname{^^d7^^e0^^f1^^f2^^fc}%
  \def\enclname{^^e2^^ea^^eb.}%
  \def\ccname{^^e8^^e7}%
  \def\headtoname{^^e2}%
  \def\pagename{%
^^f1^^f2^^f0^^e0^^ed^^e8^^f6^^e0}%
  \def\seename{^^f1^^ec.}%
  \def\alsoname{^^f1^^ec.~^^f2^^e6.}%
}
```

Of course, this is not very readable, but this is done by a style developer, and only once for every machine-specific encoding of Cyrillic fonts.

One note: this mechanism cannot be applied to hyphenation patterns. TEX doesn't expect E_i-symbols representation (`^^hh`) inside command `\patterns`.

A User-Friendly Multi-Function TeX Interface Based on Multi-Edit

Michael M. Vinogradov
Institute of Economic Forecasting, Moscow
`vin@ecfor.msk.su`

Abstract

The aim of this paper is to describe the set of Multi-Edit macros created for easily processing of TeX files. Note that the standard version of Multi-Edit has some tools to work with TeX files, but these tools are not very convenient. Therefore we have created our own set of macros, which allows us to process the whole file or to compile and view any selected block of text, such as complicated formulas, for example. Also, there are additional tools for handling some standard situations.

On-Screen Compile Menu

The **Compile menu** is activated by pressing the corresponding key. This menu includes:

1. Compiling for formats: Plain-TeX, AMS-TeX, LaTeX or AMS-LaTeX. One can also use various compilers, for example, 286-compiler and big-compiler. These menu items are realized by calls of corresponding batch-files. Automatic processing can be used to find compile errors;
2. Variants of a view. In particular, one can use a view with manual input of options;
3. Variants of printing, similar to the view ones. Various output devices can be used, such as dot matrix and laser printers;
4. Additional menu items; for example, one can include a call of Russian spell-checker.

It is easily to add, to modify or to delete menu items by using standard Multi-Edit tools.

Automatic error handling

Errors identified during compiling are handled in the following way:

- The cursor is positioned at the location of the first error found;
- The corresponding error message appears in the upper line of the window;
- The log-file displayed in the OUTPUT window points to the corresponding error message, which shows only the first two lines. If this is insufficient, then the cursor in the OUTPUT window can be moved by pressing ⟨F11⟩ or ⟨Alt+Esc⟩. In this case the OUTPUT window will increase up to seven lines. To move the cursor in the text window, press ⟨F11⟩ or ⟨Alt+Esc⟩ again and the OUTPUT window will decrease up to two lines.

To find the next compile error, simply press the ⟨NxtErr⟩ key.

Block Compile

To compile and view a portion of a text (for example, a complicated formula), select the fragment and use a special macro, invoked by pressing the corresponding combination of keys. Various TeX formats (Plain-TeX, AMS-TeX, LaTeX or AMS-LaTeX) may be used also. The selected block may contain user-defined TeX macros. These macros should be placed on the beginning of the file and the line

```
%%% End of Leading Block
```

after them. How many of the signs `%` is not important. All definitions found above this line will be added into the selected block when it is compiled. If this line is absent, no macros and definitions will be added. If additional definitions are placed in the another file and then included via either `\input` or `\documentstyle`, etc., these commands also should be placed above the percented line just mentioned. If one needs to omit some commands, then the corresponding lines should be marked

```
%%% Skip line
```

For example, the typical beginning of a LaTeX file may be marked as follows:

```
\documentstyle[12pt,fleqn,draft]{article}
\newcommand\smbl{{\rm smbl}}
\newcommand\al{alpha}
\input footline.tex %%% Skip line
\begin{document}
%%% End of Leading Block
```

After the file has been compiled, the logfile containing the selected block and error messages appears in the OUTPUT window.

Additional macros

Additional possibilities can be divided into two types: macros that are useful for all TEX users and those useful for Russian TEX users only.

For all TEX users.

1. A database can be used to insert some TEX commands and macros. This database contains a set of commonly used macros for Plain-TEX, AMS-TEX and LATEX. Commands may be inserted on the cursor place or so as to surround a selected block.
2. Users can locate opening and closing parentheses or curly brackets, as well as check for any missing opening or closing brackets.

For Russian TEX users.

1. For those whose English-language skills need assistance, there is a a special database containing more than five hundreds basic sentences often used when writing English mathematical papers. The idea is adapted from Sosinsky. A part of this book is included in the HELP system to explain how to use the database.
2. Should the Russian TEX user forget to swich the keyboard layout, the file can still be processed without re-typing the text in the proper register.
3. Capitalization of a word, line or block can be changed to either upper- or lowercase. This macro works independently of the alphabet in use (Russian or English).

Conclusion

Most of these abilities can also be added to Multi-Edit for Windows. Thus we offer the TEX user a friendly multi-function Windows interface.

References

Sosinsky, A. *How to Write a Mathematical Paper in English.*

Full Cyrillic: How Many Languages?

Olga G. Lapko
Russia, 129820, Moscow
Pervy Rizhsky per., 2
Mir Publishers
`olga@mir.msk.su`

A Brief History of Cyrillic

The Slavonic writing was invented by St. Cyrill and St. Method. Now there are two well-known Slavonic writings: Glagolitic and Cyrillic.

Historians are not sure whether the author of both was St. Cyrill or whether Cyrillic script was created by St. Method while St. Cyrill invented the Glagolitic alphabet. In any case, in this paper we will deal about the alphabet nowadays called "Cyrillic."

The birthday of Cyrillic is considered to be the end of May 863. May 24 was declared by UNESCO as the day of Cyrillic. By coincidence, the first conference of the Cyrillic TeX Users Group, *CyrTUG*, was held May 24–25, 1991.

The Cyrillic alphabet is based on the Greek alphabet. There were 43 letters in this alphabet. Up until the beginning of the 20th century, four additional letters existed; these are absent in the modern Russian alphabet: 'ѵ', 'ѳ', 'ѣ', and 'і'.

Nowadays Cyrillic script is used not only by Slavonic people, but also by other nations of the former USSR. Historically, many of these nations used other scripts. Some Soviet republics such as Middle Asian republics, Azerbaijan, and the Russian autonomic republics, used the Arabian script. In Siberia, the old Mongolian vertical script was used in the Buryat language, and the Dzayapandin vertical script was used in the Kalmyk language. Soon after the October Revolution many languages started to use the Latin script with additional letters. In Abkhasia, the Georgian alphabet was used for a few years. At the end of 20s and 30s, almost all languages of the USSR changed from using Latin script to Cyrillic. In many languages new letters were created (see fig. 1).

Outside Russia, Cyrillic is used in Bulgaria, Serbia, Macedonia and Mongolia. The Bulgarian language now uses only letters of the modern Russian alphabet, but earlier 'ѫ' and 'ѣ' were also used. In Macedonian and Serbian the following additional letters are used: 'ј', 'љ', 'њ', with 'ѓ', 'ќ', 'ѕ' in Macedonian only, and 'џ', 'ћ', 'ђ' in Serbian. The Mongolian language uses Russian letters with two additions: 'ө', 'ү'.

One may also find Cyrillic letters used in scripts based on the Latin alphabet. Examples are the Chinese languages: Y, Lahu, Lisu, Myao, Juang, as well as several African languages.

History of the full Cyrillic font project

The LHFONTS[1] package was created as a part of the *CyrTUG*-EmTeX package, which is distributed among Russian and non-Russian users who use Cyrillic. LHFONTS offers the `LH` Cyrillic font family; these fonts are based on the WNCYR fonts of the CYRILLIC package — part of AMS-TeX.

The main task of the LHFONTS package was to create the Cyrillic fonts family, an extension of standard text fonts of Computer Modern, which also corresponds to Russian typesetting traditions.

First this package offered two more or less popular encoding schemes:[2] Alternative — an 8-bit Latin-Russian font encoding analogous to MS-DOS's Code Page 866, mainly used by Russian MS-DOS users; and the Washington or WNCYR — a 7-bit encoding for typesetting with transliteration, which is mainly used by non-Russian users.

The Cyrillic character encodings are described in special files — `lbcoding.mf` (Alternate encoding) and `wncoding.mf` (WNCYR encoding). One can choose between these files by changing the value of one of the following variables: `altcoding`, `vfcoding` or `wncoding`. These variables also determine the font layout:

`altcoding` Standard Computer Modern in the lower part of table plus Russian letters and additional punctuation marks in the upper one; Alternate encoding: encoding file `lbcoding.mf`

[1] This package was originally named MAKEFONT, but was renamed to avoid confusion with the utility of the same name on the 4AllTeX CD-ROM, produced by the Nederlandstalige TeX Gebruikersgroep.

[2] The package also offered virtual encoding: rather conditional 7-bit encoding which combines Cyrillic and Latin fonts.

`vfcoding` Russian letters and added punctuation marks in the upper part of table — for the following combining with Computer Modern in virtual font; Alternate encoding: encoding file `lbcoding.mf`

`wncoding` Cyrillic letters for Slavonic languages with necessary input ligatures, standard and additional punctuation marks in the lower part of table; WNCYR encoding: encoding file `wncoding.mf`

The values of these variables are determined in driver files (`ld??font.mf`) by default. The header of these files (for ex. `ldrmfont.mf`) contains the following lines:

```
if unknown wncoding: wncoding:=0; fi
if unknown vfcoding: vfcoding:=0; fi
...
altcoding:=1-wncoding-vfcoding;

if wncoding<>0: input wncoding;
else: input lbcoding; fi
...
```

Variables `altcoding`, `vfcoding` and `wncoding` may be set by hand in the file header or at the start of the METAFONT run.

The LHFONTS package contains font headers named `lh*.mf` and `ll*.mf`

The files `lh*.mf` (56 files) are virtually identical to `cm*.mf` except for the last line:

`generate` <*driver-file*>

that is, the standard Computer Modern *driver file* was changed to the analogous file for the LH fonts. These file headers generate a full 8-bit Latin-Cyrillic font.

The files `ll*.mf` (also 56 files) contain only the following line:

`vfcoding=1; input` <*header-file lh**>`;`

thus the command `vfcoding:=1;` sets generation of Russian letters and punctuation marks only.

Since the WNCYR encoding was an optional encoding in this package, the files `wn*.mf` were not created, but documentation explains how to create fonts with the WNCYR encoding using a similar one-line header file as follows:

`wncoding=1; input` <*header-file lh**>`;`

The LH Cyrillic font family offers typesetting in Russian and other languages using the Russian part of the Cyrillic alphabet — that is Virtual and Alternate encoding.

The WNCYR encoding also offers typesetting in modern Slavonic texts using Cyrillic and 19th century Russian text. But there are a lot of languages which use the Cyrillic alphabet with added letters. The following sections discuss this problem.

The Global Cyrillic font as the material for Ω project

Some time ago the multilingual project Ω was started. One of the authors of Ω, Yannis Haralambous, began to create a full Cyrillic font. He offered this font to *CyrTUG* for further work.

The data for this font was taken from the Cyrillic part of the Unicode table.[3] But this table still does not cover all Cyrillic letters; some old Cyrillic letters and national letters are missing. Probably the full assortment of accented vowels is necessary, which are not included in Unicode.

The font created during this work had more than 256 letters and marks. This font assortment should be further extended and improved. During the testing of this font, I created shortened variants, or split it into a few fonts. The two Cyrillic Unicode fonts were extended with glyphs for characters not included in Unicode and created by the methods described in this paper (see the Appendix).

The methods of partial font creation may usefully improve the economy of use of the computer's memory. One may create a big 256-letter (Unicode) font and then use a virtual font to achieve the necessary encoding. Alternatively, one may create the font immediately in its required encoding.

How TeX helps METAFONT. Creation of coding and ligature-kerning tables

To create a font, METAFONT needs program descriptions for letters (a lot of them), information about lettercodes, and kerning and ligature data.

Now there are a few well-known encodings of Cyrillic. They differ in which characters they hold and in what order (see the Appendix). So, for every encoding, a separate file is needed.

Since TeX cannot use a font containing ligatures or kerning information relating to external characters, we cannot use the same table, with all Cyrillic letters, for every font: we must create a separate table for each encoding. There are five tables for the different font shapes of the text fonts of the Computer Modern family: they are included in the driver files. We must create the same number of tables for every encoding.

[3] *Unicode* — International Standard ISO/IEC 10646–1, first edition, 1993.05.01. Information Technology — Universal Multiple Octet Coded Character Set (UCS) Part 1: Architecture and Basic Multilingual Plane (Table 11, Row 04, Cyrillic).

Furthermore, for each font and each encoding we must create a header file (the Computer Modern family has 56 text fonts).

As you can see the number of files will be very large, and they will often duplicate each other.

The best solution is to create three files which contain all the information that could potentially be duplicated. The first one contains a table with all the Cyrillic glyphs and signs and all supported encodings. The second one contains data on ligature and kerning for the complete font repertoire. The third file contains the table of font names and sizes and the necessary command lines for every font header file. The data for every font in every encoding would be taken from these files and the necessary header files created. All these files are created by TeX. TeX also can create the file containing all uccodes, lccodes and mathcodes for a given font.

Preparing font headers As mentioned above, we use the parameters of the Computer Modern text fonts for creating Cyrillic fonts. First, the header files for the LHFONTS package were copied from header files of the Computer Modern family, changing the only line (See the section entitled "History of the full Cyrillic font project"). To avoid unnecessary duplication, we create header files which load the necessary Computer Modern header file, substituting the standard driver file with the driver of LH Cyrillic fonts. This task, for example, was solved in the Polish fonts package, in there is a special tricky file `fik_mik.mf` which substitutes standard drivers with Polish ones. One of the authors of this file, Boguslav Yackovski, allowed us to use this file in the LHFONTS package.

Now it is necessary to create header files for font creation which include one or a few lines only.

The EmTeX package supports a `command` operator in its `MFJob` program, which enables one to write necessary short commands for the METAFONT run. By using this, we can avoid creation of a lot of files with the only line:

```
input fik_mik_; use_driver;
```

For font generation on other platforms we must create these files in any case. For quick generation of the files I used the file `dcstdedt.tex` from DCFONTS. The original file includes the table with all font names and font sizes. We modified the file, providing a possibility to add the line (`\mainfontspecific` macro), which can switch on variables `vfcoding:=1;` or `wncoding:=1;` when necessary, and the parameter for a few fonts, which switch on necessary shape. To specify usage of different encodings we must change the font names, so the first two letters are changed to macro `\fonttwoletters`; these two letters are set by the user according to the necessary encoding at the start of TeX's run. A fragment of such a file for font headers is shown below:

```
% by default full Cyrillic Font (Unicode)
% is generated
\ifx\mainfontspecific\undefined
 \def\mainfontspecific{vfcoding:=1;}\fi
\ifx\fonttwoletters\undefined
 \edef\fonttwoletters{uc}\fi

\long\def\FontsToBeGenerated{
\tablevalues                                 %
    ( ... 8 9 10 ... 17.28[17] )             %

\makefont\fonttwoletters r                   %
    ( ... 8 9 10 ... 17.28[17] )()
    ...
\makefont\fonttwoletters tt                  %
    ( ... 8 9 10 ...          )(specific:=0;)
}
```

By default there is creation of a set of file headers and encoding for the global Cyrillic font in Unicode (see fig. 1) in this file and the file of encoding data.

Preparing the encoding file and files of ligature/kerning tables The encoding files are created from the file which contains the table of all Cyrillic glyphs and signs and all well-known (or at least necessary) encodings.

```
% by default full Cyrillic Font (Unicode)
% is generated
\ifx\fonttwoletters\undefined
 \def\fonttwoletters{uc}\fi
\def\nolettercode{*}
\long\def\CodesToBeGenerated{
\tablevalues              ( uc lh wn     ... )

\makecod CYR_A    CYRA    ( 10 80 41[A]  ... )
\makecod CYR_BE   CYRB    ( 11 81 41[B]  ... )
...
\makecod CYR_LJE  CYRLJE  ( 09 *  01[LJ] ... )
}
```

We can see that this table is analogous to the previous one. Macros, analogous to macros for creating font headers, were used for creating the encoding files.

Now we must create tables of ligatures and kerns. In the Computer Modern fonts these tables are in the font driver files. In the LH fonts the tables are in separate files. As we said above, we need to create five tables of ligatures and kernings for text fonts: 1) for roman and sans serif shape; 2) for italic shape; 3) for caps and small caps shape, for which two tables are actually necessary, separately for the uppercase and small saps letters; and 4) for large fonts like `cminch`.

The Cyrillic font is very large, but one may see that almost all Cyrillic letters can be identified in a few shape groups. We determined 14 groups for uppercase letters, 14 groups for lowercase letters and 17 groups for italic letters in Cyrillic font. The letters in these groups sometimes repeat the shape (or contour) of Latin ones, so one may use the Computer Modern table as a base.

The file of kerning and ligature data retains the shape grouping of letters, so every new letter will be added to an appropriate group. At the beginning of each group we place a "typical" Latin letter as a comment. When the new letter appears it may be added into a necessary group:

```
\writeLig{if wn:}
\writeLig{ ligtable CYR_ZE: "1"=:CYR_ZHE,
 "H"=:CYR_ZHE, "h"=:CYR_ZHE;} % "Z"
\writeLig{fi}

\Ligtab %A
\Letter{CYR_A} \Letter{CYR_A_acute}
\Letter{CYR_LIT_YUS}
...
%b
\Letter{CYR_HARD_SIGN}\Letter{CYR_YATZ}
...
%R
\WriteLig{if serifs:}
\Letter{CYR_BIG_YUS}
...
\WriteLig{fi}
%
%O
\Kern{CYR_O}{k#}\Kern{CYR_O_lcomma}{k#}
\Kern{CYR_O_acute}{k#}
...
\Kern{CYR_ABKH_O}{k#}
%
...
\EndLigtab
```

We may create a font for kern testing by taking the characteristic examples from these letter groups (see Appendix).

For creation of the necessary ligature and kerning pair tables, we use data from the encoding file which was created just before them. Now the ligatures are used in WNCYR encoding only without any changes from the original Washington State University fonts.

In addition to the tables of ligature/kerning, TeX creates a uccode/lccode/mathcode file and a file `???cod.tex`, which is used by `russianb.ldf`[4].

How METAFONT generates only necessary letters TeX has thus created files necessary for encoding and ligature and kerning pair tables. Now METAFONT must generate only the necessary glyphs required for the given encoding.

From the very beginning the `LH` font family supported different encodings. Since in different coding schemes Cyrillic letters occupy different places, in character descriptions (`beginchar` command), explicit character codes have been replaced with their symbolic names. For example, the description of the lowercase Cyrillic letter 'a' starts with:

```
cmchar "Lowercase Russian letter a";
beginchar(CYR_a,9.25u#,x_height#,0);
...
```

In the description of uppercase letters we added the line: `if lower_case: ... fi` for redefinition of a code in the font "Small Caps":

```
cmchar "Uppercase Russian letter A";
beginchar(CYR_A,13u#,cap_height#,0);
if lower_case: charcode:=CYR_a; fi
...
```

In plain METAFONT the definition of the `beginchar` command has the following lines:

```
def beginchar(expr c,w_sharp,h_sharp,d_sharp) =
 begingroup
 charcode:=if known c: byte c else: 0 fi;
 ...
 enddef;
```

which means that a letter with an unrecognized code number is set to position '0'.

For the `LH` fonts, a letter or a sign whose code is not recognized must be skipped, so the `beginchar` command is redefined in the following way:

```
let plain_beginchar=beginchar;

def beginchar(expr c,w_sharp,h_sharp,d_sharp) =
iff known c: %
plain_beginchar(c,w_sharp,h_sharp,d_sharp);
enddef;
```

What needs to be done when it is necessary to create the Cyrillic font only, but letters and signs of standard Computer Modern have got code numbers in `beginchar` so they are always determined? The three variables which were mentioned in the section on the History of the Full Cyrillic Font Project, determined three different encodings. Now they may switch on the following:

[4] The Russian language-specific file for the Babel system.

`altcoding` Standard Computer Modern in the lower part plus Cyrillic letters and added punctuation marks in necessary encoding in the upper one

`vfcoding` Cyrillic letters and added punctuation marks in the upper, lower, or in both parts of the table — for next combining with Latin part in virtual font or for full Cyrillic font creation (for example Unicode)

`wncoding` this set was not changed — Cyrillic letters for Slavonic languages with necessary ligatures, standard and additional punctuation marks in the lower part of the table; WNCYR encoding

Now the encoding files, `lbcoding.mf` or `wncoding.mf` switched on by these variables, set necessary selection of Cyrillic letters and their encoding. In fact, the file `wncoding.mf` for WNCYR encoding was not changed. The file `lbcoding.mf` switches the necessary encoding. When we have the necessary files, we can create the font.

References

[1] К. М. Мусаев, "Алфавиты языков народов СССР", Москва, "Наука", 1965.

[2] Р. С. Гиляревский, В. С. Гривнин, Определитель языков мира по письменностям, Москва, "Издательство восточной литературы", 1960.

[3] Е. И. Убрятова, Некоторые вопросы графики и орфографии письменности языков народов СССР, пользующихся алфавитами на русской основе, Москва, 1959.

[4] Московская синодальная типография, Образцы литеръ церковныхъ, российскихъ, греческихъ, латинскихъ, грузинскихъ, еврейскихъ, немецкихъ и прочихъ, находящихся въ Московской синодальной типографіи, Москва, 1826.

[5] Образцы шрифтов, Узбекская ССР. Совет народных комиссаров, Юридическое издательство, Типография, Самарканд, 1928.

[6] Татарский язык и новые информационные технологии. Выпуск 2, Издательство Казанского университета. 1995.

[7] Языки народов СССР, 5 т., Москва, 1966–68.

[8] Andrei B. Khodulev and Irina A. Makhovaya, "On TEX experience in Mir Publishers", *Proceedings of the 7th EUROTEX Conference*, Prague, pp. 37–43, 1992.

[9] Olga G. Lapko, "MAKEFONT as part of *CyrTUG*-EmTEX package", *Proceedings of the 8th EUROTEX Conference*, Gdańsk, Poland, pp. 110–114, 1994.

[10] Fry Edmund, "Pantographia containing accurate copies of all the known alphabets in the world together with an English explanation of the regular force or power of each letter, to which are added specimens of all well-authenticated oral languages; forming a comprehensive digest of phonology", Cooper and Wilson, London, 1799.

[11] Katzner Kenneth, "The languages of the world", London, Henley:

[12] The World's major languages, ed. by Bernard Comrie, London, Sydney, 1987. Rout ledge& KeganPaul, 1977.

[13] Y. Haralambous, J. Plaice, "Typesetting in the Cyrillic alphabet with Ω — The Basic Ideas", August 24, 1994.

[14] DC-Fonts, Beschreibung der Kodebelegung: TEX 256 Zeichen — internationaler Zeichensetz, 22. März 1992.

A Appendix

Figure 1: Unicode encoding; Cyrillic part

0:		Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ		Ў	Џ
16:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
32:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
48:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
64:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
80:		ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ		ў	џ
96:	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
112:	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
128:	Ҁ	ҁ	҂	҃	҄	҅	҆									
144:	Ґ	ґ	Ғ	ғ	Ҕ	ҕ	Җ	җ	Ҙ	ҙ	Қ	қ	Ҝ	ҝ	Ҟ	ҟ
160:	Ҡ	ҡ	Ң	ң	Ҥ	ҥ	Ҧ	ҧ	Ҩ	ҩ	Ҫ	ҫ	Ҭ	ҭ	Ү	ү
176:	Ұ	ұ	Ҳ	ҳ	Ҵ	ҵ	Ҷ	ҷ	Ҹ	ҹ	Һ	һ	Ҽ	ҽ	Ҿ	ҿ
192:	Ӏ	Ӂ	ӂ	Ӄ	ӄ			Ӈ	ӈ			Ӌ	ӌ			
208:	Ӑ	ӑ	Ӓ	ӓ	Ӕ	ӕ	Ӗ	ӗ	Ә	ә	Ӛ	ӛ	Ӝ	ӝ	Ӟ	ӟ
224:	Ӡ	ӡ	Ӣ	ӣ	Ӥ	ӥ	Ӧ	ӧ	Ө	ө	Ӫ	ӫ			Ӯ	ӯ
240:	Ӱ	ӱ	Ӳ	ӳ	Ӵ	ӵ			Ӹ	ӹ						

Since the Latin part is unchanged and uses the TEX encoding scheme the next examples show only the Cyrillic part of the font.

Figure 2: Cyrillic letters which are not included in Unicode

0:	У̀	ғ	Ж̦	ж̦	З̧	з̧	К̦	к̦	Н̦	н̦	С̧	с̧	Х̦	х̦		
16:	З̌	з̌	Ꙋ	ꙋ	С̡	с̡	Δ	▵			Ꙗ	ꙗ	Ԏ	ԏ		
32:	Г̦	г̦	Л̦	л̦	І̦	і̦	Т̦	т̦	О̦	о̦	У̓	у̓	У̀	у̀	Э̄	э̄
48:	Ю̄	ю̄	Я̄	я̄	Ѳ̄	ѳ̄	Ү̄	ү̄								
64:	˘	˘̈	¨	ˊ	҃	⁀	ˢ	”	‟	’	,	ʻ	҅	ˍ́	҄	˘
80:	А́	а́	Е́	е́	Ё́	ё́	Є́	є́	И́	и́	І́	і́	Ї́	ї́	О́	о́
96:	У́	у́	Ы́	ы́	Ъ́	ъ́	Э́	э́	Ю́	ю́	Я́	я́	Ү́	ү́	Ә́	ә́
112:	Ѳ́	ѳ́	Ӕ́	ӕ́												
128:	Ѣ́	ѣ́	Ѵ́	ѵ́	Ѫ́	ѫ́	Ѧ́	ѧ́	Ѭ́	ѭ́	Ѩ́	ѩ́	Ѥ́	ѥ́	Ꙋ́	ꙋ́
144:	Ꙋ́	ꙋ́														
160:	«	»	№													
176:																
192:																
208:																
224:																
240:				ı												

Figure 3: MS DOS cp866 encoding

128:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
144:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
160:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
176:																
192:																
208:																
224:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
240:	Ё	ё	Є	є	Ї	ї	Ў	ў			«	»	№			

Figure 4: Washington encoding

0:	,	Љ	Џ	Э	I	Є	Ђ	Ћ	њ	љ	џ	э	і	є	ђ	ћ
16:	Ю	Ж	Й	Ё	Ѵ	Ѳ	Ѕ	Я	ю	ж	й	ё	ѵ	ѳ	ѕ	я
32:	¨	!	”	Ѣ	˘	%	´	’	(	)	*	ѣ	,	-	.	/
48:	0	1	2	3	4	5	6	7	8	9	:	;	«	ı	»	?
64:	˘	А	Б	Ц	Д	Е	Ф	Г	Х	И	Ј	К	Л	М	Н	О
80:	П	Ч	Р	С	Т	У	В	Щ	Ш	Ы	З	[	“	]	Ь	Ъ
96:	‘	а	б	ц	д	е	ф	г	х	и	ј	к	л	м	н	о
112:	п	ч	р	с	т	у	в	щ	ш	ы	з	–	—	№	ь	ъ

Figure 5: KOI-8 (Unix platform) encoding

128:					«	»	№									
144:																
160:				ё												
176:				Ё												
192:	ю	а	б	ц	д	е	ф	г	х	и	й	к	л	м	н	о
208:	п	я	р	с	т	у	ж	в	ь	ы	з	ш	э	щ	ч	ъ
224:	Ю	А	Б	Ц	Д	Е	Ф	Г	Х	И	Й	К	Л	М	Н	О
240:	П	Я	Р	С	Т	У	Ж	В	Ь	Ы	З	Ш	Э	Щ	Ч	Ъ

Figure 6: ISO 8859-5 encoding

128:	«	»														
144:																
160:		Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ		Ў	Џ
176:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
192:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
208:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
224:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
240:	№	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ		ў	џ

Figure 7: Apple Macintosh encoding

128:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
144:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
160:			Ґ					I				Ђ	ђ	ҭ	Ѓ	ѓ
176:					і		ґ	Ј	Є	є	Ї	ї	Љ	љ	Њ	њ
192:	ј	Ѕ						«	»			Ћ	ћ	Ќ	ќ	ѕ
208:									Ў	ў	Џ	џ	№	Ё	ё	
224:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
240:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	

Figure 8: Windows 1251 encoding

128:	Ђ	Ѓ		ѓ							Љ		Њ	Ќ	Ћ	Џ
144:	ђ										љ		њ	ќ	ћ	џ
160:		Ў	ў	Ј		Ґ			Ё		Є	«				Ї
176:			I	i	ґ				ё	№	є	»	ј	Ѕ	ѕ	ї
192:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
208:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
224:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
240:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Figure 9: Example of national encoding: Tatar encoding

128:	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
144:	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
160:	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
176:																
192:																
208:																
224:	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
240:	Ё	ё	Ә	ә	Ө	ө	Ұ	ұ	Җ	җ	Ң	ң	Һ	һ	«	»

The Latest Developments in Ω

John Plaice
Département d'informatique, Université Laval, Ste-Foy (Québec) Canada G1K 7P4
John.Plaice@ift.ulaval.ca

Yannis Haralambous
Atelier Fluxus Virus, 187, rue Nationale, F-59 800 Lille, France
Yannis.Haralambous@univ-lille1.fr

Abstract

The Ω system has been available since early 1995, and has been used experimentally in several sites around the world. We gather here some conclusions from this experimenting and explain what aspects will be included in version 1.3 of Ω, which should be the first large-scale release of the system. Not only will the portability and performance of Ω be improved substantially, but new features, including smart fonts and multi-directional support, will be included.

When Ω was first conceived, the primary objectives were to remove the 8-bit restrictions imposed by the original design of TEX (number of characters, fonts, registers of each kind, etc.), as well as to offer the means necessary for multilingual typesetting, no matter how complex the script.

The 8-bit restrictions were removed quite easily by simply doubling the size of all data structures in the TEX program and by introducing a variant of the `tfm` file, called the `xfm` file, in which fonts of up to 65,536 characters could be built.

For typesetting complex scripts, such as classical Arabic or Hebrew, a series of finite state automata, called Ω Translation Processes (ΩTPs), can be successively applied to the input character stream to do arbitrarily complex manipulations. After each application of an ΩTP, the macro-expansion facilities of TEX are reinvoked, which means that the full power of TEX is available every time an ΩTP is used.

Performance and portability

The current version of Ω currently resides on the `ftp.ens.fr` server, and has been used experimentally by several different groups in different countries. From their responses, we now understand what must be done for Ω to be a realistic replacement for TEX.

First, Ω is too big! A typical run of Ω uses about 14MB, which is just fine when you are sitting in front of a 500MHz-machine with 512MB, but certainly not on a typical portable. This tremendous size comes from the TEX program structure, in which static arrays are allocated to handle primitives such as `\catcode` or `\delcode`, to store register values and font information, etc. However, the average user will never need 65,536 fonts of 65,536 characters each, nor 65,536 mu-registers, etc. Most of these huge tables are full of zeros, and it seems silly to have to go out and buy RAM and see your system slow down just so you can have lots of empty tables in your program.

This problem will be solved in the next version through the introduction of several primitives of the form `\MaxActiveCharacter`, `\MaxRegister`, `\MaxFont` or `\MaxWrite`. These primitives correspond to compile-time constants, which should be over-ridable upon loading Ω. By doing this, a single binary can be used, whatever the resources needed. According to Benjamin Bayart (École Supérieure d'Ingénieurs en Électrotechnique et Électronique in Paris), these primitives also make it possible for macro packages to determine whether the existing system has the required resources or whether a new run should be undertaken, with larger tables.

Second, Ω does not run correctly on Little Endian machines. This problem was solved by Bayart and will be incorporated into version 1.3. As a result, there should be working versions of Ω for Intel boxes running DOS, Windows and Linux.

Finally, performance is unsatisfactory for ΩTPs that are being used for multilingual applications. Because the macro-expansion facilities are applied with every use of an ΩTP, the use of two successive ΩTPs can slow Ω down so that it runs only 40% as fast as the original TEX. This may be acceptable for limited applications where specialized effects are wanted, but it is certainly unacceptable in a production setting where thousands of pages are being generated every day.

Support for complex scripts

It turns out, however, that for any given language, very few TEX primitives are required for typesetting high-quality output. As a result, we are looking for more efficient techniques that can be used for the standard cases.

In particular, Plaice has worked with ArborText, Inc. (Ann Arbor, Michigan), a supplier of SGML authoring and composition software and services, in developing typesetting support for ideogram scripts from East Asia. ArborText uses a TEX-based engine for printing SGML documents, and this engine was modified so that it could output Japanese text.

The fundamental understanding of a font in TEX is that each character has width, height, depth and italic correction. Characters are placed in order on the baseline, and the exact choice of character and the exact horizontal positioning can be adjusted using the ligature/kerning table.

This technique is reasonable for laying out alphabetic scripts where the characters are printed separately, as is the common case for Latin, Greek, Cyrillic, Armenian, Georgian, among others. Nevertheless, even for the Latin script, problems can arise: the Unicode standard provides for more than 900 precomposed characters. Building a complete ligature/kerning table for a font would require inordinate amounts of memory. Furthermore, it would be unlikely that, say, character `1EA9` (LATIN SMALL LETTER A WITH CIRCUMFLEX AND HOOK ABOVE), used in Vietnamese, would be found next to character `01CF` (LATIN CAPITAL LETTER D WITH SMALL LETTER Z WITH CARON), which is a Croatian digraph: much of the table would be useless. In fact, many of the characters have similar attributes: the difference betwen 'è' and 'é' is unlikely to influence the ligature/kerning program, so there are many opportunities for compressing it.

When we pass on to the ideogram scripts found in East Asia, all the characters have the same dimension. There are no ligatures, nor kerning. However, if a fixed grid is not chosen, then glue must be placed betweeen successive characters. Furthermore, line breaks cannot occur after left-bracket-like characters or before right-bracket-like characters. To handle such situations, penalties must be placed automatically in the appropriate places.

For vowelized Arabic, the requirements are different yet again. Not only must the correct presentation form — isolated, initial, medial or final — of each consonant be chosen, but the diacritics, including vowels and hamza, must be properly placed with respect to the consonants. To do this requires additional parameters about each character, designating the horizontal and vertical placement required to place the different diacritics. Finally, *keshidehs* (straight lines or Bézier curves) must be placed between consonants to fill out lines.

For the Arabic script in Nastaliq style, as is normally used for Farsi or Urdu, typesetting becomes even more complicated, since successive characters are not placed on the baseline. Rather, the characters within a word are placed in a sort of staircase situation. The first character, to the right, is placed highest. The lowest is the last character, to the left. Once again, extra character parameters are required so that the successive characters can be displaced vertically by the correct amount.

The work undertaken with ArborText implied designing another extension to the font metric files so that an arbitrary number of different kinds of parameters could be defined for the font as a whole or for each individual character. Currently, five sorts of parameters can be defined: integer, fixword, rule, glue and penalty. In addition, the ligature/kerning program has been modified to allow the automatic insertion of glue and penalties between characters, as is required for East Asian ideogram fonts. In addition to changes to the TEX driver, the `pltotf` and `tftopl` both had to be modified.

Under an agreement with ArborText, we will be incorporating these ideas into Ω. In fact, Ω will support a generalization of these ideas: the ligature/kerning table will allow two-dimensional capabilities, thereby solving all of the difficulties in typesetting calligraphic scripts such as Arabic.

Multiple directions

Multilingual typesetting requires printing in several directions. Scripts currently in use today can be categorized into four groups.

1. The most common group includes most of the world's scripts (Europe, Caucasus, India and South-East Asia). Lines are read from left to right and pages are read from top to bottom. When diacritics are used, in most cases, they are placed above characters in a line.

2. The next group includes several scripts that originate in West Asia, such as Arabic and Hebrew. Lines are read from right to left and pages are read from top to bottom. Diacritics

are placed both above and below characters in a line.

3. The third group includes the scripts found in East Asia. When the traditional scripts are used, lines are read from top to bottom and pages are read from right to left. It is standard for technical documents to be printed in the same manner as the first group. When the traditional means are used for printing and, say, English text is inserted, then it will be rotated 90° clockwise, and one can read the English insertion as if the text were in landscape mode.
4. The final group consists of Mongolian and other languages in the Uighur region of China and in Mongolia. In these languages, lines are read from top to bottom and pages are read from left to right. One way to perceive such texts is that they are similar to Arabic or Syriac, but rotated 90° counterclockwise.

It should be understood that the writing direction affects the entire page. An Arabic text will indent from the right, and successive entries in a table will be placed from right to left. Similarly, headers in traditional ideogram typesetting are found at the right-hand side of the page. Nevertheless, some aspects remain constant across the different groups. No matter what group is being typeset, mathematics will remain as if it were part of the first group.

In Ω, a writing direction is defined using two parameters: the *primary direction* corresponds to the direction in which successive lines follow each other on a page and the *secondary direction* corresponds to the direction in which successive characters follow each other on a line. The above four groups can be summarized as follows:

	primary	secondary
1.	top-down	left-right
2.	top-down	right-left
3.	right-left	top-down
4.	left-right	top-down

In TEX parlance, the primary direction corresponds to 'vertical' displacement and the secondary direction to 'horizontal' displacement. For example, an `\hspace` command in Mongolian would actually mean a downwards displacement on the page, along the baseline.

Different writing directions can interact on the same page, possibly even in the same paragraph, for quotations, insertion of mathematics, etc. The common interactions are 1–2, 1–3, 1–4 and 3–4. The first two have been the subject of *TUGboat* articles (Knuth and MacKay 1987, Hamano 1990), and Ω will use similar techniques to effect the proper changes.

The writing direction can even change when printing the same script. This was a technique used in boustropheidon, for ancient Greek texts: it combined the first two writing directions, alternating from line to line, with the characters mirrored as the text changed directions. In addition, ancient Egyptian texts would freely intermix the first two writing directions as well. We are looking at means to support these different techniques.

Conclusion

The coming releases of Ω, as outlined here, will ensure that Ω will truly become a multilingual successor to TEX.

References

[1] Hamano, Hisato, "Vertical typesetting with TEX", *TUGboat* 11,3 (1990), pages 346 – 352.

[2] Knuth, Donald, and Pierre Mackay, "Mixing right-to-left texts with left-to-right texts", *TUGboat* 8,1 (1987), pages 14 – 25.

[3] International Organization for Standardization/ International Electrotechnical Commission, "Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane", (ISO/IEC 10646-1), 1993.

[4] The Unicode Consortium, "The Unicode Standard: Worldwide Character Encoding", Version 1.0, Volume 1, Addison-Wesley, 1991.

StarTeX — A TeX for Beginners

Dag Langmyhr
Department of Informatics
University of Oslo
Norway
dag@ifi.uio.no

Abstract

This article describes StarTeX, a new TeX format for students writing their first report and other novice users. Its aim is to provide a simpler and more robust tool for users with no previous knowledge of TeX and LaTeX.

The problem

Students taking courses at our department are required to write short project reports, and LaTeX [2] has been the preferred tool. Several years experience has, however, shown us that LaTeX is not ideal for this.

This project report is the first encounter most students have with LaTeX, and they face many problems:

- The major problem is the error messages. They are very terse at best, and since they are sometimes produced by LaTeX and at other times by TeX, understanding the messages requires reasonably good knowledge of both systems. Most students tend to look only at the line numbers when examining their error logs.
- LaTeX is not very robust; trivial syntax errors can cause a serious burst of confusing error messages, like when you forget a `\\` prior to `\hline` in an `array` environment.

  You can also experience undesired effects if you use the commands incorrectly, for instance if you write

  ```
  \abstract{text}
  ```

  rather than the correct

  ```
  \begin{abstract}
    text
  \end{abstract}
  ```

  This error produces no error message, but will cause the whole article to be set in a smaller font.
- LaTeX does not hide the primitive commands of TeX, making it possible for the users to access them accidentally. For example, one of our users defined a macro for her name:

  ```
  \def \else {Else Hansen}
  ```

  This error alone produced more then 100 error messages.
- LaTeX uses ten special characters: `#`, `$`, `%`, `&`, `~`, `^`, `_`, `{`, `}` and `\`. Users need to remember that these characters are special, and they must learn which commands are necessary to produce them if they are required in the text. Fewer special characters would be an advantage.
- The command notation `\xxx` used in LaTeX often causes problems with the space following it.
- LaTeX has borrowed its error recovery philosophy from plain TeX: the user is expected to manually correct each detected error to allow LaTeX to proceed. The problem with this approach is that you will get many confusing error messages if you do not correct the error properly.

  None of our students use this interactive recovery facility; they either restart after having discovered the first error, or they let the processing run to completion without any interaction. An automatic error recovery scheme like that employed by compilers would be a great benefit for these users.
- LaTeX provides a mixture of structural markup commands as well as visual mark-up. The advantage is that experienced users can achieve the visual appearance they desire; the disadvantage is that less experienced users — particularly those who have used other document processing tools — spend too much of their time trying to coerce LaTeX into producing exactly the layout they think is proper.
- LaTeX is a large system and running it is not as fast as for instance plain TeX. For instance, a 2½ page sample document takes from 2.8 seconds on a Sun SparcStation20 to 8.7 seconds

on a Silicon Graphics Indy. Since novice users tend to process their documents very frequently to remove errors or test the effect of a feature, execution times do matter — even in this range.

The requirements

All these problems indicate that LATEX in its present form is not the tool we want for our students, at least not for their first report. We want a document processing program with the following properties:

- It must be based on TEX to achieve the desired quality in mathematical formulae.
- It should use a different notation for its mark-up commands; one which caused less confusion concerning spaces and has fewer special characters.
- It must hide all the internal TEX commands; this is the only safe way to avoid students using them accidentally.
- It must be small and easy to understand, so that it may easily be adapted to the particular need of each installations.
- It should contain structural mark-up commands only, and no visual mark-up.
- It should be robust.
- It should produce better error messages. If possible, no messages from TEX should ever appear. If this is impossible, error messages from TEX should be preceded by a message produced by the new tool.
- Since most students tend to just disregard all messages about under- and over-full boxes, it should try to reduce the number of such messages.
- It should run in nonstop mode and use automatic error recovery to detect as many genuine errors as possible.
- It should be as fast as plain TEX.
- The command handling should be insensitive to uppercase and lowercase. This is not an important issue, but case confusion has caused problems for some.

The solution

Attempting to achieve the goals mentioned above, StarTEX was designed. The name was chosen to indicate that it was a *Starters' TEX*.

StarTEX is a new format, and is thus a simple cousin of AMS-TEX [5] and LATEX. It is built on top of the plain TEX [1] commands.

The notation

At the EuroTEX conference in Arhem in September last year, Philip Taylor [6] proposed a different notation for (LA)TEX commands:

`<xxx>` rather than `\xxx`.

I decided to use this notation in StarTEX as it solves many of our problems:

- Spaces following the command are no longer a problem. There is no need for special rules like "When a space comes after a control word, it is ignored by TEX." [1, p. 8].
- Only one special character is needed: `<`. The characters `#`, `$`, `%`, `&`, `~`, `^`, `_`, `{`, `}` and `\` can be defined to be just ordinary characters.
- The command name may contain almost any character, not just letters.
- The scheme is easy to implement: all that is required is to make `<` an active character, and let the corresponding command regard everything up to the following `>` as a parameter.
- Since all commands are called through this interface, it is easy to make all internal TEX commands invisible.
- It is easy to check whether the user command is defined, and provide suitable error recovery if it is not.
- It is easy to `\lowercase` the user command, thus making the command handling insensitive to case.
- This command notation is the same as in HTML [4] with which many students are familiar.

I could have used any bracketing symbol pair, like `[xxx]` or `{xxx}` or `/xxx\`, but I chose `<xxx>` because it resembles HTML and because `<` and `>` are not used very frequently.

Command parameters A few commands need a parameter to specify non-printing matter like a file name or a label. I chose to use square brackets for this, as in

`<ref>[`*label*`]`

Using a special notation indicates more clearly that the parameter is not to be typeset.

The command set

The set of available StarTEX commands was chosen with the following aims in mind:

- There should be sufficient commands for writing a student report, but otherwise there should be as few commands as possible.
- There should be no commands for visual mark-up, only structural specifications.

- The commands should have a form that makes them easy to check for errors, and to automatically recover from the errors.

In table 1 are listed most of the StarTEX commands with their LaTeX counterpart.

Paragraph separation It was decided to use `<p>` to separate paragraphs, as in HTML. Even though the blank line used by (La)TeX is easier to type, it does cause problems with indentation of the paragraph following an environment like a list. Using `<p>` alleviates this problem.

Another advantage of using the `<p>` notation is that it can be employed as line separator (like `\\` in LaTeX) in environments where the concept of paragraph makes little sense, as in the `<title>` or `<author>` environments. This provides a double benefit: a special command for line breaking is no longer necessary, and using `<p>` in a `<title>` environment is now legal.

Font selection A few commands for font selection are necessary, but my belief is that `<b>` (for **bold text**), `<i>` (for *italic*) and `<tt>` (for `typewriter text`) form a sufficient set of commands. The commands may of course be nested to provide for instance *`italic typewriter text`*.

Some might argue that these commands are visual rather than structural, and that the HTML approach of providing a wider selection of structural commands like `<dfn>` for definitions, `<em>` for emphasis, `<kbd>` for keyboard input and `<samp>` for literal characters, is more logical. My own experience is that there are seldom enough definitions to suit my needs, so I will for instance use a specification like `<strong>` when I really want to indicate a reserved word in a programming language. Providing a few simple type changing commands is simpler.

PostScript figures Since nearly all figures used in LaTeX documents at our department are PostScript files, it seems reasonable to specialize the interface for this. The notation

`<psfig>[`*file name*`]` *caption text*`</psfig>`

was chosen as only two keywords were necessary. All figures are automatically scaled and they float to the top of the current or following page.

Tables The notation for tables was also chosen to be as simple as possible, and to ease error detection and recovery. Only very regular tables are catered for, but this is the price one has to pay for a simple notation.

Table 2: A small table sample

Index	**Data**
12	199
17	0

A table is a complex structure, with entries in columns within rows inside the table, but a notation was found which will seldom give grouping errors:

```
<table>caption text
<row>text<col>text<col>...
<row>text<col>text<col>...
     :
</table>
```

Every `<row>` starts a new row, and each `<col>` starts another column. The text prior to the first row is regarded as the table caption.

The number of columns is determined automatically. All columns are centered, and a grid of horizontal and vertical rules is always added. For example, the code

```
<table>
  A small table sample
  <row> <b>Index</b> <col> <b>Data</b>
  <row> 12 <col> 199
  <row> 17 <col> 0
</table>
```

will generate the table shown as table 2.

Document styles All documents need some adaption to conform to a particular style. I propose to let the user decide this by stating

`<style>[`*style file*`]`

The style file is written in plain TeX and contains the necessary definitions and modifications. Since the user has no visual mark-up commands at his or her disposal, all design decisions are made by the style designer. This makes it easier to have all reports conform to the approved standard.

My hope is that each site using StarTEX will develop styles of their own. These styles should be comprehensive, so the user should only have to specify that one style. For instance, our style `ifi-report` defines

- the page size (A4 paper),
- Norwegian format of `<today>` and `<now>`,
- Norwegian translations of fixed texts like “Figure” and “Table”,
- the page headers and footers, and
- various minor typographic details.

Cross references StarTEX uses more or less the same mechanisms for cross references as LaTeX. Interesting sections, figures and tables are given a label using the `<label>` command, which may then be referenced using the `<ref>` command.

	StarTEX	**LATEX**
Document bounds	<body>*text*</body>	\begin{document}*text*\end{document}
Document style	\style[*style file*]	\documentclass{*style file*}

Document
head
	<title>*text*</title>
	<author>*text*</author>
	<info>*text*</info>
		\title{*text*}
		\author{*text*}
		\date{*text*}

Font
change
	<b>*text*</b>
	<i>*text*</i>
	<tt>*text*</tt>
		\textbf{*text*}
		\textit{*text*}
		\texttt{*text*}

Paragraph break	<p>	⟨blank line⟩
Mathematical formula	<math>*formula*</math> <displaymath>*formula*</displaymath>	\(*formula*\) \[*formula*\]

Sectioning	<h1>*text*</h1>
	<h2>*text*</h2>
	<h3>*text*</h3>
	<h4>*text*</h4>
		\section{*text*}
		\subsection{*text*}
		\subsubsection{*text*}
		\paragraph{*text*}

Itemized
list
	<list>
	<item> ...
	:
	</list>
		\begin{itemize}
		\item ...
		:
		\end{itemize}

Enumerated
list
	<list>
	<numitem> ...
	:
	</list>
		\begin{enumerate}
		\item ...
		:
		\end{enumerate}

Description
list
	<list>
	<textitem>*text*</textitem> ...
	:
	</list>
		\begin{description}
		\item[*text*] ...
		:
		\end{description}

PostScript
figure
	<psfig>[*file name*] *caption text*
	</psfig>
		\begin{figure}
		\caption{*caption text*}
		\begin{center}
		\epsfig{file=*file name*,...}
		\end{center}
		\end{figure}

Table	<table>*caption text*
	<row>*text*<col>*text*<col>...
	<row>*text*<col>*text*<col>...
	:
	</table>
		\begin{table}
		\caption{*caption text*}
		\begin{center}
		\begin{tabular}{|c|...}\hline
		text& *text*& ...\\ \hline
		text& *text*& ...\\ \hline
		:
		\end{tabular}
		\end{center}
		\end{table}

Footnote	<footnote>*text*</footnote>	\footnote{*text*}
Unformatted text	<code>*text*</code>	\begin{verbatim}*text*\end{verbatim}

Cross
references
	<label>[*label*]
	<ref>[*label*]
		\label{*label*}
		\ref{*label*}, \pageref{*label*}

Comments	<comment>*text*</comment>	%*text*⟨end-of-line⟩
User macro	<define><*name*>*definition*⟨end-of-line⟩	\newcommand{*name*}{*definition*}

Table 1: StarTEX command overview

Symbol	StarTeX code
$<$	`<lt>`
$>$	`<gt>`
–	`<-->`
—	`<--->`
⟨a tie⟩	`<~>`
...	`<...>`
⟨today's date⟩	`<today>`
⟨the present time⟩	`<now>`
TeX	`<tex>`
LaTeX	`<latex>`
StarTeX	`<startex>`

Table 3: The remaining StarTeX commands

The appearance of the reference is defined by the document style, but will normally contain the page number if the reference is a different page; there is thus no need for a `\pageref` command. (This is similar to the `varioref` package [3].

Mathematical formulae One of the most important reasons for choosing a typesetting system based on TeX is its ability to typeset mathematical formulae. All the math mode commands available in (LA)TeX are implemented in StarTeX, and most of them use a notation similar to HTML version 3.0. For example, the formula

$$\int_1^\infty \frac{f(x)}{1+x}\partial x$$

is typed as

```
<displaymath>
  <int><sub>1</sub><sup><infinity></sup>
    <frac>f(x)<over>1+x</frac>
    <partial>x
</displaymath>
```

User-defined macros It was decided to allow the users to define their own commands, but with the following restrictions:

- The macros may not have parameters.
- No macros may be redefined.

The StarTeX notation

> `<define>`<*name*>*definition*⟨end-of-line⟩

was chosen to make error recovery easier. There is now no chance of a runaway definition, like you would get in (LA)TeX if you forgot a final `}`.

Various other commands In table 3 are shown the few remaining StarTeX commands.

An example In figure 1 is shown an example document using some of the StarTeX commands.

Other design decisions

Error recovery As mentioned previously, StarTeX can employ the `<xxx>` notation to detect errors and provide some error recovery. For instance, it keeps track of both the current and the outer environments, and which commands should be used to exit those environments. This means that it can detect and remedy the following situations:

- A missing terminator `</xxx>` will be detected when the outer environment is finished. In this case, both environments will be exited, and you would get an error message like

```
** StarTeX error detected on line 7:
   <i> on line 7 terminated by </b>.
   An extra </i> has been inserted.
```

- A superfluous terminator `</xxx>` will be recognized as such, and ignored, and the user would be notified with the following error message:

```
** StarTeX error detected on line 15:
   <body> on line 1 terminated by </b>.
   The </b> will be ignored.
```

Paragraph parameters LaTeX is a program for quality typesetting, and this is reflected in the standard parameters for paragraph breaking. Even paragraphs that look quite good to an untrained eye may produce messages about under- or over-full boxes. When LaTeX is unable to find a set of breaks it regards as acceptable, the result may be truly horrible, with words sticking into the margin, or all excess space put into the first line. This occurs quite often in Norwegian which has many long compound words. An experienced LaTeX user will easily detect the problem word and fix that or rephrase the text, but novice users seldom understand these messages and tend to ignore them.

All the messages about over-full and under-full boxes create another problem for the LaTeX novices. Since many of them use tools (like AUC-TeX [7]) that run LaTeX in non-stop mode, they get pages and pages of serious error messages intertwined with innocuous warnings, so they tend to just ignore all the messages as long as the printed result looks acceptable to them.

StarTeX sets its standard parameters for very loose typesetting with high values for `\tolerance` and `\emergencystretch`. The reasons for this are:

- If a good set of paragraph breaks exists, TeX will still choose that.
- Since the users tend to ignore messages about bad breaks, it is better to have a loosely broken paragraph than the very bad result you may get when TeX has to give up.

```
<body>
<title> <startex><--->A <tex> for beginners </title>
<author> Dag Langmyhr<p> Department of Informatics<p>
  University of Oslo<p> <tt>dag@ifi.uio.no</tt>
</author>
<info> <today> </info>

<abstract> This document describes <startex>, a special <tex>
  format for students writing their first project report.
</abstract>

<h1> The basic philosophy of <Startex> </h1>
<Startex> was designed for novice <tex> users. It employs a
different notation and a different set of commands from <latex>,
and the idea is that this makes it more user-friendly for these
users than plain <tex> or <latex>.

<p>
The notation used in <startex> resembles HTML and some of the
commands are the same, but the philosophy of the two is
different. HTML was designed to display hypertext information
on a computer screen, while <startex> is used to produce a
student report on paper.
</body>
```

Figure 1: An example StarTEX document

- The results achieved this way are at least as good as those produced by other typesetting and text-processing software.

This solution does not solve the problem of obtaining good paragraph breaks, but experience so far has shown that it goes a long way.

Concluding remarks

StarTEX has been completed and is being introduced to the students the coming term. It has — in my opinion – achieved most of the specified goals, but not all.

- It is quite small, consisting of fewer than one thousand lines of TEX code plus documentation. Whether the code is easy to understand is for others to judge.
- It is moderately robust. Most simple errors are handled by StarTEX, but grave ones still confuse it.
- It is reasonably fast; the 2 1/2 page example document mentioned at the beginning of this article is processed in 0.9 and 1.6 seconds, respectively.

Even though the users are taught a different format with a different command syntax, I believe StarTEX will serve as a suitable introduction to LATEX and document processing, because it provides training in the *concepts* of LATEX and structural mark-up.

(An analogy from computer science: The programming language C is widely used, and most programmers should know it. It is, however, a language for experts, so a common view is that students should first learn the concepts of programming in a different language before being exposed to C.)

The invention of StarTEX is not intended as any kind of criticism against LATEX, which is still our main tool for larger documents and for the more experienced users. The aim of StarTEX is to help one specific group of users, and provide them with a gentler introduction into the world of (LA)TEX.

On the other hand, StarTEX can be regarded as a tribute to TEX which so easily allows one to produce a different user interface to its powerful mechanisms.

Why not use HTML? Some users have asked why we do not use HTML when the notation is so similar. There are several reasons for that:

- There is no yet final definition of HTML. There are several versions available, in addition to the inventions of various software companies. Nobody knows what HTML will look like a few years from now.

- HTML is growing very complex, with many constructs of little interest to the student writing a report.
- It is difficult to write a robust parser of HTML in TEX.

Availability If anyone is interested in obtaining a copy of StarTEX, they can find it available for anonymous FTP on ftp.ifi.uio.no in the directory pub/tex/startex.

References

[1] Knuth, Donald E, *The TEXbook*, Addison-Wesley, 1991.

[2] Lamport, Leslie, *LATEX user's guide and reference manual*, Addison-Wesley, 1994.

[3] Mittelbach, Frank, "The `varioref` package", Part of the LATEX 2$_\varepsilon$ distribution.

[4] Raggett, Dave, "HyperText markup language specification version 3.0 draft", Available at `http://www.w3.org/pub/WWW/MarkUp/html3/`.

[5] Spivak, Michael, *The joy of TEX*, American Mathematical Society, 1986. The guide to $\mathcal{AMS}$-TEX.

[6] Taylor, Philip, "TEX: an unsuitable language for document markup?", Talk given at the EuroTEX 1995 conference; does not appear in the proceedings.

[7] Thorup, Kresten Krab, "AUC TEX", An Emacs mode for editing (LA)TEX code; available from `http://www.iesd.auc.dk/\symbol{126}amanda/auctex/`.

Do Journals Honor LaTeX Submissions?

Gabriel Valiente Feruglio
Technical University of Catalonia
Departament of Software
E-08028 Barcelona, Catalonia, Spain
valiente@lsi.upc.es
URL: http://www-lsi.upc.es/\~valiente/

Introduction

This note addresses the survival of LaTeX in the academic world, and it does it from the perspective of electronic publishing of LaTeX articles in scientific journals. Such a perspective is necessarily limited, since survival of LaTeX in the academic world will undoubtely depend on a multitude of factors, often intertwined, but it is quite interesting in itself since it will provide further motivation for PhD students, young scientists, and teaching assistants to adopt LaTeX as an integral solution for their typesetting needs along their academic lives, from writing a PhD thesis to typesetting class notes, research articles, and textbooks.

In fact, the original motivation for writing down this note was to attract potential LaTeX users among PhD students by showing them still another benefit of adopting LaTeX for their typesetting tasks, namely that scientific journals accept and encourage electronic submission of LaTeX sources. Such was also the motivation behind the chapter on electronic publishing in the author's recent LaTeX textbook [12].

An extensive research over the Internet was then conducted in order to find all journals that accept electronic submission of LaTeX articles in source form. Despite many journals not even mentioning the possibility for TeX or LaTeX submissions, the research shows that LaTeX use has spread well beyond the traditional subject areas of computer science, mathematics and physics.

The section entitled "Dynamics of LaTeX submissions" gives an overview of the whole process of LaTeX article submission, processing, and publishing. The results of the research over the Internet are summarized in in the section entitled "Journals" and they are discussed in the section entitled "Discussion". As a direct consequence of that discussion, the creation of a Technical Working Group to support and coordinate publisher's efforts is proposed in the section entitled "Conclusions". The data resulting from the research over the Internet is presented in Appendix A.

Dynamics of LaTeX submissions

Submission of articles marked up with LaTeX may have different pros and cons for the people involved, from author and academic editor to reviewer and publisher. The whole process of submitting, processing and publishing a LaTeX journal submission is briefly reviewed in the following in order to put some of the issues involved in the right perspective:

1. The author writes a LaTeX article.
2. The author submits the article to one of the journal's academic editors.
3. The academic editor selects one or more reviewers and sends them the article.
4. The reviewers judge the article and advise the academic editor on acceptance.
5. The academic editor decides to accept the article, with or without changes, or to reject it.
6. On acceptance, the —probably revised— article is sent over to the publisher.
7. The publisher processes the article.
8. Although the author can obtain galley proofs (laser printer output), in some cases the publisher sends a page proof (phototypesetter output) to the author.
9. The publisher —usually a technical editor or a copy editor— applies final corrections to the article.
10. The article is included in a journal issue, either printed and/or electronic, and the issue is distributed.

Compared to traditional manuscript submission and processing, submission of LaTeX sources offers many advantages:

Faster delivery LaTeX sources can be sent by electronic mail or by ftp, a delivery method that is much faster than regular mail or even courier mail and much cheaper than the latter. This is an interesting issue, since an article is sent several times, at least three: author to academic editor, academic editor to each of the reviewers, and academic editor to publisher. It must be noticed, however, that editor and reviewers can still communicate by any means they choose about the review, including —but not limited to— further LaTeX sources[1], irrespective of whether the submission was a LaTeX source.

Reduced proof-reading Since there is no need of re-keying the submitted article from a paper copy, there is no real need for the publisher to send galley proofs to the author. No typing errors are (supposed to be) introduced in the article[2].

Shorter publication time Bypassing the typesetter and reducing or even eliminating proof-reading, production of page proofs is much faster and the overall cost of publication is reduced.

Reliability Whenever the publisher makes a LaTeX macro package available, the author can compile the article and obtain a preprint which is almost identical to the published article, perhaps differing only in page numbering and journal identification. Layout problems can be fixed by the author even before first submitting the article, contributing then to a further reduction in publication time and cost. The dark side of this issue is a burden on the author, who gets distracted from the article's content and becomes more of a copy editor.

Availability The author has an almost final version of the submitted article, which can be further distributed —usually in the form of a DVI or PostScript file— by electronic mail, `ftp`, the World-Wide Web (WWW), or a preprint archive [10]. This is indeed a highly controversial issue, since it affects the interests of the publisher, but as long as authors do not transfer copyright to publishers they are entitled to, say, put their articles in their WWW home pages. Some kind of balance will surely have to be found between author's interest in having their work as broadly disseminated as possible and publisher's economic interest which makes such a dissemination possible[3].

There are, however, some disadvantages to the submission and processing of LaTeX sources:

Processing burden Processing the LaTeX submission by academic editor and reviewers can be much of a burden on them. They need to assure that they get the complete submission, which often consists of several LaTeX source files and a set of EPS illustrations. The submission may fail to compile due to missing parts, required LaTeX macro packages not available at their installation, errors in included EPS figures, etc. It should not be overseen that most academic editors and almost all reviewers are not paid for their services.

Investment in learning Publishing staff and typesetters need to invest in learning TeX —which shows a steep learning curve— and in setting up and maintaining a whole TeX system, including high-resolution output devices and their drivers, integration of text and images, etc.

Some of these issues may explain why many journals accept and process LaTeX submissions but in most cases the academic editors prefer paper submissions; see Discussion below.

Journals

Finding out those journals that accept electronic submission of articles marked up with LaTeX would have not been possible if publishers did not offer journal information on the Internet. As a matter of fact, most publishers already maintain home pages for their journals on the World-Wide Web, and in many cases these pages offer extensive information for authors.

The following list gives the number of journals found within each scientific field that accept LaTeX

[1] In the case of the *Rewriting Techniques and Applications* conferences, for instance, review reports are standard LaTeX document templates which the conference organizers send to the reviewers, who fill them in and send back to the organizers, who then send over to the authors, and the whole process takes place over electronic mail.

[2] During the review of the book "On Being a Machine, Vol. 1: Formal Aspects of Artificial Intelligence," by A. Narayanan (Ellis Horwood, 1988) I had found over 300 typographic mistakes which the author attributed to the publisher's re-keying of the submission. A. Narayanan moved then to LaTeX and provided Ellis Horwood with camera-ready copies for the second volume, "On Being a Machine, Vol. 2: Philosophy of Artificial Intelligence" (Ellis Horwood, 1990). The review appeared in *Artificial Intelligence* 12(4):96–97, 1991.

[3] A first step in this direction has been taken recently by Elsevier Science for the *Electronic Notes in Theoretical Computer Science* series of Conference Proceedings, whereby authors are forbidden to make their contributions available by anonymous `ftp` or ever the WWW but are allowed instead to include links from their WWW pages to Elsevier Science's own WWW pages, where full access to articles is only granted to people accessing from an institution which holds a subscription to the *Theoretical Computer Science* journal.

submissions, according to the *Science and Engineering Field Classification* made by the National Science Foundation. The classification scheme is available at `http://www.qrc.com/nsf/srs/rdexp/`.

• Computer Sciences	97
• Mathematical Sciences	89
• Engineering	77
• Physical Sciences	61
• Life Sciences	51
• Environmental Sciences	18
• Social Sciences	18
• Other Sciences (Multidisciplinary)	7
Total	418

As can be seen from the previous list, adoption of LaTeX in scientific publishing has spread well beyond the traditional subject areas of computer science, mathematics and physics. Notice, however, that for each journal accepting submissions of articles marked up with LaTeX there may be up to ten journals in the same field which do not accept LaTeX submissions.

Discussion

Some of the issues behind the situation described in the section entitled "Journals" are depicted in the following in the form of short *provocative statements*, which are not meant to be definitive assertions but to rather spark further debate within the TeX community about the future of LaTeX in the academic world.

Publishers regret to accept LaTeX submissions because it doesn't pay off Let alone publishers who have never heard about LaTeX, even for those who care about LaTeX keeping up with LaTeX developments may represent too big an overhead. Take for instance Springer Verlag, who has even replaced its well-known `llncs` macro package by the LaTeX 2.09 formats (NFSS version 1) *CLMono01* and *CLMult01*.

As a matter of fact, the proof is that almost two years after the first release of LaTeX 2_ε, relatively few scientific publishers have updated their LaTeX macro packages to LaTeX 2_ε.

Moreover, many publishers argue that setting up a TeX system, keeping it up-to-date, and polishing LaTeX submissions to match their house styles is usually more expensive and time-consuming than rekeying the submitted articles from author-supplied hard copies.

Publishers do not get articles marked up with LaTeX for publication One of the reasons why most publishers in the fields of environmental, life, and social sciences do not honor LaTeX submissions is that they rarely get articles marked up with LaTeX for publication. As a matter of fact, authors seem to be the driving force behind the adoption of LaTeX by scientific publishers.

Publishers force authors to submit *standard* LaTeX articles Publishers complain that it is almost impossible to have authors submit articles marked up with *standard* LaTeX, that is, without author-defined macros, while authors complain that publishers limit their creativity by forcing them to comply with some LaTeX macro package [7]. Maybe both sides are right in their complaints, but the truth is that publishers have a good deal of work at polishing LaTeX submissions and resolving macro name clashes, while it is both unreasonable and contrary to LaTeX's philosophy to forbid authors defining new macros in their articles.

A solution to both sides of the problem can be foreseen in the form of either an extension to the LaTeX kernel, a macro package or some kind of utility program, which would expand all author-defined macros and output a *standard* LaTeX article source. The question is, what exactly is a *standard* LaTeX article source?

Journals honor LaTeX submissions but academic editors do not Although many publishers have all the hardware, software and know-how needed to process LaTeX submissions, however, academic editors for each of the journals they publish always have the last word.

Take, for instance, some of the major scientific publishers which are moving into electronic publication [11]. Elsevier Science accepts, in principle, LaTeX submissions for all of its 1100 journals but academic editors for only 7 of them are willing to accept LaTeX submissions.

A similar pattern is repeated for other publishers. Academic editors at Springer Verlag only accept LaTeX submissions for 8 of its 350 journals, at John Wiley & Sons only 9 out of 326 journals do, at Blackwell Science only one out of 200 journals does, and at Academic Press only two out of 175 journals accept LaTeX submissions.

The question is, why do most academic editors desencourage submission of articles marked up with LaTeX, even though publishers provide them with running TeX systems and house styles already encoded in LaTeX macro packages?

Journals may no longer honor LaTeX submissions as they move electronic Electronic journals, as well as preprint databases [10], accept any

ASCII submission but in most cases prefer TeX or LaTeX, at least in the fields of engineering and computer, mathematical, and physical sciences. When it comes to environmental, life, and social sciences, however, it is much more common to find journals which only accept either RTF or HTML submissions.

LaTeX to HTML conversion may be seen as a practical solution. `LaTeX2HTML` [3] even allows the inclusion of hypertext links in articles. In practice, however, it may sacrifice typographical quality, since all mathematical formulas, figures and tables are converted to GIF (Graphics Interchange Format) images or PostScript pictures, which in most cases have a low resolution and cannot be zoomed in and out without distorting the image.

In addition, `LaTeX2HTML` fragments a well-structured LaTeX document into too many little files. Although the degree of splitting can be controlled by a parameter, it is set to a high value by default and, in practice, this turns reading the document with an HTML browser into a kind of...

As HTML develops into HTML3, with some degree of support for mathematics and tables, it is possible that HTML takes over as the preferred format for submission to electronic journals in the fields of engineering and computer, mathematical, and physical sciences as well.

Conversion of TeX and LaTeX into SGML [8, 1] may help to avoid HTML ever displacing LaTeX as one of the preferred formats for submitting articles to scientific journals, since the scientific publishing industry seems to be moving definitely towards SGML.

Conclusion

An author may have to deal with many publishers, and therefore may need to comply with different TeX macro packages and instructions to authors. Adoption of LaTeX by an author may prove to be, in that sense, a rewarding decision as long as publishers encode their house styles in LaTeX macro packages. This would let authors concentrate on scientific content while keeping LaTeX training needs down to a point somewhere between [6] and [4].

An ideal situation would be for the author to write a standard `article`-class LaTeX document and to later add a

`\usepackage{`*publisher*`}`

mark, or even better a

`\usepackage[`*journal*`]{`*publisher*`}`

mark, right before submitting it to the publisher.

In practice, however, complying with the author instructions for a particular journal may involve various changes to the original LaTeX source, ranging from low-level font selection to high-level macros for theorem-like environments, inclusion of encapsulated PostScript figures, and author affiliation.

Such a high degree of transparency of publisher styles with respect to the standard LaTeX `article`-class can only be reached by a serious standardization effort. Maybe the time has come for the TeX Users Group to set up a new Technical Working Group (TWG), with the goal of coordinating publishers' efforts at encoding their journal styles in LaTeX macro packages. Such a TWG should also liason with the LaTeX3 Project Team in order to enhance the standard LaTeX `article.cls` document class and perhaps also `book.cls` and `report.cls`, by including more structural information in the front matter which would offer a standard interface to authors and could also be easily adapted to the particular needs of different publishers. As a matter of fact, some publisher packages that show the need for such an enhancement have been available for several years, among which Springer [9], Elsevier Science [2], DANTE [5], and many others.

In any case, the author sincerely hopes not to be charged with the whole task just because of having had such a bright idea.

Acknowledgement

I am very grateful to Barbara Beeton, Sebastian Rahtz and Christina Thiele for early comments on the very idea of this paper, and to the anonymous referees, whose suggestions have led to a substantial improvement of the article.

References

[1] Anne Brüggemann-Klein. Wissenschaftliches publizieren im umbruch. *Informatik—Forschung und Entwicklung*, 10:171–179, 1995.

[2] Elsevier. *Preparing Articles with LaTeX: Instructions to Authors for preparing Compuscripts*. Electronic document available at `http://www.tex.ac.uk/tex-archive/macros/latex/contrib/supported/elsevier/`, 1995.

[3] Michel Goossens and Janne Saarela. TeX to HTML and back. *TUGboat*, 16(2):174–214, 1995.

[4] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 2nd edition, 1994.

[5] Gerd Neugebauer. Eine klasse für die TeXnische komödie. *Die TeXnische Komödie*,

4/95:6–15, 1996.

[6] Tobias Oetiker, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *The not so short Introduction to LaTeX 2ε*. Electronic document available at `http://www.tex.ac.uk/tex-archive/info/lshort/`, 1995.

[7] Nico Poppelier. Two sides of the fence. *TUGboat*, 12(3):353–358, 1991.

[8] Sebastian Rahtz. Another look at LaTeX to SGML conversion. *TUGboat*, 16(3):315–324, 1995.

[9] Springer-Verlag. *Instructions for Authors using LaTeX and the Springer Macro Package CLMono01 or CLMult01.* Electronic document available at `ftp://trick.ntp.springer.de/pub/tex/latex/clmomu01/`, 1995.

[10] Gary Taubes. Electronic preprints point the way to author empowerment. *Science*, 271(5250):767, February 1996.

[11] Gary Taubes. Science journals go wired. *Science*, 271(5250):764, February 1996.

[12] Gabriel Valiente. *Composició de textos científics amb LaTeX*. Edicions UPC, Barcelona, 1996.

A Journals accepting manuscripts marked up with LaTeX

This appendix lists journals for which at least one of the editors accepts electronic submissions written using LaTeX, grouped by publisher. An HTML version of this list is available on the Internet at the address `http://www-lsi.upc.es/\~{}valiente/journals.html` that links about 40 publishers and more than 400 journals to their home pages on the World-Wide Web. Any help to make it more complete and to keep it up-to-date is warmly welcome.

Academia Scientiarum Fennica

- Annales Academiæ Scientiarum Fennicæ

Academic Press

- Analytical Biochemistry
- J. of Approximation Theory

American Astronomical Society

- Astrophysical J.
- Astrophysical J. Supplement
- Astrophysical J. Letters
- Astronomical J.

American Institute of Physics

- The J. of the Acoustical Society of America

American Mathematical Society

- Bulletin of the $\mathcal{AMS}$
- Electronic Research Announcements of the $\mathcal{AMS}$
- J. of the $\mathcal{AMS}$
- Mathematics of Computation
- Notices of the $\mathcal{AMS}$
- Proc. of the $\mathcal{AMS}$
- Trans. of the $\mathcal{AMS}$

American Physical Society

- Physical Review A
- Physical Review B
- Physical Review C
- Physical Review D
- Physical Review E
- Physical Review Letters
- Reviews of Modern Physics

Association for Computing Machinery

- ACM Trans. on Mathematical Software
- Comm. of the ACM
- J. of the ACM
- IEEE/ACM Trans. on Networking
- J. of Experimental Algorithmics
- Trans. on Computer Systems
- Trans. on Computer-Human Interaction
- Trans. on Design Automation of Electronic Systems
- Trans. on Graphics
- Trans. on Information Systems
- Trans. on Mathematical Software
- Trans. on Modeling and Computer Simulation
- Trans. on Prog. Languages and Systems

Birkhäuser Verlag

- Aequationes Mathematicae
- Algebra Universalis
- Aquatic Sciences
- Archiv der Mathematik
- Botanica Helvetica
- Chemoecology
- Circuits, Systems, and Signal Processing
- Commentarii Mathematici Helvetici
- Computational and Applied Mathematics
- Computational Complexity
- Eclogae Geologicae Helvetiae
- Elemente der Mathematik
- EXPERIENTIA
- Fresenius Environmental Bulletin
- Geometric and Functional Analysis
- Helvetica Physica Acta
- Inflammation Research
- Insectes Sociaux
- Integral Equations and Operator Theory
- J. of Evolutionary Biology
- J. of Geometry
- J. of Mathematical Systems, Estimation, and Control
- MapleTech
- Medical Microbiology Letters
- Medicine
- Nonlinear Differential Equations and Applications
- NTM
- Pure and Applied Geophysics
- Resultate der Mathematik
- Selecta Mathematica, New Series
- Sozial- und Präventivmedizin
- Zeitschrift für angewandte Mathematik und Physik

Blackwell Publishers

- Computer Graphics Forum

Cameron University, Oklahoma

- Southwest J. of Pure and Applied Mathematics

Chapman & Hall

- Optical and Quantum Electronics

Computer Society of South Africa

- The South African Computer J.

Deutsche Mathematiker-Vereinigung

- Documenta Mathematica

DANTE

- Die TEXnische Komödie

Elsevier Science

- Artificial Intelligence
- Discrete Applied Mathematics
- Discrete Mathematics
- Electronic Notes in Theoretical Computer Science
- Linear Algebra and its Applications
- New Astronomy
- Theoretical Computer Science

Heldermann Verlag Berlin

- Beiträge zur Algebra und Geometrie
- J. of Lie Theory

Institute of Electrical and Electronics Engineers

- Computer
- IEEE Annals of the History of Computing
- IEEE Computational Science & Engineering
- IEEE Computer Graphics and Applications
- IEEE Design & Test of Computers
- IEEE Electron Device Letters
- IEEE Expert
- IEEE J. on Selected Areas in Communications
- IEEE J. on Selected Topics in Quantum Electronics
- IEEE J. of Microelectromechanical Systems
- IEEE J. of Quantum Electronics
- IEEE J. of Solid-State Circuits
- IEEE Micro
- IEEE Microwave and Guided Wave Letters
- IEEE MultiMedia
- IEEE Parallel & Distributed Technology
- IEEE Photonics Technology Letters
- IEEE Signal Processing Letters
- IEEE Software
- IEEE Trans. on Antennas and Propagation
- IEEE Trans. on Applied Superconductivity
- IEEE Trans. on Automatic Control
- IEEE Trans. on Biomedical Engineering
- IEEE Trans. on Circuits and Systems for Video Technology
- IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications
- IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing
- IEEE Trans. on Communications
- IEEE Trans. on Components, Packaging, and Manufacturing Technology Part A
- IEEE Trans. on Components, Packaging, and Manufacturing Technology Part B
- IEEE Trans. on Components, Packaging, and Manufacturing Technology Part C
- IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems
- IEEE Trans. on Computers
- IEEE Trans. on Control Systems Technology
- IEEE Trans. on Education
- IEEE Trans. on Electromagnetic Compatability
- IEEE Trans. on Electron Devices
- IEEE Trans. on Engineering Management
- IEEE Trans. on Fuzzy Systems
- IEEE Trans. on Geoscience and Remote Sensing
- IEEE Trans. on Image Processing
- IEEE Trans. on Industrial Electronics
- IEEE Trans. on Industry Applications
- IEEE Trans. on Information Theory
- IEEE Trans. on Instrumentation and Measurement
- IEEE Trans. on Knowledge & Data Engineering
- IEEE Trans. on Magnetics
- IEEE Trans. on Medical Imaging
- IEEE Trans. on Mechatronics
- IEEE Trans. on Microwave Theory and Techniques
- IEEE Trans. on Neural Networks
- IEEE Trans. on Nuclear Science
- IEEE Trans. on Oceanic Engineering
- IEEE Trans. on Parallel & Distributed Systems
- IEEE Trans. on Pattern Analysis & Machine Intelligence
- IEEE Trans. on Plasma Science
- IEEE Trans. on Power Electronics
- IEEE Trans. on Professional Communication
- IEEE Trans. on Rehabilitation Engineering
- IEEE Trans. on Robotics and Automation
- IEEE Trans. on Semiconductor Manufacturing
- IEEE Trans. on Signal Processing
- IEEE Trans. on Software Engineering
- IEEE Trans. on Speech and Audio Processing
- IEEE Trans. on Systems, Man, and Cybernetics Part A: Systems and Humans
- IEEE Trans. on Systems, Man, and Cybernetics Part B: Cybernetics
- IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control
- IEEE Trans. on Very Large Scale Integration (VLSI) Systems
- IEEE Trans. on Visualization & Computer Graphics
- IEEE Trans. on VLSI Systems
- IEEE/ACM Trans. on Networking
- IEEE/OSA J. of Lightwave Technology
- Proc. of the IEEE

Institute of Physics Publishing

- Bioimaging
- Classical and Quantum Gravity
- Distributed Systems Engineering
- European J. of Physics
- High Performance Polymers
- Inverse Problems
- J. of Micromechanics and Microengineering
- J. of Physics A: Mathematical and General
- J. of Physics B: Atomic, Molecular and Optical Physics
- J. of Physics: Condensed Matter
- J. of Physics D: Applied Physics
- J. of Physics G: Nuclear and Particle Physics
- J. of Radiological Protection
- Measurement Science and Technology
- Modelling and Simulation in Materials Science and Engineering
- Nanotechnology
- Network: Computation in Neural Systems
- Nonlinearity
- Physics Education
- Physics in Medicine and Biology
- Physiological Measurement
- Plasma Physics and Controlled Fusion
- Plasma Sources Science and Technology
- Public Understanding of Science
- Pure and Applied Optics
- Quantum and Semiclassical Optics
- Reports on Progress in Physics
- Semiconductor Science and Technology
- Smart Materials and Structures
- Superconductor Science and Technology
- Waves in Random Media

IOS Press

- AI Communications
- Asymptotic Analysis
- BioFactors
- Bio-Medical Materials and Engineering
- Chinese Science Bulletin (Kexue Tongbao)
- Education for Information
- Environmental Policy and Law
- Fundamenta Informaticæ
- Human Systems Management
- Information and Systems Engineering
- Information Infrastructure and Policy
- Information Services and Use
- Information Technology for Development
- Int. J. of Applied Electromagnetics and Mechanics
- Int. J. of Risk and Safety in Medicine
- J. of Computer Security
- J. of Economic and Social Measurement
- J. of Environmental Sciences
- J. of High Speed Networks
- Pharmacotherapy
- Reviews in Toxicology
- Space Communications
- Spectroscopy: An Int. J.
- Statistical J. of the United Nations Economic Commission for Europe
- Technology and Health Care

Kent State University

- Electronic Trans. on Numerical Analysis

Kluwer Academic Publishers

- Acta Applicandae Mathematicae
- Adsorption
- Analog Integrated Circuits and Signal Processing
- Applied Cardiopulmonary Pathophysiology
- Applied Categorical Structures
- Applied Composite Materials
- Applied Intelligence
- Applied Scientific Research
- Aquatic Geochemistry
- Archives of Suicide Research
- Astrophysics and Space Science
- Automated Software Engineering
- Autonomous Robots
- Biodegradation
- Biogeochemistry
- Bioseparation
- Biotherapy
- Boundary-Layer Meteorology
- Celestial Mechanics and Dynamical Astronomy
- Climatic Change
- Compositio Mathematica
- Computational Economics
- Computational Optimization and Applications
- Computers and the Humanities
- Crime, Law and Social Change
- Cytotechnology
- Design Automation for Embedded Systems
- Designs, Codes and Cryptography
- Discrete Event Dynamic Systems
- Distributed and Parallel Databases
- Documenta Ophthalmologica
- Dynamics and Control
- Earth, Moon and Planets
- Economics of Planning
- Educational Studies in Mathematics
- Empirica
- Entomologia Experimentalis et Applicata
- Environmental Monitoring and Assessment
- Euphytica
- European J. of Health Law
- European J. of Population
- Experimental Astronomy
- Financial Engineering and the Japanese Markets
- Formal Methods in System Design
- Gazette
- Genetic Resources and Crop Evolution
- Genetica
- Geology and Mining (Geologie en Mijnbouw)
- Geometriae Dedicata
- Geriatric Nephrology and Urology
- Hydrobiologia
- Instructional Science
- Interface Science
- Int. J. of Clinical Monitoring and Computing

- Int. J. of Computer Vision
- Int. J. of Fracture
- Int. J. of General and Molecular Microbiology
- Int. J. of Salt Lake Research
- Int. J. of Value-Based Management
- Int. J. on Group Rights
- Int. Ophthalmology
- J. for General Philosophy of Science
- J. of Algebraic Combinatorics
- J. of Applied Phycology
- J. of Aquatic Ecosystem Health
- J. of Atmospheric Chemistry
- J. of Automated Reasoning
- J. of Biological Physics
- J. of Elasticity
- J. of Electronic Testing
- J. of Engineering Mathematics
- J. of Global Optimization
- J. of Inclusion Phenomena and Molecular Recognition in Chemistry
- J. of Intelligent Information Systems
- J. of Logic, Language and Information
- J. of Mathematical Imaging and Vision
- J. of Paleolimnology
- J. of Sol-Gel Science and Technology
- J. of Systems Integration
- K-Theory
- Letters in Mathematical Physics
- Lifetime Data Analysis
- LISP and Symbolic Computation
- Machine Learning
- Machine Translation
- Man and World
- Meccanica
- Medical Progress Through Technology
- Molecular Biology Reports
- Multidimensional Systems and Signal Processing
- Multimedia Tools and Applications
- Mycopathologia
- Natural Hazards
- New Forests
- Nonlinear Dynamics
- Nutrient Cycling in Agroecosystems
- Origins of Life and Evolution of the Biosphere
- Philosophical Studies
- Photosynthesis Research
- Plant and Soil
- Plant Cell, Tissue and Organ Culture
- Plant Growth Regulation
- Potential Analysis
- Real-Time Systems
- Review of Industrial Organization
- Set-Valued Analysis
- Social Indicators Research
- Solar Physics
- Studies in East European Thought
- Surveys in Geophysics
- Systematic Parasitology
- The EDI Law Review
- The J. of Supercomputing
- The J. of VLSI Signal Processing
- The Int. J. of Cardiac Imaging
- Transport in Porous Media
- User Modeling and User-Adapted Interaction
- Vegetatio
- Water Resources Management
- Water, Air and Soil Pollution

Masaryk University, Czech Republic

- Archivum Mathematikum

Morgan Kaufmann

- J. of Artificial Intelligence Research

Optical Society of America

- Applied Optics
- J. of the Optical Society of America A
- J. of the Optical Society of America B
- Optics Letters
- J. of Lightwave Technology
- Chinese J. of Lasers B
- J. of Optical Technology
- Optics & Spectroscopy

Oxford University Press

- The Computer J.

Royal Astronomical Society

- Monthly Notices of the Royal Astronomical Society

Sociedad Colombiana de Matemáticas

- Revista Colombiana de Matemáticas

Societat Catalana de Matemàtiques

- SCM/Notícies

Société de Mathématiques Appliquées et Industrielles

- ESAIM: Control, Optimisation and Calculus of Variations
- ESAIM: Probability and Statistics
- ESAIM: Proc.

Society for Industrial and Applied Mathematics

- SIAM J. on Applied Mathematics
- SIAM J. on Computing
- SIAM J. on Control and Optimization
- SIAM J. on Discrete Mathematics
- SIAM J. on Mathematical Analysis
- SIAM J. on Matrix Analysis and Applications
- SIAM J. on Numerical Analysis
- SIAM J. on Optimization
- SIAM J. on Scientific Computing
- SIAM Review

Springer Verlag

- Constructive Approximation
- Few-Body Systems Electronic
- Informatik—Forschung und Entwicklung
- J. of Nonlinear Science
- J. of Universal Computer Science
- J. of Very Large Databases
- Numerische Mathematik Electronic Edition
- Semigroup Forum

TEX Users Group

- TEX and TUG News
- TUGboat

The International Linear Algebra Society

- Electronic J. of Linear Algebra

Universidad Nacional Autónoma de México

- Revista Electrónica del Departamento de Matemáticas

University at Albany, State University of New York

- New York J. of Mathematics

Univerzita Komenskeho, Bratislava

- Acta Mathematica Universitatis Comenianæ

The Johns Hopkins University Press

- American J. of Mathematics

The MIT Press

- Artificial Life
- Computational Linguistics
- Evolutionary Computation
- J. of Functional and Logic Prog.
- Neural Computation
- The Chicago J. of Theoretical Computer Science
- The Int. J. of Robotics Research

John Wiley & Sons

- Comm. in Numerical Methods in Engineering
- Electronic Publishing: Origination, Dissemination and Design
- J. of Combinatorial Designs
- J. of Graph Theory
- Int. J. for Numerical Methods in Engineering
- Naval Research Logistics
- Numerical Methods for Partial Differential Equations
- Random Strucrures and Algorithms
- Theory and Practice of Object Systems

World Scientific

- Int. J. of Cooperative Information Systems
- Int. J. of Foundations of Computer Science
- Int. J. of High Speed Computing
- Int. J. of High Speed Electronics and Systems
- Int. J. of Information Technology
- Int. J. of Modern Physics A: High Energy Physics
- Int. J. of Modern Physics B: Condensed Matter Physics
- Int. J. of Modern Physics C: Computational Physics
- Int. J. of Modern Physics D: Astrophysics
- Int. J. of Modern Physics E: Nuclear Physics
- Int. J. of Reliability, Quality and Safety Engineering
- Int. J. of Shape Modeling
- Int. J. of Software Engineering and Knowledge Engineering
- Int. J. of Uncertainty, Fuzziness and Knowledge-Based Systems
- Int. J. on Artificial Intelligence Tools
- J. of Circuits, Systems and Computers
- J. of Computational Acoustics
- J. of Knot Theory and its Ramifications
- Mathematical Models and Methods in Applied Sciences
- Modern Physics Letters A: High Energy Physics
- Modern Physics Letters B: Condensed Matter Physics
- Parallel Processing Letters

Other

- BIT
- Electronic J. of Combinatorics
- Electronic J. of Differential Equations
- Electronic J. of Probability
- Electronic Comm. in Probability
- Theory and Applications of Categories
- Reliable Computing

A New Approach to the TEX-related Programs: A User-friendly Interface

Sergei V. Znamenskii and Denis E. Leinartas
Department of Mathematics Krasnoyarsk State University, Svobodnyi prospekt 79, 660041 Krasnoyarsk, Russia
znamensk@math.kgu.krasnoyarsk.su, den@math.kgu.krasnoyarsk.su

Abstract

Various TEX-related programs such as TEX, METAFONT, BM2FONT, DVISCR, DVIPS, BIBTEX, and others have to use sophisticated command line options and configuration files to work cooperatively. Modifying the configuration of such a complicated system becomes a problem. A good solution is a hypertext-based language based on an intellegent shell which provides an easy interface for TEX-users.

The number of various configuration options to use in a command line or in environment to start a TEX-related program has become somewhat enormous. The appropriate choice of an option set to use depends in a rather sophisticated way on hardware configuration, desirable interface and a lot of installed software-specific features.

Therefore the proper modification of a TEX system configuration becomes a problem to the ordinary user even if he uses some shell to facilitate the work. There exists a wide variety of TEX shells and different users have different opinions on the subject of which shell is better. The TEXShell V2.7.1 by von Jörgen Schlegelmilch (available on CTAN) is currently the most popular non-commercial shell in Russia to run under DOS. It provides a convenient user interface. For example, the user can change printers by just selecting appropriate item in the menu system; change any command line options; save changed configurations completely or partially; use various configurations for different directories; to run TEX, preview or print a file; and other actions – by just hitting a 'hot' key from the internal editor. The user may find and correct errors in a source file(s) easily and get complete hypertext help on LATEX and shell functions. The user can switch the color selection of TEX commands and control symbols, add any custom batch file or program to the user menu from inside the editor, or use his favorite external editor, etc. The shell supports the new emTEX directory system under DOS and OS/2 and uses less then 1k of memory when TEX or the previewer is executing. The $_4$TEX shell (last release 3.26) provides more-or-less similar features, but TEXShell seems to be a bit more easy to use, especially with a set of different configurations. Does anybody really need more?

The TEX user certainly can be happy working with the TEXShell or any other convenient shell with the same facilities. Problems arised in the case he needs to:

- control, on the screen, the margins to be used on printed paper;
- install another printer;
- use some new emTEX features (such as booklet printing);
- use one of the very nice extra utilities such as BM2FONT or MFPIC which depends on printer font set; and/or
- repeat all the actions above simultaneously.

In such cases, the user will have to look into the documentation and perhaps write new configuration and batch files again and again without any hope of getting a final version. If you only change printers, for example, You usually have to change the path to write BM2FONT and MFPIC .pk font files, recalculate the extension set to start GFTOPK execution, defining resolution, font directories, screen margins and scaling options for the previewer. Moreover, in order to determine the correct way configuration, you need to collect information from a set of documentation files with the size of hundreds of kilobytes.

None of the existing shells can do it automatically. It is a task for human intellect. And nobody is able to "teach" any existing shell to perform this kind of job. You should know all of the interrelations between a type of printer, a resolution set and mode for METAFONT to make the whole system work correctly. Moreover, the interrelations mentioned above may also depend on font directories, existence of .pk files and other machine-dependent conditions which you should check before you may achieve success. The new version of the same program usually handles the environment and the command line options in different way. It is really difficult.

The experimental TEX shell was designed as emTEX-based, with the support of RFBR to make the following extremely easy:

- the use of any TEX–METAFONT interface-based packages such as MFPIC;
- the use of other packages such as BM2FONT;
- the change of printer driver or changing to another *svga* mode;
- the ability to see the same margins on screen as those on the printed page;
- different configurations for different kinds of jobs (different journals for example);
- writing all temporary files (`.log, .dvi`, etc.) to a special directory; and
- the support of all features of the current emTEX distribution.

The test version of the shell was created based on the TX sources by Ricard Torres (`torres@upf.es`) and which are available on CTAN (a very simple and effective shell for work with TEX under DOS) and successfully tested in the Krasnoyarsk State University, several institutions at Krasnoyarsk and in the Information and Publication Centre of Steklov Mathematical Institute (Moscow). After we replace the temporary Russian fonts with the standard set and probably some other changes as proposed by the project, the shell will be freely available from the RFBR Russian TEX server `ftp.tex.math.ru` via *anonymous ftp* as an add-on package to the current emTEX distribution.

Though it is not the purpose of this paper to describe our shell completely, we need to give some information on it. It contains an extremely small menu, which saves screen space allowing the user to see a maximum of the program finished before output. The menu size depends on the number of files to work with and the number of currently available commands. One available command is Set (hit the "S" key) to call a set configuration menu in which you can select (change) a printer resolution, *svga* mode, portrait or landscape mode, output directions, or printer behavior (for two-side printing, etc.). The format is normally selected automatically, but you can change it if Hit the `<E>` key (Edit) to call the default editor; hit `<T>` to TEX a file; hit `<V>` to preview the file; and so on. Usually the shell gives a prompt for the next command, and you can just press `<SPACE>` to approve the command. If you work with LATEX files, the default sequence for the first time to run will be `<T>_<T>_<V>_<E>`. With MAKEINDEX the sequence will be `<T>_<M>_<T>_<V>_<E>` and with MFPIC `<T>_<F>_<T>_<V>_<E>` where `<F>` starts METAFONT in the appropriate mode, starts GF2PK and generates the correct output files. Unfortunately there is no on-line help available in our shell at this time.

As soon as we announced that we were about to place this shell into the base of a non-commercial "Russian TEX" distribution, the biggest disagreement became what set of packages was to be supported. Almost everybody wanted to restrict the number or packages to use as much as possible, but insisted upon the inclusion of the packages he know best. How to resolve this discussion? The only way we can see is to make a TEX shell to use a widely variable set of packages which are easy to install and use – "unpack an play". The TDS (TEX directory structure) standard gives us a chance to make the packages extremely independent upon operating systems. The more we try to configure the shell for a new package, which is complex yet still easy-to-use, the more time we spend checking and correcting all of the places in the configuration files which affect the package behavior.

Thus we need a new approach to the shell and user interface. It should be a shell with some features of the human mind, or an "intelligent" shell.

What does that mean? Are we to develop a system of artificial intellect only to start TEX-related programs? Of course not! But we want our shell to be able to check logical conditions and change the behavior according to the current situation. This may be useful not only in the configuration but in the interface too. It isn't necessary to have a menu item "Print" if there is no `.dvi` file, as well as there is no need to run METAFONT if an `.mf` file does not exist. It would also be better if the shell would lead us through the menus, choosing the next reasonable item; i.e., after running TEX, the next action is previewing and after that it is another edition in the most cases. The user can agree with the shell by pressing `<ENTER>` or choosing something else if he wants. We believe such advantages makes work a bit more convenient.

In addition to this, we consider a fully detailed, context-dependent HELP as a very important part of the shell. Being concerned with TEX for about three years we have come to the conclusion that users sometimes want to have a more detailed explanation of every menu item than the existing shells support.

How can we improve this service? Nowadays hypertext structures are very much appreciated in user guides on computer systems. Many shells provide reference information organized as hypertext. TEXShell has only the HELP mentioned above. But this is not context-dependent, so sometimes it is quite difficult to find the information you want. Besides, this is a separate part of the shell and the user would need to interrupt work to study what he needed to do next.

So we can say that the help is laid down into the base of the future shell described here. This is a hypertext-based shell. Each hypertext page contains all reference information about marked words. Every such word means an action. It may be running a program or changing its configuration as a new hypertext page which allows one to take the next step. To avoid this information, you are able to enter an "expert" mode and see nothing except for hypertext references. This seems to be quite easy to use such shell.

The other task this shell will have to solve is installation of any additional package which needs to be integrated into the current conglomeration of programs. There are many such packages for TEX now and there'll be many more in future. Each of them presupposes its own way of installation and configuration. Thus, any package which can use the common directory structure may be installed just by unpacking the appropriate archive containing a configuration file for our shell. This file should include all information about behavior of this package and have some machinery for configuring the package. The shell definitely cannot know all packages' names. That is why the configuration files should have a special extension, say TXC (for TEX Config).

Now we want to consider the system of configuration files for our shell in more detail. Besides the packages' configuration files mentioned above, the system must comprise the main *read-only* file.

It should be emphasized here that this file is the shell itself. We mean that the shell is only a program written for an interpreter of a new language which allows creation of hypertext with the functions of a shell.

The next standard file is a file which contains information about the current state of a shell. This includes some variables and technical data. The other file is user-defined, where one can put the name of the work directory, necessary format, type of printer and so on. One more file of the same type is the job configurations file. Here are the common options for a group of users which do similar work. Another important file describes various devices used by printers and some programs. And last, but not least, the station-independent file where common commands for all supported platforms options are collected. Having these files, one can be comfortable working on any station where this shell is installed. You can prepare your text on one station, compose it on the server, view on another station and print somewhere else.

We are not yet ready to describe the programming language which allows creation of such shells; we can just say that it was developed for purposes like this and we are still working on it. We only show one example of a very simple configuration file. It isn't a complete shell;however, it gives a glimpse of the new language under creation: As you can see from this example, this language is quite similar to TEX but digits and other special TEX signs are referred to as strings. One-line remarks preceded by % are allowed. The command `\newpage` defines the highlighted words in hypertext, followed by page and the hot-key which one can press as well as the 'Enter' key to achieve the next page. Language's devices provides various detail levels of reference information which one may use in the shell. It may be whole pages for a novice or simply highlightings for an expert. `\def` looks like TEX's and is very easy to understand. It defines new hypertext pages as are used in `\newpage`. There are some logical primitives for `\if` in addition to `\exists`. They give one an opportunity to compare dimensions like texttpt, in, `cm` and so on as well as numerals and strings. Standard environment variables defined above are available as `$name$`. For example `$mfmode$` means `canonbj` if you use Cannon Bubble Jet printer in your system. The command `\addnewpack` finds the configuration file of new package and reads it contents. This file should include everything about the package itself and all interrelations with other packages too.

The package configuration file can contain special commands describing what must be done *before* execution of the main commands of some other package(s) and *after* it. This looks like a possibility to avoid user problems of installation. Only unpacking will be required, even for a rather complicated package such as MFPIC.

We did not have the intention to describe a completed thing. The aim of this paper was to discuss a new approach to this problem. In a year we plan to put the first version of a new shell with a sources on a Russian TEX server to be freely available at least for non-commercial usage by *anonymous ftp*.

```
%%% A simple configuration file for the shell
\def{\TheHomePage} Now you can compose your file using
\newpage{TeX}{\TeXpage}{T}
\if \exists{$name$.dvi}
or view composed file by  \newpage{DviScr}{\dviscrpage}{V}
and print it on the printer \newpage{Print}{\printpage}{P}
\fi
\if \exists{$name$.aux}
\if \infile{$name$.aux}{\bibdata}
And now you can call \newpage{BibTeX}{\bibtexpage}{B} to
complete the bibliography.
\fi
\fi
%%% Below the approximate scheme of adding new packages is shown
%%% the command \addnewpack  adds found in the directory ../texmf/txc/
%%% files <package_name>.txc to the page \packages

\addnewpack

From the moment of the latest upgrade of our shell the
\newpage{New Packages}{\packages}{}
is appeared.
}

%%% The  Home Page  definition is completed.
%%% Now we should define the pages mentioned in it
\def{\TeXpage}{By pressing the word \newpage{\TeX}{\runshellscript}{}
you run TeX and provided there are no mistakes new file $name$.dvi will
appear.
\if \exists{$name$.dvi}
And you can \newpage{View}{\dviscrpage}{V} it.
\fi
You can configure  TeX on the page
\newpage{configuration}{\texconf}{}}
...
```

Figure 1: Sample configuration shell

Acknowledgements

This work was supported by Russian Foundation of Basic Research grant 95-07-19400v.

The Strait Gate to TEX

Ivan G. Vsesvetsky
Ioffe Physico-Technical Institute, 26, Polytechnicheskaya str., 194021, St Petersburg, Russia
vig@ioffe.rssi.ru

Abstract

TEX is the best ... It is 'ideal'! So why has it not gained the world? Because (1) it's not for everybody; and (2) this world is *REAL*, not ideal. There exist a real competition between TEX and, say, Ventura.

This article shows how capabilities of TeX can be enhanced to a level surpassing 'professional' typesetting.

What this Article is About

For more than three years Ioffe Institute has prepared camera-ready copy for four physical journals (about 1000 A4 pages of `11pt` text plus illustrations per month). `plain` TEX with *CyrTUG* russification and local styles are used. This article is an attempt to generalize the experience of a small TEX publishing bureau.

All speculations are *emphasized* for easy skipping.

Why TEX?

The journals were published for years with metal type. In about 1991, the type became too old and the publisher ("Nauka") moved to DOS Ventura. It appeared that doing formulae with Ventura is a hard manual toil was quite toilsome and the editors (they are all from the Ioffe Institute) turned to our computing center, which had some experience with scientific publishing. Thus, from April 1993, the journals are produced using TEX.

Problems

It would be strange if we did not encounter problems attempting to emulate metal type with TEX.

Personnel education *NonTEXnical people consider TEX too complex for a human being without a university mathematical education.* Most our TEX typists were mainframe operators before 1993.

Limitations of emTEX Up to 1993, we used PCTEX and TurboTEX. These commercial implementations gently died in the Ioffe Institute after the appearence of wonderful emTEX.

Only one part of emTEX caused some problems — `dvi` drivers for DOS — they abort quite often on our documents that are filled with graphics and virtual fonts.

The main reason for using DVIPS is the inability of emTEX to please our editors — it cannot do landscape tables.

Free software is... good or bad?

Bad. Because it is not debugged.

DVIPS supports emTEX `\special`s, but sometimes PCX files are displayed as a black boxes (very rarely), and at other times raster graphics are distorted.

Good. Because we can debug it ourselves.

The availability of the source code for DVIPS made it possible to overcome the above errors.

DVIPS is slow Most grahics typed in the journals are hand drawings scanned to PCX format. DVIPS, preparing its PostScript output, unpacks them (expanding thee to four times) and codes them in hexadecimal (a 2-fold increase). The resulting giantic file (30–50MB at 300 dpi resolution) can take from three to five hours to print. It is rather easy to estimate the resources needed for 600 or 1200 dpi (last year we got such devices).

After some experimenting, the algorithm of (rather) fast graphics decompression was coded in PostScript [1] and the compression counterpart was included in DVIPS. Now DVIPS works 1.5–2 times slower than emTEX driver on the same file, and this coefficient does not grow with printer resolution. After CCITT 'FAX' compression [2] was implemented in DVIPS for the new PostScript Level 2 printers, it runs even faster than DVIHPLJ of emTEX. Patches for DVIPS are available on request.

Lack of fonts *Good Cyrillic TEX fonts are scarce.* Up until now, it was not a problem — the editors are happy with the Washington Cyrillic fonts.

Placement of illustrations *is not easy with TEX. Here Ventura (or Word, etc.) certainly wins. It seems to me that the problem cannot be solved within*

TEX — content (visual in this case) should not be generated automatically.

Prospects

We are planning to move to LATEX.

We are beginning to use PostScript fonts. Problems with screen preview in emTEX are solved using `ps2pk` program. Matching different fonts and their adaptation to DVIPS is done with quite simple AWK scripts.

Conclusions

The ex-Soviet desktop publishing market is rather strange. It seems that the only professional typesetting software available here is free *TEX. Its 'freeness' is imaginary of course — you must spend (at least your time) on personnel education, software debugging and tuning, etc.*

References

[1] Adobe, *Postscript Language Reference Manual*, Addison-Wesley, 1990.

[2] Brown, Wayne and Barry Shepherd, *Graphic File Formats*, Prentice-Hall, 1994.

DVI-based Electronic Publication

Laurent Siebenmann
Matématiques, Bâtiment 425, Université de Paris-Sud, F-91405, Orsay, France
lcs@topo.math.u-psud.fr

Abstract

This article offers an introduction to three proposals that would make TEX's original device-independent output format `dvi` more useful for electronic publication of scientific documents.

The race is on to establish formats for publishing the world's scientific literature in purely electronic form. Let us first recall the technological circumstances that have set this race in motion.

It is widely believed that the Internet now has the capacity necessary to deliver scientific articles in just a few seconds to any computer on the Internet — via WWW (World Wide Web), and also older systems such as FTP (File Transfer Protocol). Hopefully, the costs for Internet transmission will remain quite negligible in spite of commercialization. Rapid access (hard disk) storage cost per page in an electronic library is already negligible. Provided efficient storage formats are used, the cost is now less than one cent per page for the 4-year life-span of a hard disk — and it is still rapidly falling.

However, there are serious traffic congestion problems that currently nearly freeze the Internet for key portions of the 24-hour day throughout considerable parts of the world. Thus, a couple of complications should be kept in mind. Firstly, although delivery via Internet is the favored vehicle, alternative delivery formats may have to take on an important slice of the action, notably compact read-only optical disks, known as CD-ROMs. This delivery format is slow and chunky, but its cost is low, and its capacity is probably orders of magnitude greater than those of the Internet. Secondly, a congested Internet will be more tolerable if we have a particularly efficient standard for science such as `dvi`.

Optimists expect that public fascination with bulky imagery available on the Internet will push data transport speeds and cost efficiency on the Internet *ahead of and beyond* the needs of the browsing scientist. But this assumes that scientists use efficient norms insulated from technological inflation.

The leading candidates among electronic formats for the mathematical sciences seem to be:

`tex` : TEX's ASCII typescript input format defined by Knuth and elaborated by TEX markup formats such as Plain, LATEX and others. This format usually must be enriched by `\special`-ly included graphics objects. Hence it should perhaps be called the `tex_etc` format.

`dvi` : TEX's binary device-independent output format. Similarly, we will talk of a `dvi_etc` format that evolves with the `\special` additives and their attendant software.

`ps` : the Adobe PostScript page description language, accepted by a majority of 'laser' printers, i.e., laser printers.

`pdf` : Adobe's format for its (currently) freeware viewer, Acrobat, nourished by its commercial `ps` to `pdf` converter, Distiller.

`html` : the WWW Internet format used by most browsers. Mathematical material has been skimpily served, except through numerous bitmaps, and these have proved cumbersome. However, direct support for simple mathematics is being implemented under version 3 of `html`.

How do these formats measure up to expectations of users of the scientific literature? For me, the major fault today of all of these formats (judged along with the software and hardware that support them) is low bandwidth for browsing. Indeed, one still assimilates far less of interest in an hour of electronic browsing than in a well-run physical library. The hypertext features and net references (via URLs) are worthy innovations, but they do not fully compensate for slowness.

I believe that this is a sign that browsers (also called viewers or previewers or screen drivers) are in their infancy. Their speed should improve with increased microprocessor power — provided system complexity does not outstrip the power increase. The 10-fold increase in processor clock speed from 8 to 80 mhz in about 10 years augurs well. As does the doubling of bus width from 16 to 32 bits. One *bad* omen is that the resolution of images will lag behind — if the progress from 70 dpi to 100 dpi in 10 years is any indication.

Equally worrying is the widely heard prediction that Internet services will be too expensive for third-world countries and also for any scientists without robust financial support. It is based on the observation that the Internet is being inexorably commercialized.

One can hope that a combination of growth and competition will make the necessary services affordable. However, there is a grave problem casting its shadow on this optimism — it can be stated as follows. Prices tend to be as high as a market can bear,[1] while science teachers and researchers have relatively little financial clout (and students even less). Thus, we would be wise to ask whether scientists will not be playing a losing game in the same Internet league as business and professional groups.

My answer is the same as for transmission speed. To assure excellent service from the Internet in spite of lacking financial clout, scientists should adopt their own particularly *efficient* methods of electronic publication. So efficient that even third-world scientists will have adequate access to the electronic literature.

In the struggle for effective communication, the scientists' secret weapon is TEX. Secret because TEX's inherent complexity will (I believe) restrict its use to a small circle centered on the scientific community in which it was developed. Donald Knuth paid extraordinary attention to efficient operation of TEX: the input `tex` typescripts and the binary output `dvi` page description files have information density comparable to that of ordinary ASCII prose (to within a factor rarely worse than two or three). Like prose they are compressible to well under 50%.

To some of you, the idea that electronic publishing with TEX should be done 'on the cheap' is revolting. I claim that flexibility of design can allow the same TEX systems to do electronic publishing involving sophisticated typography and high tech — for example color MPEG animations. Better, I want the same posting to serve both poor and rich scientists: a poorly equipped reader might see a still, black-and-white image instead of the color animation, but he should be able browse the same posting.

In prospective summary, our goal should be to extend the domain of extraordinary efficiency of TEX from the paper publishing world of the 1980s to the electronic publishing world of tomorrow — and without sacrificing high-tech ambitions.

To return to a more specific question: In what format should science be electronically published? Probably no one format will prove superior in all respects. Scientists should reckon carefully where their best interests lie, and be ready to provide some development effort at crucial points.

The `tex` format (in LATEX dialect) is currently standard on the pioneering preprint server `ftp://xxx.lanl.gov`, initiated by Paul Ginsparg. My article [7] can be interpreted as an attempt to make Plain TEX as efficient and as archival a markup system as LATEX but by a radically new approach: macro compilation. In my view, however, `tex` format is best for author and publisher; I believe it is too complex and fragile for the reader.

The `dvi` format, although intended for the job by Knuth, seems to be in a checkmate position where browsable publication is paramount. As I will now explain, it could lose out entirely if not further developed.

The `pdf` article format is browsable since both text and graphics are more-or-less instantly viewable (as well as printable). Exotic languages can be handled because `pdf` tends to include necessary fonts or the relevant parts thereof.

In contrast, a `dvi` article format is, as of today, probably only printable at best. Here is a quick rundown on the difficulties which it faces and which are addressed by my three proposals. The printability of `eps` graphics inclusions seems to depend on future implementation of a proposed standard for `\special` syntax recently published by Rokicki [5]. The browsability of `eps` graphics is a largely unsolved problem except for the lucky few who have a 'dvi-to-ps' converter and an exceptionally good PostScript viewer. Finally, where one of the more exotic European languages such as Polish and Czech is involved, both printing and browsing currently usually depend on the weighty EC (alias DC) font collection (outside of Europe, expect to be disappointed when you call for it!).

On the other hand, this `dvi` format, which was designed by Knuth's collaborator David Fuchs, has one signal advantage. It is by far the simplest format for the representation of TEX output; indeed it is presented in Knuth [3], along with helpful commentary, in less than 10 pages. Alternative formats such as `ps`, and `pdf` are easily ten times as complex. Therefore `dvi` is comparatively easy to interpret and transform. For example, `dvi` is readily converted to `ps`, while the (possibly useful) inverse

[1] This occurs particularly with monopolies and oligopolies, which are widespread in the world communications industry.

conversion of ps to dvi plus eps seems never to have been attempted.

I will focus on the following three proposals, which are intended to help the dvi format to avoid checkmate and then, just possibly, to go on to surpass all competitors! Originally, I explained these innovations on Internet discussion lists, and my often eminent interlocutors made significant contributions. Where the text of this article is concerned, particular thanks are due to Hans Hagen and Tomas Rokicki for some close criticism. Each proposal remains tentative in the sense that it is just a sketched plan for action waiting for support. Developers can immediately examine many details by consulting the list archives indicated below, or by contacting me personally.

- The first proposal is a set of small auxiliary 'atomic' fonts to complement Knuth's (atomic!) Computer Modern CM fonts so as to serve all European languages with a Latin (non-Cyrillic) alphabet. Here is a (last?) opportunity to make Knuth's original CM fonts part of an efficient and durable international system. The new auxiliary fonts would undoubtedly be extracted from the rather comprehensive EC font system currently being completed. Beyond the English-speaking world, typesetting would normally use virtual fonts with composite accented characters based on these atomic fonts. A utility such as Peter Breitenlohner's dvicopy would finally convert the resulting dvi files to the more portable form using only the atomic fonts. The strong points of this scheme are unrivaled efficiency, and compatibility with the core of TEX users who have no reason to abandon Knuth's CM fonts.

- The second proposal is the notion of a 'multi-standard' graphics object for inclusion in a TEX document. This notion was initially designed to solve the old problem of providing portable previewing of eps (encapsulated PostScript) graphics. But it promises much wider compatibility: a single dvi-based posting should be able to serve all the multifarious graphics formats, and all the abilities and debilities of TEX viewers and printers in all computer environments. At least one programmer of a dvi viewer has been developing very much the sort of mechanism I recommend, namely Hippocrates Sendoukas, with his viewer *DVIWin* for PCs. The near-coincidence of our independently thought-out ideas is a good omen, suggesting that, in the present instance, there may be essentially one way to skin the cat neatly.

- The third proposal can be viewed either as a very general approach to implementing any standard for 'special' syntax or, alternatively, as a way to achieve dvi portability without such a standard. On each of the major platforms, one creates a freely available utility program converting a (candidate) standard for specials to the native format of any selected TEX dvi interpreter on that platform. The onus of implementation of a standard can thereby be entirely shifted from the implementors of TEX screen and printer drivers to the publishers of dvi-based postings. A low-profile utility project will be discussed in some detail.

An 'atomic' complement to CM fonts

One problem with dvi postings is specifically a European one. No single font encoding quite manages to cover the needs of all European languages that use a Latin-based alphabet. It is well known that Knuth's CM fonts are not sufficient for the Poles, and Czechs. Even the basic EC font encoding (the Cork norm) with numerous accented letters is not sufficient for the Latvians, Welsh and others. There are quite simply more than 256 characters in all. The auxiliary text font series tc of the EC collection [4] will probably have to be combined with virtual fonts to bring them under the common EC roof. This will tend to render non-portable the dvi files involving such languages.

As for the Cyrillic alphabet of our hosts at Dubna, so few accents are used in modern scientific Russian, that conceivably just one 256-character encoding including compound characters may suffice for use by scientists — much as classical CM suffices in the English-speaking world.

My suggestion for the Latin alphabets is to supplement Knuth's CM fonts by a single parallel series for auxiliary 'atomic' characters. It would contain chiefly separate accents, in distinct versions for lower- and uppercase letters. But it would also include some notoriously troublesome characters, for example, the Slovak character ľ (typed as \v l), which is difficult to render reliably and beautifully from atomic characters. Such troublesome characters are to be treated as atomic.

Note that the intervention of virtual fonts makes it possible to assemble composite characters from 'atoms' in distinct fonts; thus, the CM fonts need not be modified in any way in order to be the core of such a system.

To achieve optimal typographic quality the author will use virtual fonts — but the virtual fonts are to be subsequently 'atomized' and so eliminated from the posted dvi file, for example by use of Breitenlohner's dvicopy. There is a significant typographical benefit to be gained from the use of virtual fonts: accent positioning can be adjusted to suit any national typography. For example the dieresis (Umlaut) accent is usually placed lower in Germany than Knuth's, whose accent positioning seems most acceptable in France.

This scheme is compatible with the (core) EC fonts in the sense that virtual EC fonts (hopefully with the true EC metrics) can be based on CM and the envisaged atomic complement to CM. This imposes inclusion of some miscellaneous characters that made their TEX début in the Cork encoding.

To be widely adopted for browsing electronic scientific postings, such complementary atomic fonts will have to be made available in Adobe Type 1 format, matching the BSR and BaKoMa Type 1 font series.

A somewhat similar proposal (independent of mine) for Adobe-distributed Type 1 fonts was made by Pierre Mackay. It influenced the May 1995 encoding named 8r (for 8bit raw) used by the PSNFSS font system of LATEX fame. But 8r is not atomic — if it had been made atomic (as the sadly lacunary Adobe Standard Encoding nearly was) that would have denied access to numerous accented characters present in the Type 1 `afb` files and reduce them to extra baggage. In my view, the CM fonts of Knuth are unique and deserve unique treatment.

The window of opportunity for acting on the proposal above is opening up with the approaching completion by Jörg Knappen of the EC fonts. The main discussion of this proposal took place on the Math Fonts list [2] from March through May 1994; hopefully, its archive will remain available. Technical problems related to the notion of 'drift' are discussed there.[2]

EPS graphics integration

The most glaring defect of `dvi` format has hitherto been the lack of a standard 'special' syntax for inclusion of graphics objects via TEX's `\special` command — for it tends to make `dvi` files non-portable.[3] Where `eps` graphics (i.e., Encapsulated PostScript graphics) are concerned, a specific proposal for standard 'special' syntax has recently been put forward by Rokicki.

Unfortunately TEX screen viewers are, more often then not, unable to directly view `eps` graphics. Prompt to recognize this problem, Adobe provided standard enhanced formats for screen viewing on Macintosh, and DOS (or MS Windows) systems.[4] On Macintosh, the enhanced viewer file includes a PICT resource numbered 256 in the resource fork of the `eps` file; it contains either a "packbits" compressed bitmap or (more recently) vectorized PICT graphics.[5] On PCs, the file has a binary format sometimes called EPSP; this has a characteristic header, a segment for the `eps` file, and a segment for the bitmapped preview; the latter may be a simple `tif` bitmap with "packbits" compression, or a vectorized `wmf`. Here, `tif` means Tagged Image Format File (TIFF is the official acronym), and `wmf` means Windows MetaFile.

If the enhanced PC and Macintosh viewer formats had been one and the same format, it would surely have been adopted overnight as a standard. But, as matters now stand, the enhanced `eps` files pose an annoying portability problem — and neither seems to have been widely adopted on UNIX systems — which tend to rely on PostScript interpreters for graphics previewing.

Thus, in practice, the standard proposed by Rokicki will assure portability between printers but often not between TEX viewers on different platforms. This puts `dvi` format in grave jeopardy as a publication format.

The portable viewing problem for `eps` graphics could (as noted above) be solved by setting up a viewing standard common to the Mac and PC platforms. Here is an obvious approach: have the `eps` text file `myfig.eps` accompanied by a parallel `tif` bitmap file `myfig.tif`. With modest effort, viewer builders on all platforms could derive the desired preview from the `tif` bitmap. Such bitmaps can be displayed as fast as the surrounding text. If the bitmap is scaled downwards (not upwards!) to its screen size, the image quality is usually good — at least for the line figures that dominate in the mathematical sciences. For best quality, figure labels should be put in a TEX overlay. The syntax proposed by Rokicki is adequate; so one merely enhances the functionality to provide previewing using the `tif` bitmaps.

[2] These problems (rather minor ones, I believe) could be entirely bannished by a TUG standard for `dvi` interpretation.

[3] The `tex` format does not have an acute problem here. My `boxedeps.tex` (Feb. 1991; a TEX macro package available on CTAN) first made `tex` typescripts using `eps` graphics portable in spite of this lack of `\special` standardization. LATEX has recently adopted a similar approach (basically one of systematically accommodating all extant syntaxes) and extended it to `\special` commands governing color, rotations, etc.

[4] Adobe also provided a device-independent preview format EPSI (I for Interchange). Alas, its bulk is high and its quality low, and it is almost never used.

[5] This view-enhanced Macintosh `eps` file format is universally supported in the Macintosh world. The benefit justifies the considerable cost in space: typically the compressed volume devoted to graphics is doubled.

This solution seems worth implementing since it would solve an urgent problem. The basic functionality, i.e., use of a parallel `tif` file to give a viewer image corresponding to an `eps` file is already present in the two MS Windows viewers *DVIWin* [6] and *DVIWindo* [9]. They are by Sendoukas and Y&Y Inc., respectively. Unfortunately, such a development seems unlikely to come about spontaneously...

Multi-standard graphics integration

The above potential solution for `eps` graphics integration with previewing is a very particular case of the use of several graphics standards to represent a single graphics object. Would it not be worthwhile to have a general scheme that goes well beyond the viewing problem as Adobe understood it ten years ago? We need a multi-standard scheme that can provide viewers with the very best screen representations available for any given platform!

This seems to me a bold enough idea to command the respect of a standards committee. I am hopeful that any number of distinct graphics norms can be handled by one and the same protocol so as to allow the same `dvi` file to serve on all platforms under all circumstances. Sendoukas was perhaps the first to envisage this possibility; see the documentation of his *DVIWin*.

Fortunately, the sort of 'special' syntax proposed by Rokicki seems adequate for the functionality we want. For example:

```
\special{:: object multifig=myfig
                        width=200 height=120}
```

might be be interpreted as follows: there is a graphics object whose name root is `myfig`; its representation norm is to be selected by the driver among those available; the image is to be linearly scaled to fit exactly onto the rectangle 200 bp wide and 120 bp high, with its lower left corner located at the TEX insertion point (all mentioned dimensions are thereafter corrected by TEX's magnification factor). Thus, in this most convenient case, the image rectangle (on paper or screen) is specified in the `\special`'s argument string viewed in isolation.

More specifically, consider the integration of a graphics object `myfig` that is represented as a high-resolution color bitmap `myfig.jpg` of JPEG norm for viewers that have such wonderful[6] capabilities, and by a simple `tif` bitmap file[7] `myfig.tif` designed to serve as a lowest-common-denominator format for viewers without JPEG display capabilities. We explain the integration in terms of this example, which is one that possibly no TEX viewer yet handles today. Indeed, Possibly the one-file `eps`-with-preview offered by Adobe blinded most developers to the virtues of the more general multi-standard scheme.

Only the article author's intended experience, and that of the browsing reader needs to be described, for we have already indicated the sort of 'special' syntax that TEX and the viewer will manipulate behind the user's back.

The user will exploit a macro package for integration of multi-standard graphics; this package will have syntax similar to one of the several integration packages for `eps` graphics.[8] (The TEX user rarely wants to have to work with explicit figure dimensions!) Thus, figure placement and size will be adjusted using conventional macro commands such as `\ForceWidth{0.75\hsize}`.

Much as with LATEX's cross-referencing, the result of a first production cycle is likely to be disappointing; typically the figure is crammed into a squared-out box. This is because TEX is quite unable to find out the shape or size of the figure since both graphics files mentioned are binary and TEX reads only text files. But the viewer can conveniently take on that task,[9] and the author is instructed to preview and expect good results on second and later runs.

If the viewer is able to accept JPEG files, previewing is in color, and, if not, the image is the black and white familiar for `eps` previewing.

To post the `dvi` file with its graphics inclusion, the author packages it with the `jpg` and `tif` files in a directory — whence the posting format name `dvi_etc`. He should omit an ephemeral ASCII file `myfig.gdf`, say, a temporary 'graphics description file', which is created by the viewer or alternatively by the utility mentioned in the last note, and which serves to pass on to TEX the bounding box information on the graphics object that TEX initially could not read.

What happens when a user browses this `dvi_etc` posting with a different viewer supporting the proposed multi-standard? The user immediately

[6] Notably unrivaled compactness and nearly perfect scalability.

[7] Simple `tif` means black-white, 'packbits'-compressed, and with strip structure but no tiles. This is the sort of preview that Adobe Illustrator can include inside `eps` files for PCs.

[8] It can be derived from the existing version by relatively minor generalizations.

[9] Alternative: It would be more conventional to assign this task to a utility that is to be run first off; on the other hand, one utility per platform would suffice, and a disappointing first run would be avoided. For example, on on PCs, a utility called TIFFTAGS can be used to deal with `tif` files.

sees the best image his viewer can offer on the basis of available files. But let us imagine that the user sees only the modest `tif` and is unhappy because he knows that his viewer can handle good-quality `gif` images (but not `jpg`). Here the multi-standard again comes to the rescue: it suffices to convert from `jpg` to `gif` and the high-resolution images appear![10]

Popular PICT and WMF scalable graphics become portable. The above JPEG example of multi-standard graphics integration is forward-looking. Here is an application that gives an immediate payoff. One creates a preprint posting using the Macintosh's native vectorized PICT graphics (suffix `pct`) and the multi-standard 'special'. A notice with the electronic posting recommends that users on other platforms convert the PICT graphics to their favorite local norm, which for PCs means a Microsoft Windows MetaFile with extension `wmf`.[11] The multi-standard graphics integration using a single 'special' syntax will assure that the graphics appear on the end-user's screen as soon as a version in his native norm is available. Successful viewing of vectorized graphics almost always implies optimal printing, but not conversely.

In summary, with multi-standard graphics integration, one can confidently use and post with TEX the native vectorized (scalable) linefigures of Macintosh and Microsoft Windows. The viewing is then of optimal quality at all scales; this includes the labels if they are based on hinted outline fonts. A high degree of portability will be assured.

Since there is, at long last, a technical working group devoted to 'special' standards [5], let us wish them the intelligence, openness, stamina and goodwill necessary to deliver a powerful standard. The long-term health of TEX's `dvi` format requires such a standard.

The Internet discussion of the above graphics multi-standard proposal can be consulted on WWW as a hypermail archive:

`http://math.albany.edu:8800/hm/emj/`

Assisted portability of dvi files

The classic retort to pious predictions concerning future standards is: "Don't hold your breath while waiting!" If the past is the measure of speed of progress with 'special' standards it could be another decade before an official standard deals frontally with multi-standard graphics. This educated skepticism drives me to propose an immediately applicable scheme, one which is similar in spirit to the `boxedeps` utility that I developed in the period 1989–1991 to make `tex` typescripts with `eps` graphics immediately portable.

This last proposal can be viewed either as a very general approach to rapidly implementing standards for 'special' syntax (graphical or not!) or, alternatively, as a way to survive without such standards.

Zapping specials We first discuss a bite-sized form of assisted portability. We consider a small but embarrassing problem that has conjecturally held back the dissemination of `dvi` files containing *Hyper*TEX specials. The shortage of viewers that support the specials is not the culprit. In principle, electronic science journal postings in `dvi` format should *all* include *Hyper*TEX specials, since these potentially offer the revolutionary viewer features that are making `html` and `pdf` postings so attractive. The presence of these specials would be a strong incentive to TEX viewer modernization and a payoff to those who have installed *Hyper*TEX viewers. So why are *Hyper*TEX `dvi` files so rarely posted? Mostly, I claim, because so many existing viewers and printer drivers complain loudly about unrecognized specials, and sometimes go so far as to halt processing. This is quite incompatible with my own interpretation of Knuth's intentions for the `\special` command (see *The TEXbook*, pages 228–229). I would say that he recommends a driver execute only specials it understands; and that, for those specials it does not understand, the driver should do exactly nothing, gracefully. Clearly many programmers have a different interpretation and espouse a less tolerant policy; and unfortunately, Rokicki does not clarify this issue.

And so, my introductory challenge is: Find a way to let existing *Hyper*TEX specials circulate in `dvi` files without wreaking havoc where *Hyper*TEX is not supported. There is a practical response. In view of the relative simplicity of the `dvi` format, one can make available on all platforms a small utility to automatically delete all specials from any `dvi` file. Thus, specials that cause a fuss can be suppressed whenever necessary. In reality this 'zapping' would

[10] There is a regrettable tendency among users to convert `jpg` files at an early stage to `eps`; this often means the file is encysted in the `eps` in a bulky form that moreover usually cannot be exploited for screen viewing. A best possible format, such as `jpg` for color bitmaps, deserves to be posted as is.

[11] Powerful conversion utilities are becoming available on most platforms. For the many users who refuse to become involved in conversions, one can hope that WWW servers will soon automatically supply to any WWW-connected viewer the graphics norm(s) it requests. Sendoukas recommends an alternative: viewers and printer drivers could ally themselves with conversion utilities so that they become able to exploit a broad spectrum of graphics norms.

probably be just one among several functions of a dvi utility.

There is a related development that should be mentioned here since it inspired the proposal for assisted dvi portability. Geoffrey Tobin has defined a new ASCII version dvl of dvi format along with converters in both directions [8]. As it stands, Tobin's system allows electronic publishers to adjust or update 'special' syntax using a programmable editor. On the other hand, any autonomous and fully automatic utility for users would probably better operate directly on the binary dvi files.

After the above 'teaser' featuring 'special zapping', let us consider more important roles for utilities that propose to adapt dvi-based postings to the idiosyncrasies of viewers and printer drivers. There are many possible profiles for such dvi 'localizer' utilities; I mention two.

A) High-profile utility Given any dvi file, this utility would convert the specials in it to those required by a driver to be specified (within bounds) by the user. Reasonable bounds might mean: to suit any of the screen and printer drivers present on the platform where the utility operates. Such a utility would certainly be worthwhile, but it would be of considerable size, and tiresome to construct, because specials and their syntaxes are numerous and various. There would also be a big maintenance problem since specials are proliferating; thus, one might frequently encounter messages to the effect that certain specials listed in the log file could not be understood, and that, just possibly, a more recent version of the utility would do a more thorough job.

B) Low-profile utility This would be designed to serve only quite restricted dvi files — for example, those adhering to an ad hoc specials standard. This seems very suitable for a group of electronic publishers.

Let us consider a hypothetical but plausible instance. A group of electronic scientific journals decides that, for their current needs, a limited set of specials will suffice — say, those governing color, eps integration with bitmapped previewing, basic hypertext, and net references (URLs). They post dvi files plus graphics files adhering to their ad hoc standard and they build the corresponding utilities for all platforms and post them conspicuously. I provisionally call both the format and the utility dvi_etc.

I believe that such a low-profile scheme is ripe for implementation. So let us examine how some obvious difficulties can be overcome.

Problem (i) The posted dvi had better not contain the ad hoc specials standard in an active form, since many users will be assaulted by complaints from touchy drivers.

Solution: The posted dvi can have its ad hoc specials removed to an external module, the 'special' locations being remembered by a system of pointers. Then the dvi file contains no true specials whatsoever, and there will be no complaints if the uninformed user directly views the dvi. In that case, there should be a clear notice on a cover sheet (the first dvi page displayed!) indicating that all specials are currently inactive and should be activated by use of the dvi_etc utility. When that is done, the cover sheet changes to indicate for which driver(s) the dvi is now suitable and how versions for other drivers can be obtained.

Problem (ii) It should be possible to arrange for automated reception and viewing of downloaded documents.

Solution: Scenarios for automation will vary. Supposing (for example) that the reception platform is Macintosh, Mime conventions can assign to the auxiliary module brouwer.etc a type and creator so as to launch the dvi_etc utility. If this utility has been parametered for a specific dvi viewer and put on alert status, then conversion of the posting to the format of the driver can proceed automatically and the localized dvi file can appear automatically in a window of the viewer.

Problem (iii) Graphics objects in viewer-ready bitmapped form never have the same format for MS Windows and Macintosh systems. But, for simplicity, it is highly desirable to have one and the same graphics material for all platforms. What to do?

Solution (for b/w bitmaps): Since the conversion of the simplest tif to a bitmapped Macintosh PICT resource is not difficult to program, the external module need only contain the tif bitmap. The utility should therefore (as necessary) combine tif plus eps files to create the Macintosh or PC preview-enhanced eps file format of Adobe.

Thus, an article brouwer can be posted in two parts: brouwer.dvi, a classical dvi file (without specials), and brouwer.etc, an auxiliary module containing 'special' material. There is also a one-file alternative mentioned below.

Problem (iv) The ps and pdf formats present an article as a single file including the graphics. To compete successfully, a dvi-based posting should be equally unified.

Solution: This problem more or less disappears if each article is presented as a zip archive file. Zip is an interplatform standard for packaging and compressing files and/or directories. It attends to such niceties as directory structure, the text versus binary distinction, Macintosh types and creators, and UNIX permissions, etc.

Improved Solution: Anselm Lingnau lingnau@tm.informatik.uni-frankfurt.de pointed out on the EMJ list that, at the risk of violating the letter of the law for dvi format, it is in practice possible to insert an arbitrarily long segment of material into a dvi file without influencing its behavior under any known driver. (His aim was to include pk fonts in dvi files.) More precisely, in the terminal segment: *post* ⟨postamble⟩ ⟨font definitions⟩ *post_post* $q[4]$ 2 223's$[\geq 4]$, one is free to insert extra stuff before the four-byte pointer $q[4]$ to the file offset of *post.*

Thus, all the 'special' and graphics material for an article can alternatively be located in a single file, which will itself behave as a dvi file. (This dvi should have a cover sheet that tells how it could be exploited more fully.)

This insistance on a one-file format is somewhat misguided. When interplatform standards such as tif or eps are involved, it can be helpful to informed users to present such files 'tel quel'. For example, in the absence of a viewer with adequate graphics capabilities, tif graphics could be viewed autonomously.

Notice that in this low-profile dvi_etc scheme, the material a user fetches from a server is always independent of his computing platform, so it lends itself to email delivery and to exchanges among colleagues, i.e., to situations where the sender has no knowledge of the addressee's platform. (However, to serve platforms not yet equipped with a dvi_etc utility, it makes sense to dispense 'prelocalized' dvi_etc files suitably requested by email.)

The dvi_etc format is still in flux; it will have to be designed with care for maximum flexibility since use of specials knows no bounds.

The politics of assisted dvi portability This new strategy — particularly in the above low-profile form — seems to have many advantages:

(a) the total amount of work involved is hopefully less than if individual driver builders were all asked to retrofit a new standard;

(b) by creating such a coherent system of utilities one can provide a universally valid distribution format without having to extract assent and effort from every last driver builder;

(c) upgrading such a format is a pay-as-you-go affair, which can be transparent to the end-user and to the driver builders;

(d) competing formats can coexist peacefully;

(e) a small cross-platform group of like-minded developers is able to implement the system.

Scorecard

Here is a quick ranking of dvi format measured against its two closest rivals, ps and pdf.

- reliability: dvi and ps are very reliable whereas pdf is still having numerous very serious teething problems. I will not discuss them except to point out that they are providing a few years' grace for the TEX community to come up with a competitive dvi-based format.

- viewer availability: dvi and pdf rank about equal for scientists, while ps ranks last since ghostscript-based PostScript viewers are not widely installed outside the UNIX world.[12] This could change rapidly in the PC world. But, in the Macintosh world, there is still no PostScript viewer that is suitable for browsing.

- format portability: The straggler is the dvi format: Only printed English text with mathematics is currently portable in dvi format; graphics or other 'special' enhancements are non-portable. That is not good enough for dvi format to hold its own as a publication format. But the proposed standard [5] and the three proposals of this article are aimed at extending this portability to cover essentially all scientific publication in electronic form.

- print quality: dvi is the currently the winner because for most printers (PostScript and non-PostScript) there exist several good drivers. For the PostScript printers dvi and and pdf rank equal and offer optimal quality. The ps format is slightly inferior in current practice; for, by tradition, non-scalable pk fonts of 300 or 600 dpi resolution are used.

- print convenience: ps files often pose problems for lack of a parent application to manage the printing operation. In particular, the printing of a part of a PostScript file is too often out of reach of the casual user.

- graphics screen viewing quality: ps and pdf beat dvi whenever the dvi viewer uses bitmapped previewing. However, provided figure labels are put in a TEX overlay, the bitmaps are usually quite satisfactory.

[12] Some ghostscript-based viewers will soon accept pdf.

- text (and math) screen viewing quality: ps appears to be the loser. On PC and UNIX platforms, ghostscript-based viewers are available. But there is a quality gap since the Type 1 hinting that makes ATM (Adobe Type Manager) font rendering so remarkable is still poorly supported by the viewers. The ghostscript-based viewers are learning to use grayscale blending (known as anti-aliasing) to replace Type 1 hinting.[13] If ATM and Type 1 fonts are available, dvi matches pdf quality, and both provide best quality. If not, pdf is far superior at most magnifications.
- viewer speed: dvi viewers are faster with few exceptions, often dramatically faster. Their bitmapped graphics are also fast. I consider slowness the worst failing of the ps and pdf viewers.
- file size: dvi is the winner: typically only 2K octets per page once compressed. Even when compressed, pdf is bulky because it is currently necessary to include (partially downloaded) Type 1 fonts; 20K octets per page is typical for articles of moderate length.
- public acceptance: Only time will tell; the race has only begun. A big piece of the pie is sure to go to pdf since it is being actively promoted by a powerful corporation aiming at a nascent multi-billion dollar market: the totality of electronic publishing.

The prospects of dvi format

I have an undisguised bias in favor of dvi format for scientific postings, and for some good reasons:

(a) The TeX community is undisputed master of the dvi standard.

(b) dvi format is output by TeX; thus, so long as TeX dominates the composition of scientific documents, dvi format will hopefully be conveniently convertible to all page description formats, including newcomers.[14]

(c) The comparative simplicity of the dvi format makes possible significant developments in return for a moderate effort.

(d) TeX viewer development has been the cutting edge of TeXnology for many years. If pdf format gains a near-monopoly position for browsable postings, I fear there will be a dramatic wilting back of TeX system development, particularly in the public domain and shareware realms. When a viewer falters, the linked TeX user interface also falters. Moreover, a monetary hurdle to electronic science publication may then become a fact of life — one that is, alas, substantial for an individual but negligible for an organization. (Acrobat Distiller costs several hundred dollars, as do commercial TeX systems.)

The future role of the dvi format in electronic publication of science has yet to be decided by programming and subsequent competition. I believe the three developments informally proposed in this article could help TeX's own dvi format to win a role that is viable, and indeed enviable.

References

[1] Electronic Math Journal List. Archive at http://math.albany.edu:8800/hm/emj/; wais index at http://nyjm.albany.edu:8000/SF/emjsearch.html; this list was organized by Mark Steinberger and the author.

[2] Math font discussion list. math-font-discuss@cogs.sussex.ac.uk; see its archives on ftp cogs.sussex.ac.uk; this list was organized by Alan Jeffrey, alanje@cogs.susx.ac.uk.

[3] Knuth, D. *TeX The Program*. Reading, Mass.: Addison Wesley, 1986.

[4] Knappen, Jörg. "Release 1.2 of the dc-fonts: Improvements to the European letters and first release of the text companion symbols." *TUGboat* 16,4 (1995), pages 381 – 387.

[5] Rokicki, Tomas G. "A proposed standard for specials." *TUGboat* 16,4 (1995), pages 395 – 401. [Compare earlier proposals by Don Hosek (1987), and Nelson Beebe (1990), archived on CTAN in dviware/driv-standard/papers/.]

[6] Sendoukas, Hippocrates. *DVIWin*, a dvi viewer and printer driver for MS Windows, exploiting pk fonts. [Available on CTAN in dviware/dviwin/.]

[7] Siebenmann, L. "Occam's Razor and Macro Management." *Proceedings of the Ninth European TeX Conference* (Sept. 4–8, 1995, Papendaal, Netherlands), 317 – 329. [Included in electronic form in the distribution of Occam; see CTAN in macros/generic/ or ftp://matups.math.u-psud.fr/pub/TeX/.]

[8] Tobin, G. *DVL: A DVI Text Language*. 1995. [A portable package found on CTAN in dviware/dtl.]

[9] Y&Y Inc., *DVIWindo*, commercial dvi viewer and printer driver for MS Windows, exploiting Type 1 fonts. See http://www.YandY.com.

[13] In the PC world, a "ps" viewer exploiting anti-aliasing has recently appeared; it is PSVIEW and the contact address is tdieting@iicm.tu-graz.ac.at.

[14] On the other hand, no electronic publisher would dream of discarding the tex source files!

BLUe's Format — the off-off alternative

Kees van der Laan
Hunzeweg 57
9893 PB Garnwerd
The Netherlands
Email: cgl@rc.service.rug.nl

Abstract

BLUe's format is an independent set of TeX macros to assist self-publishing authors with creating, formatting, exchanging and maintaining compuscripts. It comes with a user's guide and TeXnical documentation. The format builds upon manmac, upon functionalities provided in the *TUGboat* styles, and upon experience gained by the AMS in TeX formatting.

Introduction

In this short paper I will account for the history and global aspects of BLUe's format[1] without going into too much detail. BLUe's format was already on its way as I studied LaTeX, AMS-TeX, *TUGboat* styles, Spivak's, and of course Knuth's works, only I did not realize it at the time. I envisioned a series of manuals: Publishing with LaTeX, Publishing with TeX (PWT for short), Publishing with SGML, and Publishing with you-name-it. The idea was that all these guides would essentially treat the same typesetting goals with only the tool exchanged.

At BachoTeX94, where I presented "Manmac BLUes" and concluded by saying "If only there had been a user's guide for manmac, the (LA)TeX world would look different," Bogusław Jackowski challenged me to write such a guide. I replied that I had 2.5D METAFONT on my mind. Once home, however, I thought it over and at the EuroTeX94 in Gdansk that fall I reported about the birth of a BLUe's format which not only built on manmac but also accounted for developments since the debut of manmac.

At EuroTeX95 I spoke about indexing on-the-fly within one pass, and about the database approach to store and reuse formats, tools (add-ons), references, addresses, and pictures. A tool to assist conversion of a script towards other environments also emerged, called BLUe-2-LaTeX — nicknamed BLUe's convertor assistant.

Now, at TUG96 I can look back, while continuing with 2.5D METAFONT, and say that BLUe's format is a personalized alternative to LaTeX, and off-off for the time being. The macros are shareware. The documentation is free for personal use.

Why?

I needed macros to provide computer-assisted handling of information, be it consumption or production and dessimination, and — something which I understand — would serve a lifetime.

Looking back, BLUe's format system is, on the one hand, an answer to such questions as:

- What has Knuth done in electronic publishing?
- Why?
- How?
- How does he use it?
- What can I do with it?

On the other hand, it integrates developments since.[2]

The user is me,[3] but it might be worthwhile for all those authors who:

- practise self-publishing
- choose for English and ASCII
- adhere to TeX formatting
- favor minimal markup
- like a lifetime tool, with goodies such as stability, consistency, simplicity, portability, generality, flexibility, extensibility
- favor an open, documented system
- prefer an extensible (formatting) language
- support the public domain software credo

[1] BLUe stands for Mr. BLUe — my innocent user and relative of Ben Lee User of *The TeXbook* fame (Knuth 1986).

[2] Compare Douglas Adams' "The answer is 42 if only I knew the question."

[3] Well, ...and a few others. A Dutch student reported "The way you describe Knuth's markup makes you feel that this is the natural way to mark up scripts."

However, the style-designer, and hacker, might find it interesting to see how to handle a bibliography in one pass, how to index in one pass, how to code a minimal markup macro on top of a two-part macro with nearly the same functionalities, or how to provide for options without parsing arguments.

What is BLUe's format?

BLUe's format is a self-contained set of macros on top of plain TEX and `manmac`, designed to assist authors with creating, formatting, exchanging and maintaining compuscripts. It comes with a user's guide[4] and TEXnical documentation. These last two have appeared as articles in MAPs, the so-called "Paradigm series" (see references).

The format builds upon `manmac` (Knuth's macros to typeset *The TEXbook* and *The* METAFONT*book*), the functionalities provided by the *TUGboat* styles, and experience gained by the AMS in TEX formatting.

It contains a default note format, as well as a format for reports, letters, and transparencies. The letter format allows mail-merge with addresses provided in a database. It also allows the typesetting of address labels. Graphics can be done via TEX alone by Knuth's boxes of `manmac` or his picture environment subset `gkpmac`,[5] and by some turtle graphics macros of my own. A ToC (Table of Contents) utility assists an author when developing a text. Indexing can be done in one pass, completely within TEX.

Add-ons, formats, references, and pictures are stored in databases, such that you don't have to "pay" for what you don't use, nor do you have to be aware of the physical location. The latter is very convenient when working on different systems. A convertor assistant — blue-2-LATEX — facilitates submitting compuscripts for regular publication.

Usage aspects

Each format has its specific tags. The default note format differs from the letter format, because the task is different. The transparencies format shares the tags with the default but the markup is biased by line-by-line processing, because it centres.

A template for the default note format reads as follows:

```
\input blue.tex \loadindexmacros
                \loadtocmacros
\bluetitle ...
\bluesubtitle ...
\blueissue ...
\bluekeywords ...
\blueabstract ...
\beginscript
\bluehead ...

<copy proper>

\pasteupindex\pasteuptoc
\endscript
```

A `\blue<tag>` takes a blank line as (implicit) terminator, and there are headings for three levels.

The token variables `\author`, `\address`, and `\netaddress` are provided with defaults, so they are not needed in the source file.[6] Bibliographic entries are inserted into the database `lit.dat`; the look-and-feel of typeset references has been borrowed from the AMS. The markup of index reminders is consistent with the approach taken by Knuth for *The TEXbook* and *The* METAFONT*book*.

Special paragraphs. All special paragraphs are coded independently; they can be used with any flavor of TEX. Details are provided in the "Publishing with TEX" user's guide. Some highlights are noted here.

The special paragraphs from `manmac` include `\begindisplay` and `\enddisplay`, `\beginsyntax` and `\endsyntax`, and so on. Items take care of automatic sequencing, be it by numbers or by letters. The end separator is `\smallbreak`.

Verbatims have default ! as escape character. These verbatims are called semi-transparent. File verbatim inclusion goes via `\bluefileverbatim`. Pascal fragments can be marked up by bracketting the fragment by `\beginpascal` end `\endpascal`, without inserting any additional markup into the code — the formatting macros are loaded behind the scenes.

Binary trees can be marked up via `\beginbintree` and `\endbintree`. The specification of data for a table goes via `\data`. Presentation of a table can be varied by using the attributes `\framed` and

[4] A Russian version is in progress.

[5] The macros used to typeset *Concrete Mathematics* (Knuth 1988).

[6] Maybe I should provide these and others in a small file which can be customized more easily. Maybe I can even use `\everyjob` and prompt for the personalized data, with an instruction at the end that `\everyjob` inactivate itself.

`\ruled`. Math markup has been extended by compatible automatic forward cross-referencing. The creation of pictures has been separated from the reuse of pictures. For creation of pictures by TEX alone, Knuth's `manmac` boxes and his subset of the picture environment has been adopted. Additionally, I have developed some turtle graphics macros, which allow, for example, the typesetting of fractals or (rotated) trees, all within TEX alone.

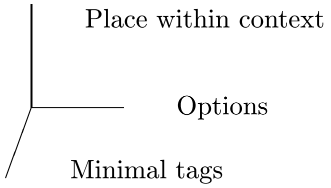
Design considerations

Specifications are needed for:

- look-and-feel in print — typographic design
- markup language — syntax and semantics
- coding style — two-part macros, handling of options, and so on

Typographic design. For the look-and-feel of the result of a blue script in two-column I took over the *TUGboat* layout.

Markup language. I like the orthogonal approach; for example, to treat the following elements: independently from one other.



Conventions for control sequences. The general conventions include:

- `\begin<tag>`, `\end<tag>` pairs, which take global options via `\every<tag>` and local options via `\this<tag>`[7]
- `\<tag>`, the equivalent one-part macro
- `\blue<tag>`, minimal variant
- token variables `\this<tag>` and `\every<tag>` for user guidance (also called options)
- token variables `\pre<tag>` and `\post<tag>` for controlling positioning within context
- token variables to convey information, i.e., the title
- `\pasteup<tag>` to position the formatted text element within its context

More specifically, the following examples show the control sequences for the section heads:

```
\blue<tag>...  \<tag>{...}  \begin<tag>
                               ...
                            \end<tag>
```

optionally preceded by

```
\every<tag>{...}     \this<tag>{...}
```

Where the place within context (see below) differs from the default, supply:

```
\pre<tag>{...}       \post<tag>{...}
```

Options For compuscripts, it is useful to specify whether the latest version is at hand or not. Another option is to specify `\onecol`.[8] With verbatims it is handy to supply catcode changes and to specify a verbatim input file. Pictures can take different values from the default for their size, and some pictures allow visibility levels.

Place within context Formatting can be seen as placing elements within context. To parameterize the place within context is a good approach because it provides the user or style designer with the hooks needed for the look-and-feel of the result. A good example of this is `\head` and such, where the user can set the whitespace via their versions of the commands `\prehead...` and `\posthead...` . The same can be done for verbatims.

For tables the `\postbtable` token variable is already used for measurements of the table. Usually I place the `\vbox` (which results from my `btable` macro) in a math display, which is consistent with the examples in *The TEXbook*.

I'm still considering how to format tables at the beginning of a text and how to paste up these boxes in the right place; this is similar to the list of references or my supporting ToC. The difference with references is that we have usually many tables, and because box registers are limited available, we might end up in trouble. The next best idea is to also have the tables stored in a database and then load them from there selectively, similar to what I already do with figures.

For the moment there is nothing special about figures to be included in the database, as long as they obey the syntax of a database element. For figures I have not yet provide a `\prefigure` or `\postfigure`. The pictures are delivered in an `\hbox`.

Changing the defaults Token variables can be changed in two ways:

- assign new values to the token variables
- extend the token variable with your extras

An example of the latter follows, showing cases of appending to `\post<tag>` and then proceeding to `\pre<tag>`:

```
\post<tag>\ea{\the\post<tag>
              <your extensions>}
%
```

[7] Consistent with Knuth's \every⟨*tag*⟩{...}.

[8] Default is `\twocol` in the note format.

```
\def\preadd{<your extensions>}
\pre<tag>\ea{\ea\preadd\the\pre<tag>}
```

Coding

In this part some ideas behind the coding are supplied. In the Paradigm series of articles, various cases — such as two-part macros and one-part on top, handling of options, coding of items, headings, and so on — were treated.

Conventions. Again, to follow Knuth's lead, especially as done in `manmac`, I have favored the so-called two-part macros at the lowest level.

Starting with the rootname `<tag>`, composite macro names have been created, using prefixes such as `blue`, `begin`, `this`, `every`, `pasteup`, `pre`, `post` and `end`, and postfixes such as `box` and `name`. For example, the user will find `\bluehead`, `\head`, `\beginhead`, `\endhead`, `\prehead` and `\posthead`. However, not all possible combinations exist; for example, `\thishead` is not provided.

For language-dependent names I have introduced definitions of the form `\<tag>name`, for example, which is for use within the environments `keywords`, `abstract` and `references`.[9]

Where available macros have been used, I have added an extra level, to comply with my markup conventions. When the original macro name had to be preserved for the outer level the name was changed, usually by using a prefix pointing to the source.

Beginning of the BLUe script. Design goals include:

- several compuscripts can be processed in one run
- preliminary matter can be provided in any order
- start with a rule in print
- set title in bold
- set authorname indented
- store address information to be set at the end
- set keywords in smaller type
- set abstract 'narrowed' and in smaller type
- set (small) table of contents
- title and issue are used in running headline
- running foot contains draft, page number and copyright

The typesetting of the preliminary matter is done by `\beginscript`.

Headers. Design goals include:

- discourage setting the title alone at the end of a page
- typeset (flexible) vertical space before (large) and after (medium) the title
- gobble spaces at the beginning of the title
- set the title in bold face (and the current size) unindented
- don't indent the first line of the subsequent text

Templates for the coding of heads come from *The TeXbook*'s `\beginsection` and `manmac`'s `\beginchapter`.

The above is only part of the story. In `\report` I also needed to reuse the titles for the Table of Contents, Table of Examples, and the running heads. I could not combine storing and processing on the fly. The best I could achieve was to allow for changes in category codes now and then. This is handy for math material appearing in titles; however, because the circumflex is used for other purposes, its catcode has to be temporarily changed to 7 in titles.

Selective loading. The specifications must:

- load specified parts
- prevent side-effects (work within a group)
- separate typesetting from paste-up
- cope in an elegant way with typos in macro names

At the heart of the coding lies the principle that specified control sequences are redefined. This is in contrast with common usage, which seeks to prevent a format or style file from being overwritten. My use is the opposite: I specify the names I want to have selected via a definition and explicitly require this definition to be overridden.

The list element tag — also called an active list separator — is fundamental. The idea is that this list element tag can be defined such that we can either use its arguments to form a definition — that is, the macro is loaded — or we can just skip them.

One-pass handling of references. The goal is to:

- load only specified references from a database
- store these
- associate the names with their sequence number for cross-referencing
- separate typesetting from paste-up
- cope with typos in the macros names in an elegant way

Various steps have to be integrated to:

- maintain a database of references

[9] This is an aside because I consider English the lingua franca for scientific communication.

- provide a references macro to perform the tasks of selecting and formatting
- specify the list of names as argument to the macro[10]
- load selectively from the database
- format the selected entries
- prevent side-effects
- redefine the names of the selected entries by their number and store these globally
- provide a paste-up tag

I have made use of the FIFO (first in first out) paradigm, so an unspecified number of entries can be supplied.

Practicalities

When a compuscript is in the proofing phase we have to get rid of overfull and underfull `\hboxes` and `\vboxes`. A pragmatic approach has been summarized lucidly by Phil Taylor (1993): overfull `\hboxes` usually result from words which extend beyond the righthand margin.

Get rid of overfull `\hboxes`. In writing in (LA)TEX about (LA)TEX — especially in narrow columns — long control sequence names often yield problems. Generally, the hyphenation algorithm is not in action for such 'words.' Let us assume that these words are set in `\tentt`, then we can activate hyphenation and allow for some room via:

```
\hyphenchar\tentt='055
%and room for example via
\tolerance500
\hbadness=499
\hfuzz=5pt
```

Get rid of underfull `\hboxes` and `\vboxes`. These generally require adjustments to the text itself. Underfull `\vboxes` are usually the result of boxes which don't fit on the page and are therefore moved to the next page or column, leaving an underfull column or page behind them. We can decide to let these elements float via:

```
\topinsert%or \midinsert
<box material>
\endinsert
```

or otherwise adjust or rewrite the text. As yet, no `\raggedbottom` is provided.

What is not included?

No special hyphenations or font selection in the spirit of NFSS has been included, nor has the use of encapsulated PostScript been treated in the "Publishing with TEX" user's guide. As well, there are no macros to handle tables larger than the page size.[11] Of course, this list is by its very nature incomplete, but it rounds out the description of things one wants to know.

Related work

First a disclaimer. It is not possible to really compare `blue.tex` with other formats because each has its own goals. What follows has to be read as a rough indication of how `blue.tex` relates to the works mentioned.

In a sense LATEX provides the functionalities I'm after; however, LATEX is too complex to master for me, and I don't like its coding, or its appearance in print using the standard styles. LATEX aims at a general audience, while BLUe's format is customized for my own purposes. Because of LATEX's complexity, customization is too time-consuming. Moreover, it is subject to change — it is only quasi-static — and from the results I have seen so far, its complexity has increased, alas, and the end to changes not in sight. LATEX does not worry about adaptations to other formats because people involved with it consider LATEX a (de facto) standard. The concept of active documents is not in there, apart from what Knuth has already provided, albeit under different names.

A quick survey of other macro sets revealed that each has its own deficiencies, and none included all the functionalities I was looking for. Doob's macros for typesetting his "Gentle Introduction to TEX" (1990) are too limited, especially when we consider the life-cycle of documents. Berry's `eplain` builds upon `plain` alone. Spivak provides much of LATEX's functionalities in LAMS-TEX, but he avoids `plain`'s math markup. The infotex macros of the Free Software Foundation look as if they had been designed bottom-up. What is the use of a meta-parsing macro — very clever, there is no question about that — if we can do without this functionality in ordinary formats? No attempt for simplicity, nor a set of common markup tags as a foundation to build upon.

The functionality of *TUGboat*'s output routine lies at the heart of the page makeup of `blue.tex`. However `blue.tex` goes much further than the *TUGboat* styles in, for example, the markup for pictures, references, (bordered) tables, verbatims,

[10] It is used twice: for selecting and for formatting.

[11] It is not hard to provide a row separator which would allow splitting of the table, I guess.

math cross-referencing, indexing on-the-fly, all designed for use with AnyTEX; that is, it is not tied up with `plain` exclusively.[12] Add to that the way references and pictures are selected from a database and one would have to agree that `blue.tex` is a leap forward. The coding of options is less monolithic, and the setup via modules with thin interfaces more flexible. `blue.tex` is not a goal per se. It has an eye open to change in general, and adaptation towards other contexts in particular.

Furthermore, as far as I'm aware of, nobody has paid sufficient attention to how to cope with the balance of a *personalized* system in relation to the outer world: using a stable format as basis, adaptable to requirements from the world outside when the need arises.

Availability

`blue.tex` comprises some 118KBytes, `fmt.dat` is 40K, `tools.dat` is 60K, `lit.dat` takes 78K, and `pic.dat` another 38K; `address.dat` is still very small. The transparencies macros have been incorporated into the database `fmt.dat`, and, as of March 1995, they are together with the letter and report formats. Although this paper bears the title "BLUe's Format," the filenames are in lowercase letters only: `blue.tex`, `fmt.dat`, `tools.dat`, `lit.dat`, `pic.dat`, `address.dat`, and the article `fmt.art`, in addition to the PWT user's guide. All files available[13] on the NTG's 4AllTEX (double) CD-ROM from 1995, CTAN (in `info/pwt`, and from the NTG and GUST fileservers.

Looking back

The hardest thing has been to not mix up all the influences I have been exposed to. To stand back and decide about simple coding conventions was not easy, especially when the material has not yet been mastered completely. When to use token variables and when to use definitions is also difficult when trying to prevent confusion at the user level. It is tempting to introduce parameter separators, but I have refrained from those in the outer-level macros, and have instead used the straightforward parameter mechanism. Having gained some practice with the `\this<tag>` idea, I am still very pleased with it, especially because of its local character and absence of parsing overhead.

Looking forward

BLUe's format is stable. I use it whenever I have to typeset notes, letters, submisions for TEX bulletins, or transparencies. I guess it will grow along with my needs. The "Publishing with TEX" user's guide is also stable, although now and then some parts get improved. For some time to come, I'll be working on METAFONT/MetaPost. After that I'll embark on Literare Programming — who knows how that will influence BLUe's format.

Acknowledgements

I owe the TEX community much for borrowing free material to build upon, starting with what Don Knuth gave to the world. In particular I would like to thank the NTG for allowing me to bring out earlier versions of my papers in the MAPs publications; I would also like to thank GUST for having invited me to lecture at BachoTEX94 about `manmac`, and thank the attendees for having asked for a `manmac` user's guide. Thank you!

Conclusions

With respect to format or style development, macro writing is a special way of programming; to embed this within the realm of software engineering is a real challenge, and much needed.

It is still wishful thinking to hope for a full-fledged simple format which is easy to use, and whose marked up copy can be easily transformed into a representation suitable for other situations. `blue.tex` comes close, mainly because I have thought more than twice about what Knuth has provided, before writing anything of my own, and because of my handful of outer-markup tags (very few indeed). Therefore, I conclude that:

– a variety of tools is needed

– `blue.tex`≡ `manmac` + *TUGboat* abbreviations and output routine + variant formats + add-ons
  variant formats ≡ report + letter + transparencies
  add-ons ≡ bordered table + crossrefs + `gkppic` + turtle graphics + verbatim suite + index set + . . .

[12] For example, I have made the index macros separately available to cooperate with AnyTEX.

[13] Macros are shareware. The price is the minimum of US$25 and the local price of 20 loaves of bread, in dollars. CyrTUG members can join for free.

- the principles include consistency, portability, longevity, flexibility, intelligibility, extensibility, and correctness
- user documentation is paramount
- when attempting to keep up with Knuth, be realistic

When you consider book formatting you might consider customizing `manmac`.

My added value. I have integrated `manmac` and developments since, such as:

- separation of specification and paste-up
- specification of options via `\this<tag>{...}` and `\every<tag>{...}`
- parameterizations of `\pre<tag>{...}` and `\post<tag>{...}`
- formats, tools, addresses, literature and graphics databases
- one-pass bibliography handling
- formatting references in the spirit of the AMS
- cross-referencing of math and bibliography items
- outer and inner markup separation
- a verbatim suite
- a bordered table macro
- a conversion tool for outer markup tags
- report, letter and transparencies formats, in addition to the default note
- one- and two-column options for notes
- one-pass indexing on-the-fly
- ...various other add-ons

Last but not least, a user's guide and TEXnical documentation are part of BLUe's format system.

With respect to coding I have added the paradigm — a real pearl, to paraphrase Bentley[14] — to systematically and automatically add a one-part macro with the same functionalities on top of a two-part macro. My thesis is that the users can benefit from `blue.tex` to typeset like craftsmen, to achieve the quality of Knuth.

The extra bonus is stability, and simplicity as well. The disadavantage is investment in learning which, by trial-and-error, takes a substantial amount of time. It all has to do with your attitude, whether you believe that TEX proper will serve a lifetime, and whether you like to invest in learning plain and `manmac` as a basis. The reward is freedom. You are no longer dependent upon the gurus for what-and-when. Furthermore, I adhere to De Vinne's adage that "The last thing to learn is simplicity."

The user is encouraged to peruse *The TEXbook*, and to also look at the markup in the file. I hope you will arrive at the same conclusion as I have: Knuth's markup is unsurpassed. Then simply practise Knuth's minimal markup, have fun and all the best.

References

Adams, D. *The Hitchhiker's Guide to the Galaxy*. New York, Harmony Books, 1989.

Berry, K. `eplain`. Available from CTAN(`/macros /eplain`).

De Vinne, T.L. *Invention of Printing*. 1876. New York: Hart, 1969.

Doob, M. *A Gentle Introduction to TEX*. *TEXniques* 13, Providence, Rhode Island: TEX Users Group, 1990.

Knuth, D.E. *The TEXbook*. Reading, MA: Addison-Wesley, 1986.

Graham R.L, D.E. Knuth, O. Pastashnik. *Concrete Mathematics*. Reading, Mass.: Addison-Wesley, 1988.

Spivak, M.D *LAMS-TEX, The Synthesis*. Texplorator, 1989.

Taylor, P. "A Pragmatic Approach to Paragraphs." *TUGboat*, 14,2 (1993), pages 138–140.

van der Laan, K. "Paradigms — Headache?" MAPS 94.2 (1994), pages 212 – 214.

van der Laan, K. "Paradigms — Plain's Items Extended." MAPS 94.2 (1994), pages 210 – 211.

van der Laan, K. "Paradigms — Two-Part Macros." MAPS 95.1 (1995), pages 200 – 204.

van der Laan, K. Publishing with TEX: BLUe's Selection. Garnwerd, Holland, 1995. Available from CTAN (`info/pwt`).

[14] Jon Bentley writes a regular column, "Programming Pearls", for the *Communications of the ACM.*

Turtle Graphics and TEX — a child can do it

Kees van der Laan
Hunzeweg 57
9893 PB Garnwerd
The Netherlands
Email: cgl@rc.service.rug.nl

Abstract

Papert's Turtle Graphics in TEX provide the user with a new method for handling drawings via TEX alone. Being aware of explicit coordinates is replaced by the body language of drawing using the points of a compass. The approach is suited for highly systematic figures such as fractals. Example shapes include spiral, Pythagorean tree, binary tree, and rotated binary tree. The macros are part of BLUe's format system and available from CTAN and the NTG's 4AllTEX CD-ROM.

Introduction

Turtle Graphics has its roots in the pedagogical approach of Piaget. It comes down to learning by metaphors. Computer graphics are demonstrated to children via a turtle[1] moving on the screen. A petal can be drawn by starting at the origin and moving north towards 'up + right', arriving horizontally; then leaving southbound and arriving horizontally at the origin.

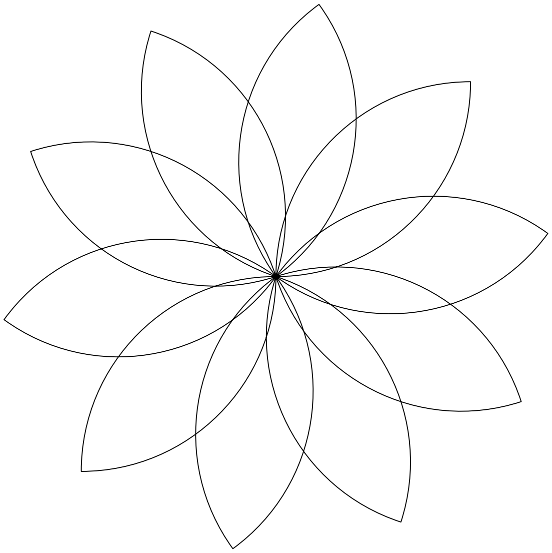
Example *(Flower borrowed from Papert)*

The flower picture is obtained via first creating a basic petal by moving in quarter circles, and then combining several of them. The turtle moves along rotated petals. In METAFONT this is coded essentially as follows:

```
petal=origin{up}..
     {right}(up+right){down}..
     {left}origin;
for k=1 upto 10:
   draw petal rotated36k;
endfor
```

petal is a path; the path data structure in METAFONT is powerful.

In this short paper I will discuss what has been used in BLUe's format system[2] as an extension to manmac in the turtle graphics macros, especially in the coding of the points of a compass: \N, \E, \S, \W, \NE, \SE, \SW, \NW, along with \ESE, and \WSW. I will restrict myself to straight lines in TEX.



Papert's petals

Examples are included which show what can be attained by these basic functionalities. Now and then a METAFONT alternative has been included. In the Appendix some tree diversions have been given.

Why?

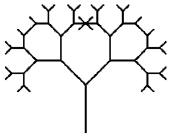
The need for general and flexible line elements arose when I faced the problem of how to draw classical fractals in TEX.[3] The very least is the possibility to draw lines at 45°, the mid-points of a compass.

[1] Knuth already used the turtle idea in his dragon figures (1996, p. 391). For those interested in turtle graphics, consult Papert (1980), for example.

[2] For more information, see my paper, "BLUe's Format — the off-off alternative," elsewhere in these proceedings.

[3] Inspired by Gurari's work on TEX and graphics.

Example *(Pythagorean tree)*



How to do this in TeX? Via LaTeX's picture environment? Too clumsy, and cumbersome when changing the order, for example. Via turtle graphics? Definitely. Via PostScript? A possibility.[4] Via METAFONT? Definitely.

However, why not see how far we can get via TeX alone?

And what about the relevancy? I'm very pleased by the spin-off how to typeset binary trees or charts, even rotated, without the use of PostScript. See the Appendix.[5]

What we need is not the Cartesian picture environment approach, but the good old pen-plotter TeXniques, better known in the pedagogical world as Turtle Graphics.

What is the problem?

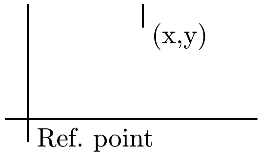
TeX's `\hrule` and `\vrule` primitives are gems and very powerful. It would be nice to have similar primitives for any direction. In the absence of these we can use line pieces provided in fonts. However, the latter suffer from the following drawbacks:

- for a few discrete directions only
- line lengths are discrete too
- line thickness is inflexible

Turtle graphics

The basic idea is that a turtle moves on the screen with the drawing as its trace. How to implement this in TeX?

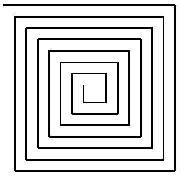
The position of the turtle is maintained in the dimen variables `\x` and `\y`, with TeX's *reference point* left invariant. Moving is parameterized by a direction and by how far to go in that direction. The accompanying figure shows the effect of `\N1`, that is draw the line (`\x`, `\y`)–(`\x`, `\y`+1) — in turtle language the turtle moves up. After completion `\y` has been increased by `\unitlength`.



The movements — our first steps in the turtle graphics world — can be achieved by the following control sequences:

- `\N`, `\E`, `\S`, `\W` mean draw north, east, south and west; similarly, `\NE`, `\SE`, `\SW`, and `\NW`
- `\whiteN`, `\whiteE`, `\whiteS`, `\whiteW` mean *white*-draw north, east, south and west, i.e., the turtle just moves[6]

Example *(Spiral)*
To get the flavor, a classical picture and its coding has been provided, which illustrates that we don't have to worry about coordinates.



The markup reads essentially as follows:

```
\unitlength... \k=1;
\loop\E{\the\k}\advance\k+1
     \S{\the\k}\advance\k+1
     \W{\the\k}\advance\k+1
     \N{\the\k}\advance\k+1
\ifnum\k<29 \repeat
```

More examples have been included in the graphics chapter of the PWT user's guide.

The winds and halfwinds

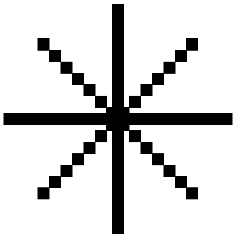
The idea is to compose lines out of elements. I used squares and/or rectangles as elements, and tiled them as follows:[7] For my own reasons, I have also chosen to speak of 'winds' and 'halfwinds' rather than of the points on a compass.

[4] Joseph Romanovsky transcribed my (recursive) code into PostScript.

[5] Or my 'Publishing with TeX' user's guide, PWT for short.

[6] The midpoints can be composed from the four main compass points in this case.

[7] In order to make it visible `\linethickness` has been set to `1ex`. Experiments with bullets and LaTeX's line fonts did not yield pleasing results.



The turtle moves `10ex` in each direction, to be tiled by `\hlfwndelm` in the halfwind directions. What is essential is how lines leave a mathematical point. The accompanying model picture has been drawn as follows:

```
\let\0\N \let\1\NE \let\2\E \let\3\SE
\let\4\S \let\5\SW \let\6\W \let\7\NW
\linethickness1ex
\setbox\hlfwndelm=\hbox{\vrule
   width\the\linethickness
   height\the\linethickness depth0pt}
\unitlength10ex
\def\draw{\csname\the\dir\endcsname1}
$$\loop
  \ifnum\dir<8{\draw}\advance\dir1
  \repeat$$
```

Pondering aloud

Can we attain compatibility with TeX's rules primitives? I don't think so, alas.

Thickness. What is meant by thickness if we overlap instead of tile? What is the perceived blackness?

I assumed that tiling with square elements of size `\linethickness`×`\linethickness` — as in the example figure — would yield the same blackness as a rule of thickness `\linethickness`.

Size. Usually the size along the x-axis must be provided. I prefer to have the *real size* specified independently from the orientation of the line. However, the resulting size is not necessarily #1×`\unitlength`.[8] In general the result differs at most by half the atom size because it is composed of a multiple of the basic element. We have to correct by $\sqrt{2}$ to compensate for the direction as we pace along one of the axes. In the example the required length is `10ex`, with as result the tiling of 7 elements of size `1ex`.

For large lines one could think of combining the line elements in LaTeX's `line10` font with the smaller elements. I refrained for two reasons: first, LaTeX's NW line element — `\char'145` — did not fit exactly in the box; and second, because of the inflexibility of the thickness of the font entries.

Design specs

With the above in mind, I specified the following for the microscopic level — the wind and halfwind commands proper — and for the macroscopic level — the placement within context.

Microscopics. The functionality is to draw a line of the specified length in the direction as implied by the control sequence name. The general specifications read as follows:

- as argument a 'factor' is expected, in order to yield the required length #1×`\unitlength`[9]
- `\linethickness` is a parameter
- after drawing, the position of the turtle is at the end of each line, the reference point has been left invariant, and all the boxes have zero width, height and depth

Extra for the four halfwinds the following:

- `\hlfwndelm` and `\linethickness` are parameters
- draw a line of approximately the specified size
- the atoms are tiled diagonally, at the corners

Macroscopics. Placement within context is the concern of the user. However, because of the zero dimensions of the boxes it is a nuisance to skip or kern when using a picture, in order to create the open space, the niche for the picture. Moreover, when the picture does not take dimensions we are in trouble at page breaks. Therefore assistance is badly needed. The picture environment idea combined with databases comes to the rescue. The *use* of prefab pictures has been simplified in this way, while there is flexibility via `\thispicture` to override the defaults.

Pictures can be stored in BLUe's format `pic.dat` database. Within each database entry the default bounding box and placement within context is provided for. Through the use of `\everypicture` and/or `\thispicture` the defaults can be overridden. This approach complies with the general principles adopted in BLUe's format system.

Coding the winds and halfwinds

It must be emphasized that all boxes have zero dimensions. I also decided to separate getting at

[8] To put it another way: the required length must be a multiple of the atom size.

[9] The idea is that not only can integer values be specified but decimal fractions as well.

the (x, y) position from putting whatever there. This is much in the spirit of the second `\point` macro in Knuth (1986, p. 389) and adheres to the *separation of concerns* adage.[10]

In TEX. Familiarity with TEX's *boxes of size zero* is essential: to know the effect of `\kern`-s and `\h/vss`-s inside, and to know the effect of combinations of these boxes.

Kern-s and stretch-or-shrink-s in boxes of size zero.

Example *(Effects of boxes of size zero)*

```
\newdimen\x \x=4ex
\newdimen\y \y=2ex
.\hbox to 0pt{\kern\x a\hss}.
\kern30ex
.\kern\x a.

\noindent and

.\hbox to0pt{\kern\x\vbox to0pt
   {\vss\hbox{a}\kern\y}\hss}.
\kern30ex
.\kern\x\raise\y\hbox{a}.
```

with result

```
    ..   a                       .   a.
```

and

```
         a                           a
    ..                           .    .
```

By this mechanism we can move to any point on the page and put there what we wish. Essential is that when a box of zero width is set the *reference point is left invariant* — it is the same before and after.

Putting it together In vertical mode the `\hbox`-es are aligned on the reference point, and when the heights and depths are zero the `\hbox`-es overprint. Moreover, the order of specification is immaterial. In (restricted) horizontal mode `\hbox`-es of width zero overprint and can be given in any order (Knuth 1986, p. 389). In math mode the zero-sized boxes overprint. In display math the invariant reference point is centered horizontally.

Coding After completion the dimension variables `\x` and `\y` have the values of the coordinates of the end of the line. The coding of a few directions has been included to convey the idea.

```
\newbox\hlfwndelm
\newdimen\auxdim %linesize
\newdimen\linethickness
\linethickness1ex
%
\def\xy#1{%Function: place #1 at x, y
   \vbox to0pt{\vss
   \hbox to0pt{\kern\x#1\hss}\kern\y}}
%
\def\xytxt#1{%Function: place text #1
             %          at x, y
   \xy{\vbox to0pt{\vss
   \hbox to0pt{\strut#1\hss
               }\kern0pt}}}
%
\def\N#1{\xy{\kern-.5\linethickness
  \vbox to0pt{\vss
  \hrule height#1\unitlength
   width\linethickness}}%
\advance\y#1\unitlength}
%
\def\S#1{\advance\y-#1\unitlength
         {\N{#1}}}
%
\def\SW#1{\auxdim#1\unitlength
          \correction%sqrt2
\loop\advance\auxdim-\wd\hlfwndelm
\ifdim\auxdim>-.5\wd\hlfwndelm
   \advance\x-\wd\hlfwndelm
   \advance\y-\ht\hlfwndelm
   \xy{\vbox to0pt{\vss
           \copy\hlfwndelm}}%
\repeat}
```

In METAFONT. The coding to go one step north in METAFONT reads as follows:

```
def north=draw z--
    hide(y:=y+size)z enddef;
```

To draw in any direction in METAFONT is implicit, just provide:

```
draw <beginpoint>--<endpoint>
```

There isn't much need to provide turtle graphics macros in METAFONT — it is essentially already there.

Coding the Pythagorean tree

The following illustrates the use of basic turtle movements for this class of problems. Moreover, it shows that coding in TEX is completely different from coding in METAFONT. This goes deeper than a mere difference in notation.

[10] In my view, it makes the code more trustworthy and avoids pitfalls, especially the potential confusion between the kerns needed to get at (x, y) and the kerns to position what has to be put at (x, y).

In TeX. Via the use of the winds and halfwinds the Pythagorean tree code in TeX reads as follows:

```
\def\pythtree{\ifnum\level=1
                    \eerthtyp\fi
    \advance\level-1
    \multiply\kk23\divide\kk32%
    {\leftbranch\draw\pythtree}%
     \rightbranch\draw\pythtree}
\def\eerthtyp#1\pythtree{\fi}
%with auxiliaries
\let\0\N \let\1\NE \let\2\E
\let\3\SE\let\4\S \let\5\SW
\let\6\W \let\7\NW
\def\leftbranch{\advance\dir7
 \ifnum\dir>7 \advance\dir-8 \fi}
\def\rightbranch{\advance\dir1
 \ifnum\dir>7 \advance\dir-8 \fi}
\def\draw{\csname\the\dir\endcsname
         {\the\kk}}
%with use
$$\unitlength0.1pt\kk128 %Size
                  \level5%Order
  \N{\the\kk}            %Trunk
  \pythtree$$
```

Note that there is no build-up of `\fi`-s, no use of either `\expandafter` or `\let` (this last has been used throughout *The TeXbook*).

In METAFONT. The turtle idea has been used in going from node to node in METAFONT as follows:[11]

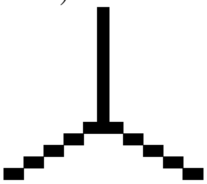
```
pair node[];
n=15;          %order
l=75;          %size of the trunk
node[0]=origin;%position, and
d= 90;         %orientation trunk
%Create nodes of leftbound branch
for k=1 upto n:
   node[k]=node[k-1]+l*dir d;
   d:=d+45;l:=.7l;
endfor
%Draw the tree
for k=n-1 downto 1:
  draw node[k+1]--node[k];
  addto currentpicture also
  currentpicture rotatedaround
                 (node[k],-90);
endfor
draw node1--node0;
drawdot origin; showit
end
```

Note that there is no recursion. The symmetry operations of METAFONT allow a concise implementation, with much faster performance than when all the leaves would have been walked through one after another.[12]

Trinaries

For 45° lines I used square elements. Why not use rectangular elements for 30° lines in conformance with the direction?

Example *(Lines at 30°)*



This model has been obtained as follows:

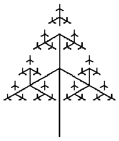
```
$$\x0pt\y0pt
  {\N{10}}{\ESE{10}}{\WSW{10}}$$
%with initializations
\linethickness1ex
\setbox\trielm=\hbox{\vrule
   width1.74\linethickness
   height\linethickness\relax}
%To account for element in 30 degrees
%direction
\unitlengthy\ht\trielm %default.2pt
\unitlengthx\wd\trielm %default.3482pt
\unitlength\unitlengthy%default.2pt
%and the macros
\def\WSW#1{\auxdim#1\unitlength
           \divide\auxdim2
   \loop\advance\auxdim-\unitlengthy
   \ifdim\auxdim>-.5\unitlengthy
      \advance\x-\unitlengthx
      \advance\y-\unitlengthy
      \xy{\vbox to0pt{\vss
                      \copy\trielm}}%
   \repeat}
%
\def\ESE#1{\auxdim#1\unitlength
           \divide\auxdim2
   \loop\advance\auxdim-\unitlengthy
   \ifdim\auxdim>-.5\unitlengthy
      \advance\y-\unitlengthy
      \xy{\vbox to0pt{\vss
                      \copy\trielm}}%
```

[11] But ...only for the left branch; to draw the other branches the symmetry operation — `rotatedaround` — has been used.

[12] In the seventies these kinds of problems had a reputation of keeping pen-plotters busy. Because of raster devices we can now do much better, and METAFONT allows us to prescribe this.

```
        \advance\x\unitlengthx
      \repeat}
```

Example *(Trinary tree)*



```
$$\x0pt\y0pt\level6 \kk128
  \N{128}\tritree$$
%with trinary tree macro
\def\tritree{\ifnum1=\level
                     \eertirt\fi
    \advance\level-1 \divide\kk2
    {\N{\the\kk}\tritree}%
    {\ESE{\the\kk}\tritree}%
     \WSW{\the\kk}\tritree}
\def\eertirt#1\tritree{\fi}
```

Remark: The `\unitlength`-s are, by default, equal to the sides of the elementary rectangular block. The size of the tree can be controlled by `\kk`.

Coding a database element

When inserting a picture in BLUe's format picture database, extra layers are added to the picture code to allow for reuse and to parameterize scalability, positioning, visibility, with defaults provided, and to set a picture within a box of the right size, the bounding box.

How to create a database element has been treated elsewhere and is not repeated here. However, I have included an example to convey the idea.

Example *(The database element* `bintreepic`*)*
The `\bintreepic` element of the `pic.dat` database reads as follows:

```
\lst\bintreepic{\bgroup
   \unitlength.5ex\kk32
   \xoffset{-32} \yoffset{-2}%
   \xdim{66}\ydim{5}%
   \def\eertnib##1\bintree{\fi}
   \beginpicture\bintree\endpicture
   \egroup\thispicture{}}
%with in the kernel blue.tex
\def\bintree{\S1\ifnum\kk=2
                 \eertnib\fi
   \divide\kk2
   {\W{\the\kk}\bintree}%
    \E{\the\kk}\bintree}
%and accounting for the leaves
\def\eertnib#1\bintree{\fi
   \global\advance\k1
   \whiteS1\xytxt{
   \csname\the\k\endcsname}}
```

Explanation: `\bintreepic` comes down to an invocation of `\bintree` with scaling and positioning parameters added, assigned with default values. The defaults can be overridden via the use of `\thispicture{...}`. The tokens provided by the latter are inserted by `\beginpicture`.

Epilogue

The lines at 45° have little compatibility with TEX's rules, alas, especially with non-neglible thickness. I was surprised to realize that TEX's defaults for rules are not symmetric around their axes in relation to the reference point.

Have fun, and all the best.

References

Gurari, Eitan M. *TEX and LATEX: Drawing and Literate Programming*. New York: McGraw-Hill, 1994.

Knuth, D.E. *The TEXbook*. Reading, MA: Addison-Wesley, 1986.

Papert, S. *Mindstorms; Children, Computers, and Powerful Ideas*. New York: Basic Books, 1980.

van der Laan, K. Publishing with TEX: BLUe's Selection. Garnwerd, Holland, 1995. Available via CTAN (`info/pwt`).

van der Laan, K. "BLUe's Format — the off-off alternative." Elsewhere in these *Proceedings*.

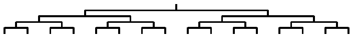
Appendix: Binary tree and chart

Example *(Binary tree)*

```
\pictures\bintreepic

$$\bintreepic$$
```

with result



Rotated tree. Once we understand turtle graphics, rotating a tree can be done easily by shifting the meaning of the directions, and adjusting the positioning of the leaves.[13] In the code below, `\bintree` and `\eertnib` come with `blue.tex`, and `\rotatedbintreepic` is included in `pic.dat`. The `\rotatedbintreepic` entry reads as follows:

```
\lst\rotatedbintreepic{%
\bgroup\unitlength1ex%
  \let\W\N \let\exchange\E
  \let\E\S \let\S\exchange
  \def\1{x}\def\2{y}\def\3{a}
  \def\4{b}\def\5{piet}%
  \def\6{hans}\def\7{etc.}%
  \k0\kk16\xdim{10}\ydim{30}%
\beginpicture\bintree\endpicture
\egroup\thispicture{}}
```

Example *(Rotated tree)*

```
\thispicture{\def\1{cgl}
  \def\2{PWT}\def\3{July}
  \def\4{1995}\def\5{\dots}
  \def\6{}\def\7{}
  \yoffset{-16}\ydim{28}}
$$\rotatedbintreepic$$
```

yields

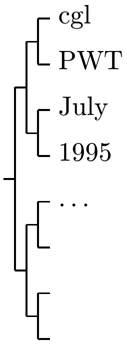
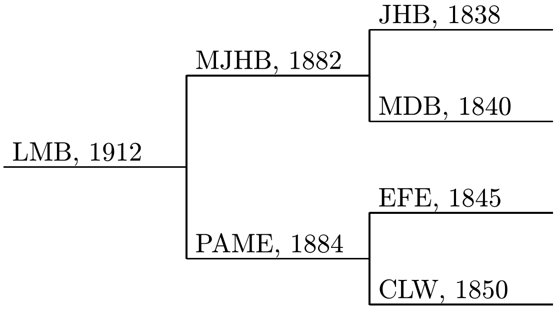


Chart. Through the `\bintree` macro we can also obtain charts elegantly.

Example *(Chart – The TEXbook, p. 248, ex. 22.14)*



obtained via

```
%labels in preorder
%(default in \chartpic)
\def\1{LMB, 1912}
\def\2{MJHB, 1882}\def\5{PAME, 1884}
\def\3{JHB, 1838} \def\4{MDB, 1840}
\def\6{EFE, 1845} \def\7{CLW, 1850}
\ekk8
\k0\unitlength2ex\x0pt\y0pt\kk8
\hbox{\modbintree}
%with auxiliaries
\let\Eold\E
\def\E{\global\advance\k1
  \xytxt{
  \csname\the\k\endcsname}\Eold}
```

Remarks: An aid in finding the numbers of the branches is to delete `\csname` and `\endcsname` in `\E`. The way of traversal at hand is called preorder.

When using `\chartpic` from `pic.dat` the texts along the branches — `\def\1{...} etc.` — have to be supplied as tokens within a `\thispicture`. And one final note: `\modbintree` is the adjusted `\bintree` macro for this case.

[13] A white lie. The tree is actually mirrored because I like the leaves to be numbered from the top. In general we can rotate via PostScript.

Some Useful Macros Which Extend the LaTeX `picture` Environment

A.S. Berdnikov, O.A. Grineva and S.B. Turtia
Institute of Analytical Instrumentation
Rizsky pr. 26, 198103 St.Petersburg, Russia
`berd@ianin.spb.su,olga@ianin.spb.su,turtia@ianin.spb.su`

Abstract

The ability to create pictures using TeX/LaTeX is relatively poor, and many extensions – (`epic/eepic`, `pictex`, `drawtex`, `xypic`, `mfpic`, etc.) – were created to extend this capability to a higher level. The package PMGRAPH.STY (*poor-man-graphics*) which is described herein is not as general as the ones mentioned above. Not being too complicated, these macros appear to be useful in our work, and it seems that they can be also useful for other TeX-users.

The pmgraph.sty Package

This package is based on the use of the pseudo-graphical fonts which are used by generic LaTeX without additional extensions — mainly because the variations of PiCTeX, METAFONT and new graphical font themes are already explored by other authors on a sufficiently higher level. To some extent the purpose of our work was to see how far it was possible to develop new useful graphical primitives for LaTeX *without* the investment of the external graphical tools.

The package PMGRAPH.STY offers the following features:

- vectors with a set of slopes which are as general as the line slopes are implemented in LaTeX;
- vectors with an arrow at the beginning, middle or end of the vector with various orientations of the arrow;
- circles and circular arcs with almost arbitrary diameters using magnified `circle` and `circlew` LaTeX fonts;
- 1/4 circular arcs correctly positioned at the center or at the corner;
- an extended set of frames, which includes various corner styles and optional multiple frame shadows with a variety of styles;
- tools which enable the user to extend the variety of the frame styles and the shadow styles as far as his/her imagination allows it; and
- automatic calculation of the picture size relative to the current width of the text — this includes `picture` environments inside list environments.

Vectors

The number of angles for inclined lines which can be used in LaTeX is limited to a great extent, but the number of angles for *vectors* is even more restricted. The variety of vectors can be increased if, instead of the *strictly* inclined arrows at the end of the inclined lines, arrows with an *approximate* inclination are added. Corresponding changes are incorporated in PMGRAPH where the relation between strict inclinations and approximate inclinations are shown in Table 1. The implementation required the modification of internal LaTeX commands such as `\@svector`, `\@getlarrow`, `\@getrarrow` and the user command `\vector` itself. As a result, the command `\vector` draws vectors for all inclinations valid for LaTeX lines as is shown in Fig. 1. The vectors are not as ideal as normally required by TeX standards, but the results are acceptable for all inclinations, except $(6,1)$.

$(1,1)$	$(1,1)$	$(4,1)$	$(4,1)$	$(5,3)$	$(3,2)$
$(2,1)$	$(2,1)$	$(4,3)$	$(4,3)$	$(5,4)$	$(4,3)$
$(3,1)$	$(3,1)$	$(5,1)$	$(4,1)$	$(6,1)$	$(4,1)$
$(3,2)$	$(3,2)$	$(5,2)$	$(3,1)$	$(6,5)$	$(4,3)$

Table 1: Relation between the line slopes and the approximate vector slopes

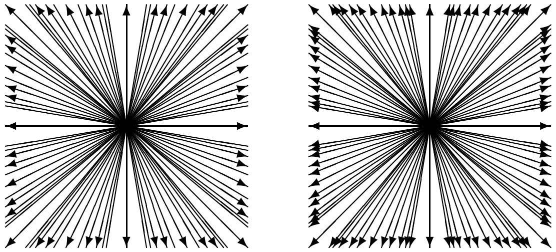


Figure 1: LaTeX and PMGRAPH vectors

LaTeX allows one to put an arrow only at the end of the vector. The `\Vector` command offers the capability of placing *arbitrary* arrows with different orientations along the vector (see Fig. 2). The predefined arrow styles assign a letter to each position and orientation of the arrow along the `\Vector`. The arrows shown in Fig. 2 are drawn

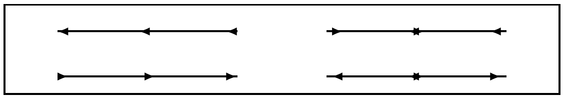


Figure 2: Multi-arrow vectors

by the commands

```
\begin{picture}(300,40)
  \put(20,5){\Vector[bme](1,0){100}}
  \put(20,30){\Vector[BME](1,0){100}}
  \put(170,5){\Vector[xmMZ](1,0){100}}
  \put(170,30){\Vector[XmMz](1,0){100}}
  . . . . . . . .
```

The letter `e` corresponds to an arrow with a 'normal' orientation at the end of the vector; `E` corresponds to an arrow with a reverse orientation. The letters `b` and `B` correspond to arrows at the beginning of the vector (with normal and reverse orientation); the letter `m` and `M` — to the arrows at the middle, etc. The optional parameter of the command `\Vector` contains a list of letters which describes the set of arrows along the `\Vector`. It is possible to create user-defined styles of arrows using the commands `\VectorStyle` and `\VectorShiftStyle` as described in the PMGRAPH documentation.

Circles

The range of the diameters for circles and disks (black circular blobs) available in LaTeX is very restricted. It can be enlarged by using magnified versions of the pseudo-graphical LaTeX fonts if the user does not have something better at his/her disposal such as `curves.sty`, PiCTeX or MFPiC. The disadvantage of the method presented here is that the width of the lines is magnified too, which is inconsistent with the rigorous TeX accuracy requirements, but for *poor-man-graphics* these circles can be satisfactory.

The scaling of circular fonts is performed by the commands

```
    \scaledcircle{factor}
    \magcircle{magstep}
```

which correspond to the TeX commands

```
    \font ...  scaled factor
    \font ...  scaled \magstep magstep
```

The valid *magstep* values are `0`, `h`, `1`, `2`, `3`, `4`, `5`. The values *factor=1000* and *magstep=0* correspond to the one-to-one magnification. The circle magnification, like other TeX commands, returns to its previous value outside the group inside which it was changed.

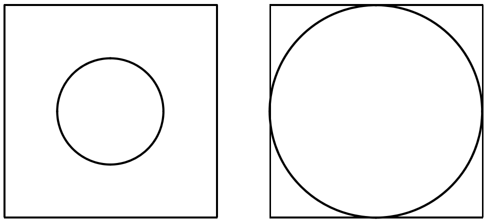


Figure 3: Magnified circles

In order to properly calculate the character code of the circle segment needed to build a circle, taking the magnification into account, it was necessary to redefine some more internal LaTeX commands such as `\@getcirc` and `\@circ`. To reflect in magnified fonts the changes of the line thickness, the commands `\thinlines` and `\thicklines` are also modified.

The example in Fig. 3 is produced by

```
\setlength{\unitlength}{1pt}
\begin{picture}(200,100)(-100,-50)
  \put(-50,0){\thicklines\circle{80}}
  \put(-50,0){\squareframe{40}}
  \magcircle{4}
  \put(50,0){\thinlines\circle{80}}
  \put(50,0){\squareframe{40}}
\end{picture}
```

where `\squareframe` is the user-defined command which draws the square with the specified side and the center at (0,0). It shows how the usage of the magnified circles enables one to overcome the upper limit of 40pt of the LaTeX circle diameter. It is necessary to note that following the magnification with `\magcircle{4}`, the thickness of the `\thinline` circles corresponds approximately to the thickness of the ordinary `\thickline` circles (`\magstep4` ≈ 2000).

Additional macros can draw 90° quarters of circles explicitly without tricky refinement of the parameters of the command `\oval`:

```
    \trcircle{diam} ⟶ \oval[tr]...
    \brcircle{diam} ⟶ \oval[br]...
    \tlcircle{diam} ⟶ \oval[tl]...
    \blcircle{diam} ⟶ \oval[bl]...
```

The center of the circular arc is positioned strictly at the point which represents the argument of the corresponding `\put`. The commands `\TRcircle`,

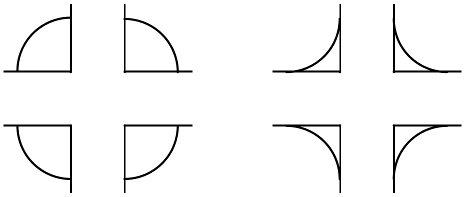


Figure 4: 90° circular segments

`\BRcircle`, `\TLcircle`, `\BLcircle` draw the 90° quarter circles with the reference point positioned at the corner instead of the center. Similarly, the commands `\tlsector`, `\TLsector`, `\blsector`, `\BLsector`, etc., draw circular segments together with horizontal and vertical radii. The proper positioning of the circular segments requires special precautions since it is necessary to take into account the line thickness and the specific alignment of the circular elements inside the character boxes.

The example in Fig 4 shows the usage of these commands:

```
\begin{picture}(200,60)(-100,-30)
  \put(-60,10){\thicklines\tlcircle{50}}
  \put(-60,10){\circle*{1}}
  \put(-60,10){\line(-1,0){25}}
  \put(-60,10){\line(0,1){25}}
  \put(40,10){\thicklines\BRcircle{50}}
  \put(40,10){\circle*{1}}
  \put(40,10){\line(-1,0){25}}
  \put(40,10){\line(0,1){25}}
  ... ... ...
```

The actual diameter of the circular segment is adjusted just as is done with the circles. The commands `\scaledcircle` and `\magcircle` also affect the thickness and the diameter of these circular segments.

Frames

The number of frames which is available in LaTeX is increased by PMGRAPH — besides the solid and dashed rectangular frames it is possible to draw double and triple frames in a variety of styles (Fig. 5). The commands `\frameBox`, `\ovalBox`, `\octalBox`, `\astroBox`, `\parquetBox` have the same structure as the command `\framebox`, but they draw the corresponding fancy frames:

```
\put(0,0){\ovalBox(100,50){oval}}
\put(70,0){\astroBox(100,50){astro}}
. . . . . . . . . . . . . .
```

An ordinary solid frame is drawn by `\frameBox`, the double and triple frames are drawn by `\frameBoX` and `\frameBOX`, respectively. Similar commands exist for double and triple fancy frames. The user can prepare personal macro commands to draw frame corners and extend the variety of fancy frames up to the limit of his/her imagination.

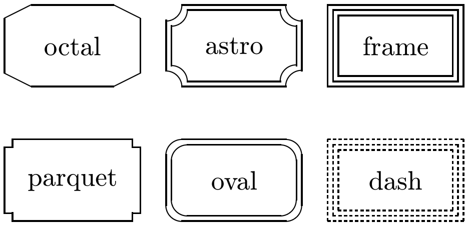


Figure 5: Examples of frame styles

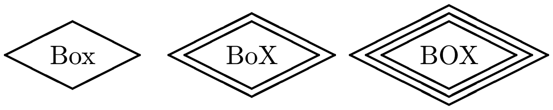


Figure 6: Romb-style frames

A more exotic variant of a frame can be created with the commands `\rombBox`, `\rombBoX` or `\rombBOX` as shown in Fig. 6. The style (i.e., inclination of the romb sides) and the distance between multiple rombs are set by the command `\rombboxstyle` with the default settings

```
\rombboxstyle(2,1,2pt)
```

The alignment of the romb around the box specified for these commands can be changed using an additional optional parameter (see the PMGRAPH manual for more details).

Each command to create a rectangular box has an optional parameter which specifies the "shadows" around the box. Each shadow style has a special letter, and a list of letters as the optional parameter results in a combination of the corresponding shadows. The standard shadow types are shown in Fig. 7. It is possible to draw several shadows of different types around an arbitrary corner of the frame as shown in Fig. 8:

```
\unitlength=10pt
\begin{picture}(20,15)
```

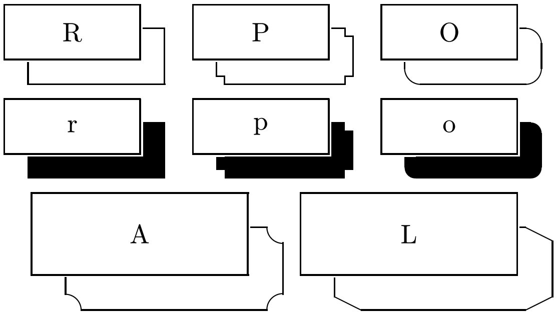


Figure 7: Examples of shadows

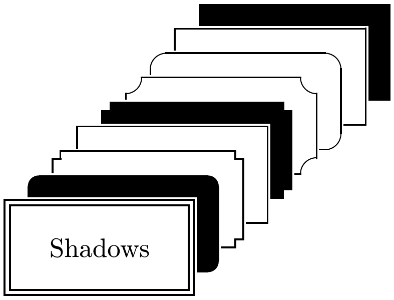


Figure 8: Multiple shadows

```
  \shadowcorner{B}
  \put(0,0){\frameBoX[oPR...](10,5){...}}
\end{picture}
```

The parameters of the shadows – thickness, corner size, additional shift, etc., – can be changed by user commands described in the manual.

Automatically scaled pictures

The idea behind the macros which are responsible for these functions is to calculate the `\unitlength` value of in terms of the *relative fraction* of the page width instead of explicitly specifying its value in points, centimeters, inches, etc.

The `\pictureunit[`*percent*`]{`*x-size*`}` command selects the value of `\unitlength` so that the picture, which is *x-size* units in width, occupies *percent*% of the width of the text. The `Picture` environment combines the automatic calculation of `\unitlength` with `\begin{picture}`–`\end{picture}`. The default, *percent*=100, corresponds to 90% of the text width. The default *percent* value can be adjusted by the command

`\renewcommand\defaultpercent{`*percent*`}`.

Examples:

```
\pictureunit[75]{120}
\begin{picture}(120,80)
...
\end{picture}

\begin{Picture}[75](120,80)
...
\end{Picture}
```

These macros are inspired by the `fullpict.sty` developed by Bruce Shawyer. Careful examination of the file `fullpict.sty` demonstrates some further bugs/warnings which require correction:

- each automatic scaling of `\unitlength` allocates a new counter;
- automatic scaling uses `\textwidth` as the reference width which results in improper functioning inside the list and `minipage` environments;
- the environments `fullpicture`, `halfpicture` and `scalepicture` are centered internally with `\begin{center}` — `\end{center}` which prevents the proper positioning of the picture in most cases.

The PMGRAPH.STY macros calculate the dimension `\unitlength` using the value `\hsize`, and as a result it works corectly also for `twocolumn` mode, inside the *list* environments `itemize`, `enumerate`, etc. (for example, all the figures in this paper are drawn using the environment `Picture`). The automatic centering and repeated allocation of the registers are corrected as well.

Acknowledgements

The authors are grateful to Dr. Kees van der Laan for the opportunity to present the preliminary results of our research at EUROTEX’95 (Aarnhem, The Netherlands). This research was partially supported by a grant from the Dutch Organization for Scientific Research (NWO grant No 07 – 30 – 07).

Calendar

1996

Jun 25 – 29 ALLC-ACH ’96: Joint International Conference, Association for Literary and Linguistic Computing / Association for Computers and the Humanities, Bergen, Norway. For information, contact Espen Ore (`Espen.Ore@hd.uib.no`, Fax: +47 55 32 26 56) or visit `http://www.hd.uib.no/allc-ach96.html`.

Jun 26 DANTE TeX–Stammtisch, Hamburg, Germany. For information, contact Volker Hüttenrauch (`Volker_Huettenrauch@HH.Maus.DE`. Last Wednesday, 18:00, details to be posted to the `tex-d-l` list. ¡

Jul 2 DANTE TeX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`MS@Dream.HB.North.de`; telephone 0421/2239425). First Tuesday (if not a holiday), 18:00, Universität Bremen MZH, 28359 Bremen, 4th floor, across from the elevator.

Jul 3 – 5 CNED96: 4th Colloque National sur l’Ecrit et le Document, IRESTE: Institute de Recherche et d’Enseignement Superieure, Nantes, France. For information, contact the Secretariat of the Colloquium, `cned96@lati.ireste.fr`, or visit `http://www.ireste.fr/cned96`.

Jul 4 DANTE TeX–Stammtisch at the Universität Karlsruhe, Germany. For information, contact Klaus Braune (`braune@rz.uni-karlsruhe.de`; telephone 0721/608-4031). First Thursday, 19:30, Rechenzentrum der Universität Karlsruhe, Zirkel 2, 3.0G Raum 316.

Jul 18 – 21 SHARP 1996: Society for the History of Authorship, Reading and Publishing, Fourth Annual Conference, Worcester, Massachusetts. For information, contact the American Antiquarian Society, `cfs@mark.mwa.org`.

Jul 28 – Aug 2 **TUG 17th Annual Meeting**: “Poly-TeX”, Dubna, Russia. For information, send e-mail to `TUG96@pds.jinr.ru`. (See the Call for Papers, *TUGboat* **16** (4), p. 429.)

Jul 31 DANTE TeX–Stammtisch, Hamburg, Germany. (For details, see Jun 26.)

Aug 1 DANTE TeX–Stammtisch at the Universität Karlsruhe, Germany. (For details, see Jul 4.)

Aug 1 DANTE TeX–Stammtisch at the Universität Bremen, Germany. For information, contact Martin Schröder (`MS@Dream.HB.North.de`; telephone 0421/2239425). First Thursday (if not a holiday), 18:00, Universität Bremen MZH, 28359 Bremen, 4th floor, across from the elevator.

Aug 20 **TUGboat 17 (4)**: Deadline for receipt of *technical* manuscripts (tentative). Theme issue, contributions by invitation.

Aug 28 DANTE TeX–Stammtisch, Hamburg, Germany. (For details, see Jun 26.)

Sep 3 **TUGboat 17 (3)** (2nd regular issue): Deadline for receipt of news items, reports (tentative).

Sep 5 DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Aug 1.)

Sep 5 DANTE TeX–Stammtisch at the Universität Karlsruhe, Germany. (For details, see Jul 4.)

Sep 23 PODP, Workshop on Principles of Document Processing, Xerox Palo Alto Research Center, Palo Alto, California. For information, visit the PODP Web page at `http://www.cs.umbc.edu/conferences/podp/`.

Sep 24 – 26 EP96, the International Conference on Electronic Documents, Document Manipulation and Document Dissemination, Xerox Palo Alto Research Center, Palo Alto, California. *Deadline for submission of papers: 1 April 1996.* For information, contact `ep96@xsoft.xerox.com` or visit `http://www.xsoft.com/XSoft/ep96.html`.

Sep 25 DANTE TeX–Stammtisch, Hamburg, Germany. (For details, see Jun 26.)

Oct 10 DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Aug 1.)

Oct 25 NTG 18th Meeting, on “Graphics and TeX”, Universiteit Utrecht, The Netherlands. Preceded or followed by a 1-day course, “(La)TeX and Graphics”, presented by Bogusław Jackowski. For information, contact `ntg@nic.surfnet.nl` or visit `http://www.ntg.nl/`.

Nov 2 DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Aug 1.)

Nov 12 **TUGboat 17 (4)**: Deadline for receipt of news items, reports (tentative).

Dec 5 DANTE TeX–Stammtisch at the Universität Bremen, Germany. (For details, see Aug 1.)

Dec 8 – 12 SGML ’96, Toronto, Canada. For information contact the Graphic Communications Association, Fax: +1 703-548-2867.

1997

May NTG 19th Meeting, Technische Universiteit Delft, The Netherlands. For information, contact `ntg@nic.surfnet.nl` or visit `http://www.ntg.nl/`.

For additional information on TUG-sponsored events listed above, contact the TUG office (415-982-8449, fax: 415-982-8559, e-mail: `tug@tug.org`). For events sponsored by other organizations, please use the contact address provided.

Status as of 31 May 1996

Institutional Members

The Aerospace Corporation,
El Segundo, California

American Mathematical Society,
Providence, Rhode Island

CNRS - IDRIS,
Orsay, France

CERN, *Geneva, Switzerland*

College of William & Mary,
Department of Computer Science,
Williamsburg, Virginia

Communications
Security Establishment,
Department of National Defence,
Ottawa, Ontario, Canada

CSTUG, *Praha, Czech Republic*

Elsevier Science Publishers B.V.,
Amsterdam, The Netherlands

Florida State University,
Supercomputer Computations
Research, *Tallahassee, Florida*

Grinnell College,
Noyce Computer Center,
Grinnell, Iowa

Hong Kong University of
Science and Technology,
Department of Computer Science,
Hong Kong

Institute for Advanced Study,
Princeton, New Jersey

Institute for Defense Analyses,
Communications Research
Division, *Princeton, New Jersey*

Iowa State University,
Ames, Iowa

Los Alamos National Laboratory,
University of California,
Los Alamos, New Mexico

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, Wisconsin

Mathematical Reviews,
American Mathematical Society,
Ann Arbor, Michigan

New York University,
Academic Computing Facility,
New York, New York

Nippon Telegraph &
Telephone Corporation,
Basic Research Laboratories,
Tokyo, Japan

Princeton University,
Princeton, New Jersey

Smithsonian Astrophysical
Observatory, *Cambridge,
Massachusetts*

Space Telescope Science Institute,
Baltimore, Maryland

Springer-Verlag,
Heidelberg, Germany

Stanford University,
Computer Science Department,
Stanford, California

Texas A & M University,
Department of Computer Science,
College Station, Texas

United States Naval Observatory,
Washington DC

University of California, Berkeley,
Center for EUV Astrophysics,
Berkeley, California

University of California, Irvine,
Information & Computer Science,
Irvine, California

University of Canterbury,
Christchurch, New Zealand

University College,
Cork, Ireland

University of Delaware,
Newark, Delaware

University of Groningen,
Groningen, The Netherlands

Universität Koblenz–Landau,
Koblenz, Germany

University of Manitoba,
Winnipeg, Manitoba

University of Oslo,
Institute of Informatics,
Blindern, Oslo, Norway

University of Stockholm,
Department of Mathematics,
Stockholm, Sweden

University of Texas at Austin,
Austin, Texas

Università degli Studi di Trieste,
Trieste, Italy

Uppsala University,
Uppsala, Sweden

Vrije Universiteit,
Amsterdam, The Netherlands

Wolters Kluwer,
Dordrecht, The Netherlands

Yale University,
Department of Computer Science,
New Haven, Connecticut

TeX Consulting & Production Services

Information about these services can be obtained from:

TeX Users Group
1850 Union Street, #1637
San Francisco, CA 94123, U.S.A.
Phone: +1 415 982-8449
Fax: +1 415 982-8559
Email: tug@tug.org

North America

Anagnostopoulos, Paul C.
Windfall Software,
433 Rutland Street, Carlisle, MA 01741;
(508) 371-2316; greek@windfall.com
We have been typesetting and composing high-quality books and technical Publications since 1989. Most of the books are produced with our own public-domain macro package, ZzTeX, but we consult on all aspects of TeX and book production. We can convert almost any electronic manuscript to TeX. We also develop book and electronic publishing software for DOS and Windows. I am a computer analyst with a Computer Science degree.

Cowan, Dr. Ray F.
141 Del Medio Ave. #134, Mountain View, CA 94040;
(415) 949-4911; rfc@netcom.com
Twelve Years of TeX and Related Software Consulting: Books, Documentation, Journals, and Newsletters
TeX & LaTeX macropackages, graphics; PostScript language applications; device drivers; fonts; systems.

Hoenig, Alan
17 Bay Avenue, Huntington, NY 11743; (516) 385-0736
TeX typesetting services including complete book production; macro writing; individual and group TeX instruction.

NAR Associates
817 Holly Drive E. Rt. 10, Annapolis, MD 21401;
(410) 757-5724
Extensive long term experience in TeX book publishing with major publishers, working with authors or publishers to turn electronic copy into attractive books. We offer complete free lance production services, including design, copy editing, art sizing and layout, typesetting and repro production. We specialize in engineering, science, computers, computer graphics, aviation and medicine.

Ogawa, Arthur
40453 Cherokee Oaks Drive,
Three Rivers, CA 93271-9743;
(209) 561-4585
Experienced in book production, macro packages, programming, and consultation. Complete book production from computer-readable copy to camera-ready copy.

Quixote Digital Typography, Don Hosek
555 Guilford, Claremont, CA 91711;
(909) 621-1291; Fax: (909) 625-1342;
dhosek@quixote.com
Complete line of TeX, LaTeX, and METAFONT services including custom LaTeX style files, complete book production from manuscript to camera-ready copy; custom font and logo design; installation of customized TeX environments; phone consulting service; database applications and more. Call for a free estimate.

Richert, Norman
1614 Loch Lake Drive, El Lago, TX 77586;
(713) 326-2583
TeX macro consulting.

Southern California PrintCorp
(800) 899-7267 x801
SAVE MONEY on 1-day Linotronic output for journals. Special *TUGboat* offer. Call now for more information.

Southern California PrintCorp
1915 Midwick Drive, Suite B, Altadena, CA 91001
(800) 899-7267 x888, Fax (818) 399-3565,
BBS (818) 398-3567
We have a ten year history providing 24-hour turn-around imagesetting of PostScript files. Call for FREE information on how *TUGboat*-ers can obtain low-cost, fastest available Linotronic publication production services in the U.S.

Type 2000
16 Madrona Avenue, Mill Valley, CA 94941;
(415) 388-8873; Fax: (415) 388-8865
pti@crl.com
$2.50 per page for 2000 DPI TeX and PostScript camera ready output! We provide high quality and fast turnaround to dozens of publishers, journals, authors and consultants who use TeX. Computer Modern, PostScript and METAFONT fonts available. We accept DVI and PostScript files only and output on RC paper. $2.25 per page for 100+ pages, $2.00 per page for 500+ pages; add $.50 per page for PostScript.

Outside North America

TypoTeX Ltd.
Electronical Publishing, Battyány u. 14. Budapest,
Hungary H-1015; (036) 11152 337
Editing and typesetting technical journals and books with TeX from manuscript to camera ready copy. Macro writing, font designing, TeX consulting and teaching.