

# TUGBOAT

Volume 19, Number 4 / December 1998

	347	Addresses
	348	A seasonal puzzle: XII / <i>David Carlisle</i>
<b>General Delivery</b>	349	From the President / <i>Mimi Jett</i>
	351	Editorial comments / <i>Barbara Beeton</i> TUG election; T <sub>E</sub> X '98; The end of an era — Phyllis Winkler retires; Sans Serif; Sauter font distribution has a new maintainer; Goodies on CTAN
<b>Typography</b>	353	Typographers' inn / <i>Peter Flynn</i>
	355	Typesetting with varying letter widths: New hope for your narrow columns / <i>Miroslava Misáková</i>
<b>Software &amp; Tools</b>	366	Editorial: EncT <sub>E</sub> X, by Petr Olšák / <i>Barbara Beeton</i>
	366	EncT <sub>E</sub> X — A little extension of T <sub>E</sub> X / <i>Petr Olšák</i>
	372	ConcT <sub>E</sub> X: Generating a concordance from T <sub>E</sub> X input files / <i>Laurence Finston</i>
<b>Language Support</b>	403	Cyrillic encodings for L <sup>A</sup> T <sub>E</sub> X <sub>2<math>\epsilon</math></sub> multi-language documents / <i>A. Berdnikov,</i> <i>O. Lapko, M. Kolodin, A. Janishevsky, and A. Burykin</i>
	417	Romanized Indic and L <sup>A</sup> T <sub>E</sub> X / <i>Anshuman Pandey</i>
	419	New Greek fonts and the <code>greek</code> option of the <code>babel</code> package / <i>Claudio Beccari</i> <i>and Apostolos Syropoulos</i>
<b>Hints &amp; Tricks</b>	426	'Hey — it works!' / <i>Jeremy Gibbons</i> Controlling abbreviations in BibT <sub>E</sub> X (Jeroen H. B. Nijhof); A small minus sign (Jeremy Gibbons); Ornamental rules (Christina Thiele)
	428	The Treasure Chest: A package tour from CTAN — <code>soul.sty</code> / <i>Christina Thiele</i>
<b>Abstracts</b>	431	Les Cahiers GUTenberg, Contents of issue 30
<b>News &amp; Announcements</b>	433	Calendar
	434	TUG'99 Announcement
<b>Late-Breaking News</b>	432	Production notes / <i>Mimi Burbank</i>
	432	Future issues
<b>TUG Business</b>	437	Institutional members
	438	TUG membership application
<b>Advertisements</b>	439	T <sub>E</sub> X consulting and production services
	440	Y&Y Inc.
	c3	Blue Sky Research

[A] ‘pi’ of type thrown on the floor constitutes no reading matter, though it contains its elements. The order, the sequence of letters in a sentence, is therefore paramount.

Victor Hammer [1955]  
“Those Visible Marks . . .”, *The  
forms of our letters* Typophile  
Monograph, New Series,  
Number 6, 1988

# TUGBOAT

COMMUNICATIONS OF THE T<sub>E</sub>X USERS GROUP  
EDITOR BARBARA BEETON

VOLUME 19, NUMBER 4  
PORTLAND . OREGON

DECEMBER 1998  
. U.S.A.

## **TUGboat**

During 1999, the communications of the T<sub>E</sub>X Users Group will be published in four issues. The September issue (Vol. 20, No. 3) will contain the Proceedings of the 1999 TUG Annual Meeting.

*TUGboat* is distributed as a benefit of membership to all members.

Submissions to *TUGboat* are reviewed by volunteers and checked by the Editor before publication. However, the authors are still assumed to be the experts. Questions regarding content or accuracy should therefore be directed to the authors, with an information copy to the Editor.

## **Submitting Items for Publication**

The next regular issue will be Vol. 20, No. 1. The deadline for technical items will be February 15; reports and similar items are due by March 1. Mailing is scheduled for March. Deadlines for other future issues are listed in the Calendar, page 433.

Manuscripts should be submitted to a member of the *TUGboat* Editorial Board. Articles of general interest, those not covered by any of the editorial departments listed, and all items submitted on magnetic media or as camera-ready copy should be addressed to the Editor, Barbara Beeton, or to the Production Manager, Mimi Burbank (see addresses on p. 347).

Contributions in electronic form are encouraged, via electronic mail, on diskette, or made available for the Editor to retrieve by anonymous FTP; contributions in the form of camera copy are also accepted. The *TUGboat* “style files”, for use with either plain T<sub>E</sub>X or L<sup>A</sup>T<sub>E</sub>X, are available “on all good archives”. For authors who have no network FTP access, they will be sent on request; please specify which is preferred. Send e-mail to [TUGboat@tug.org](mailto:TUGboat@tug.org), or write or call the TUG office.

This is also the preferred address for submitting contributions via electronic mail.

## **Reviewers**

Additional reviewers are needed, to assist in checking new articles for completeness, accuracy, and presentation. Volunteers are invited to submit their names and interests for consideration; write to [TUGboat@tug.org](mailto:TUGboat@tug.org) or to the Editor, Barbara Beeton (see address on p. 347).

## **TUGboat Advertising and Mailing Lists**

For information about advertising rates, publication schedules or the purchase of TUG mailing lists, write or call the TUG office.

## **TUGboat Editorial Board**

Barbara Beeton, *Editor*  
Mimi Burbank, *Production Manager*  
Victor Eijkhout, *Associate Editor, Macros*  
Jeremy Gibbons, *Associate Editor*,  
*“Hey — it works!”*  
Alan Hoenig, *Associate Editor, Fonts*  
Christina Thiele, *Associate Editor*,  
*Topics in the Humanities*

## **Production Team:**

Barbara Beeton, Mimi Burbank (Manager), Robin Fairbairns, Michel Goossens, Sebastian Rahtz, Christina Thiele

*See page 347 for addresses.*

## **Other TUG Publications**

TUG publishes the series *T<sub>E</sub>Xniques*, in which have appeared reference materials and user manuals for macro packages and T<sub>E</sub>X-related software, as well as the Proceedings of the 1987 and 1988 Annual Meetings. Other publications on T<sub>E</sub>Xnical subjects also appear from time to time.

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the T<sub>E</sub>X community in general. Provision can be made for including macro packages or software in computer-readable form. If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee in care of the TUG office.

## **Trademarks**

Many trademarked names appear in the pages of *TUGboat*. If there is any question about whether a name is or is not a trademark, prudence dictates that it should be treated as if it is. The following list of trademarks which appear in this issue may not be complete.

MS/DOS is a trademark of MicroSoft Corporation  
METAFONT is a trademark of Addison-Wesley Inc.  
PC T<sub>E</sub>X is a registered trademark of Personal T<sub>E</sub>X, Inc.

PostScript is a trademark of Adobe Systems, Inc.  
T<sub>E</sub>X and  $\mathcal{A}\mathcal{M}\mathcal{S}$ -T<sub>E</sub>X are trademarks of the American Mathematical Society.

*Textures* is a trademark of Blue Sky Research.

UNIX is a registered trademark of X/Open Co. Ltd.

## A Seasonal Puzzle

### XII

David Carlisle

david@dcarlisle.demon.co.uk

```

\let~\catcode~'76~'A13~'F1~'j00~'P2jdefA71F~'7113jdefPALLF
PA''FwPA;;FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevP
A@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamP
AGGFRruoPAqq71.72.F717271PAY7172F727171PA??Fi*LmPA&&71jfi
Fjfi71PAVVFjbigskipRPWGAUU71727374 75,76Fjpar71727375Djifx
:76jelsetU76jfiPLAKK7172F7117271PAXX71FVln0SeL71SLRyadR@oL
RrhC?yLRurtKFeLPFovPgaTLtReRomL;PABB71 72,73:Fjif.73.jelset
B73:jfiXF71PU71 72,73:Pws;AMM71F71diPAJJFRdriPAQQFRsreLPAI
I71Fo71dPA!!FRgiePBt'el@ lTLqdrYmu.Q.,Ke;vz vzLqip.Q.,tz;
;Lql.IrsZ.eap,qn.i. i.eLlMaesLdRcna,;!;h htLqm.MRasZ.il,%
s$;z zLqs'.ansZ.Ymi,/sx ;LYegseZRyal,@i;@ TLRlogdLrDsw,@;G
LcYlaDLbJsW,SWXJW ree @rzchLhzw,;WERcesInW qt.'oL.Rtrul;e
doTsw,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:jbye

```

Seasons greetings to all.

This code should be input to *plain* T<sub>E</sub>X, not L<sup>A</sup>T<sub>E</sub>X. For those without patience to figure out what the output will be, and to save the fingers and sanity of anyone who would like to try it out, the file can be found at <http://www.tug.org/TUGboat/Articles/tb61/xii.tex>.

Enjoy!

## General Delivery

### From the President

Mimi Jett

Greetings TUG Members!

As the year draws to a close and the new year arrives, we pause to reflect on the events and accomplishments we have enjoyed in 1998. I will review the year from my catbird seat, then go on to tell you about the fantastic meeting we had in Toruń, and our plans for a celebration—and opportunities for discovery—at our 20th annual meeting in Vancouver, BC, Canada, in August 1999. Once again, there is a good deal of administrivia to discuss. But, finally, we have the people and the plan in place to provide great service and benefits to our membership. We have elections coming up again, so please pay special attention to the words of wisdom from Barbara Beeton, and take the initiative to nominate yourself or someone else to the board, or for president. The more participation we have from our entire membership, the stronger we are. At this time, we have an extremely wonderful board; active in many aspects of creating benefits and learning for our members; willing to work across barriers; and gracious and respectful to each other; this group is a delight to work with. Most of our board has agreed to stand for re-election, that is good news! There are open seats on the in-coming board even if our standing group stays, so please think about it.

### 1998: The year in review

#### Release of $\text{T}_{\text{E}}\text{X}$ Live 3 to TUG members fantastically successful!

The first issue of *TUGboat* for 1998 included the CD  $\text{T}_{\text{E}}\text{X}$  Live 3. This was one of the most outstanding benefits we offered this year, and it was the fulfillment of a goal to include it in the first issue of the year. The combined, almost heroic, efforts of Sebastian Rahtz, Olaf Weber, Mimi Burbank (who provided many different testing platforms at the Florida State University Supercomputer Research Institute), Kaja Christiansen, Robin Fairbairns, Eitan Gurari, Fabrice Popineau, Andreas Scherer, Thorsten Schmidt and Eli Zaretskii were involved in assembling and testing the collection. Karl Berry, Thomas Esser, Graham Williams, and (as Robin puts it) “hordes of others” were the source of the software and packages. Together, they all brought forth the best  $\text{T}_{\text{E}}\text{X}$  Live release we have

ever seen. Indeed, this is one of the best things we have done for our members. Compilation and distribution of  $\text{T}_{\text{E}}\text{X}$  Live was a joint effort by UKTUG, GUTenberg, DANTE and TUG, with additional support from the Czech/Slovak, Dutch, Indian and Polish groups. Thanks to all! Truly, without the combined efforts of everyone, this would not have happened.

#### DANTE provides CTAN archive to all TUG and DANTE members

A three-CD set of the complete CTAN archive was distributed to all TUG members in *TUGboat* 19:2, thanks to a very generous donation from DANTE. We are indeed grateful to DANTE for this very valuable resource. The importance of CTAN was immediately clear to me when I heard Anita Hoover exclaim “I haven’t written a macro in years. . . everything I need is on CTAN.”

#### NTG release of 4all $\text{T}_{\text{E}}\text{X}$ provided to TUG members

In addition to the software and data resources already mentioned, we were also fortunate to have the opportunity to deliver 4all $\text{T}_{\text{E}}\text{X}$ , the popular Windows  $\text{T}_{\text{E}}\text{X}$  program, to our members. The program was developed and donated by Erik Frambach and Wietse Dol at NTG, with the cost of manufacturing and duplication covered, once again, by DANTE. Isn’t it nice to have generous relatives?

#### Confusion reigns after Microsoft announcement

An amusing, and all-too-believable article was posted as an April Fools’ Day prank claiming that Dr. Knuth had sold out  $\text{T}_{\text{E}}\text{X}$  to Microsoft. Complete with in-depth reporting, background, and quotes from the victims of this tragedy, it is no surprise the article was taken as genuine news by several people around the world. In the 10th anniversary edition super-issue of MAPS, NTG reprinted the article as it was originally written, and included color photos of Dr. Knuth and Bill Gates with serious expressions on their faces. This added credibility and fuel to the flames. I received messages from other groups asking what TUG will do, continue or disband, now that Bill is our leader? Up to that point, I didn’t realize that it was taken seriously. Of course, once the joke was realized it was appreciated as good humor.

#### Renewed interest in $\text{T}_{\text{E}}\text{X}$ and TUG apparent

This year has been another growth year for us, thanks to the great developments that are happening in Europe, Australia, and literally dozens of

countries around the world. With the migration of publications to the World Wide Web,  $\text{\TeX}$  is being discovered anew by authors and publishers of technical material. The ability to use live math for display or coded content is quite desirable in the world of database storage and delivery. All of this translates to more inquiries and requests for information about TUG. We have had a steady flow of new members, including both individuals and institutions.

### **Complete turnover in office staff**

Probably the most important thing that has happened for TUG this year is the lucky fate that sent Dick Detwiler to us. Dick has been managing the office since early summer, and it is finally beginning to feel like we are organized. Several times we have thought we had things figured out, just as another alligator surfaced from the swamp. The membership database was terribly inaccurate, with records of payment for dues confused for more than 500 members. We spent many hours editing, reviewing, and correcting the data. I think we have now fixed all records, and mailed the missing *TUGboats* and CDs to most members. If you have been billed or credited erroneously, I apologize.

### **TUG '98 in Toruń**

The 19th annual meeting was held in Toruń, Poland, August 17–22, and what a meeting it was! Several people commented that it was “the best TUG meeting ever.” Not having been to every meeting, I cannot say. But it was an incredible week. Every session was started pretty much on time and ended on time. The quality of all the papers was very good. Hans Hagen was exceptional in all of his presentations. Hans was voted the best in all three categories; content, presentation, and overall, by the

audience. The workshops were well organized and highly informative. Everything was orchestrated by the GUST team; all events, meetings, and facilities were perfect. Visiting the Teutonic Knights’ castle was quite an experience, not only feeling the history of the place, but watching Gilbert breathe fire and hearing Boguslaw leading folk songs, made the night one to remember for a lifetime. Thanks to our friends at GUST for a wonderful time, and to the people of Poland for their gracious hospitality.

### **Board News**

We are fortunate to welcome Mr. Philip Taylor to our board. Phil has been active in TUG and UK TUG for many years, and brings a depth of experience and knowledge to us. The following directors have resigned: Cameron Smith, Donna Burnette, and Jiří Zlatuška. We thank them for their service to TUG. Please don’t forget the upcoming elections. We would love to have more members standing for the board.

### **Plans for 1999**

#### **TUG '99 — 20th Annual Meeting in Vancouver, BC, August 15–20**

This will be an outstanding conference, based on the papers that have been accepted and the training, workshops, and events that are planned. (See the conference preliminary schedule on page 434.) The conference and program committees are actively creating an itinerary you will enjoy. Please stay tuned to the TUG website (<http://www.tug.org/tug99/>) for additions and announcements.

See you in Vancouver!

---

## Editorial Comments

Barbara Beeton

### TUG election

Please remember that this is an election year for TUG. Both the presidency and a number of seats on the Board are open for candidates. Nomination papers are due to the TUG office by March 15, 1999, balloting will take place during the spring, and newly elected officials will assume their offices at the annual meeting in Vancouver.

The form for nominations appeared in *TUGboat* 19(3), on page 234. The form is also available from the TUG Web site, at <http://www.tug.org/>, in the form of a PDF file, or a copy can be obtained on request from the TUG office ([office@tug.org](mailto:office@tug.org)).

Your participation in this election will help to determine the course that TUG will take headed into the new century. Don't leave it to chance—become active and help make it happen.

### TeX '98

During the past summer, Don Knuth undertook his periodic review of the accumulated bug reports for TeX, METAFONT, and the *Computers & Typesetting* book series. The new versions of all the source files are in place on CTAN, in <ftp://ftp.ctan.org/tex-archive/systems/knuth/>, mirrored from [labrea.stanford.edu](http://labrea.stanford.edu).

Here is a brief summary of changes.

- The most important *TeXbook* corrections appear in the files `errata.nine` and `errata.tex`; some insignificant changes (e.g., page numbers in index entries out of order) are corrected in `texbook.tex` but not mentioned in the errata files, so anyone who thinks s/he has found an error should check it in `texbook.tex` before reporting it.
- `plain.tex` has had a few changes and now has a format version 3.1415926, two steps ahead of the version of TeX itself. The definitions of `\AA`, `\d`, `\b`, `\c`, `rightarrowfill` and `leftarrowfill` were corrected or improved to work correctly or more robustly in a wider range of situations. A new control sequence, `\Orb`, was introduced to typeset the big circle used in `\copyright`.
- A few corrections were made to comments in *TeX: The Program*, but the version remains 3.14159.
- In *The METAFONTbook*, numerous nitpicky corrections are recorded in the errata file, but the only really important change is the cor-

rection to the syntax of path expressions on page 129 (repeated on page 213).

- METAFONT moves to version 2.7182, correcting a bug involving unprintable strings of length 1.
- Changes to Computer Modern typefaces are documented completely in the file `cm85.bug` as well as in `errata.tex` (changes to Volume E). Most corrections simply make the programs more robust in the presence of weirder combinations of parameters. Contrary to previous claims that the shapes would never change again, a few have changed in nontrivial ways, to improve their appearance in the new editions of *The Art of Computer Programming*: lowercase beta and omega, uppercase sans serif C and G, and the position of the dots on the i's in sans serif fi and ffi ligatures has descended to the normal position for i dots.
- DVltype is now version 1.6; it reports some errors better.
- VPtoVF is now version 1.5, fixing a bug with respect to rules of dimension zero.
- Typos in `GFtoPK.web` and `PKtype.web` were corrected, with no change to version numbers.
- In `logo.mf`, the S has been redesigned; it now sort of assumes that a T will follow, as in METAPOST.

Don's advice is to TeX and print out the file `errata.tex` for reference. His transmittal letter concluded, "In summary, I'm pleased that people still care enough about TeX/METAFONT to understand the details and to help me get them right. But oh how I wish I hadn't made so many mistakes!"

Don will next address TeX-related bugs in 2002. Until then, I will continue to collect reports, acting as his entomologist.

### The end of an era — Phyllis Winkler retires

When Don Knuth created TeX, he intended it to be a tool for himself and his secretary, Phyllis Winkler, to prepare his books and papers for publication.

On October 1, 1998, Phyllis retired from Stanford, after 32 years of service, for 28 of which she was Don's secretary. As he says on his Web page of news for 1998<sup>1</sup>,

She typed more than 200 of my papers, most of which required several rounds of revisions. She buffered all of my email and telephone messages. She administered the editorial

---

<sup>1</sup> <http://www-cs-faculty.stanford.edu/~knuth/news98.html>; a photo shows Phyllis with Don, who is, for I think the first time I've seen it, wearing a necktie.

work of more than a dozen technical journals, and helped out with numerous research projects. She made online indexes of all the correspondence in our files. She did all of the initial keyboarding for the new editions of *The Art of Computer Programming*, Volumes 1 and 3 — amounting to more than 1500 printed pages of what printers used to call “penalty copy” because it is so hard to do. And so on and so on, what a team we made! And she was simultaneously also serving as secretary for several other faculty members.

I remember Phyllis most fondly. I met her in 1979 when I was first sent to Stanford with a small group from AMS to learn  $\TeX$ ; she took very good care of us. Whenever a TUG meeting was held at Stanford, I always enjoyed checking in with her to find out what was happening. I learned some interesting personal things about her, for example that her son-in-law raised and trained large cats for several well known magicians; a delightful poster on her wall showed him with his hands full of tiny tiger kittens, their mother looking on with curiosity but without concern. Phyllis told me that once when she was visiting her daughter, her son-in-law suggested she might go out in the back yard to get some exercise running around with their resident panther. I *think* they were just kidding . . .

Phyllis also took care of communicating messages between Don and me whenever he would work on the current batch of  $\TeX$  bug reports. She could always be depended on to get necessary messages through to him, but insulate him, firmly but politely, from things that weren't urgent.

On September 30, members of the Stanford Computer Science Department held a retirement party for Phyllis. Among the other greetings, a resolution from the TUG Board expressed our appreciation for all her contributions over the past 20 years.

Along with many other friends, I wish Phyllis a long, productive, enjoyable retirement.

### Sans Serif

Don Hosek, editor and publisher of *Serif: The Magazine of Type & Typography*, has created an electronic adjunct — “Sans Serif: The On-line Companion to Serif”. These Web pages (found at <http://www.quixote.com/serif/sans/>) contain some material related to items in the print product, along with a full calendar of type- and print-related events. Check it out — it contains more local and specialized events than we are able to include in the *TUGboat* calendar.

### Sauter font distribution has a new maintainer

The Sauter font distribution, a comprehensive set of parameters for automated generation of Computer Modern and other METAFONT fonts, has been maintained for quite a long time by Jörg Knappen, who took over this task from the originator, John Sauter.

Owing to a change in his employment status, Jörg found it necessary to look for, and has found, a replacement. The new maintainer of the Sauter font distribution is Jeroen Nijhof; he can be reached at [J.H.B.Nijhof@aston.ac.uk](mailto:J.H.B.Nijhof@aston.ac.uk).

### Goodies on CTAN

With the recent posting of yet another translation of *The (not always) short introduction to  $\LaTeX$*  (familarly known as *lshort*, the number of languages in which this little manual is now available has reached seven: English, Finnish, French, German, Mongolian, Russian, Spanish.

On CTAN, *lshort* can be found in [/tex-archive/info/lshort/<language>](http://tex-archive/info/lshort/<language>).

This is a fine beginner's manual for  $\LaTeX 2_{\epsilon}$ , and while it doesn't replace Lamport or the other formally published manuals, it is readily available, and the price is right!

Other new or updated packages, tools, documentation, you name it, . . . , appear on CTAN in a continuing stream. How is one to know what is there, and to determine whether it is useful in one's own work? With this issue of *TUGboat*, we have initiated a new column, “The Treasure Chest”, in which one or more packages will be presented in each regular issue. Enough examples will be shown to provide a flavor of the package, so that a reader can decide to investigate further, if it's of interest. The first package to be presented is *soul.sty*. Take a look, let us know what you think, and if you have any suggestions for other packages you would like to see highlighted, send them to Christina Thiele ([cthiele@ccs.carleton.ca](mailto:cthiele@ccs.carleton.ca)). Better still, if you'd like to volunteer to help produce the column, Christina will be delighted!

◇ Barbara Beeton  
American Mathematical Society  
P. O. Box 6248  
Providence, RI 02940 USA  
[bnb@ams.org](mailto:bnb@ams.org)



# Typography

## Typographers' Inn

Peter Flynn

### 'C' stands for Euro

January 1st came and went, and we survived the introduction of the Euro, the planet's ugliest-named currency. I can now write cheques in €, do electronic transactions, and even lodge credits, should anyone be generous enough to send me money. As a L<sup>A</sup>T<sub>E</sub>X user, I have discovered the `\texteuro` command in the `textcomp` package (which I should have mentioned last time, but *mea culpa*, I am a recent convert to L<sup>A</sup>T<sub>E</sub>X and am still finding stuff squirrelled away that I didn't know about). And, I'm pleased to say, L<sup>A</sup>T<sub>E</sub>X's € is a much more suitable design to go with those serif fonts which have none of their own than the strange C-like designs put out by Microsoft in their TrueType replacement fonts (designed by and licensed from Monotype, of all people!). But `\texteuro` uses PostScript fonts (the T1 encoding) and while that's fine by me, it's not for everyone.

Full marks, therefore to Henrik Theiling for his `eurosym` package, which implements the original EU (sans-serif) design in roman, bold, italic, and outline using METAFONT, so it's usable in T<sub>E</sub>X-based systems anywhere. No seriffed version yet, but the following table shows some of the glyphs available.

	n=sc	sl=it	ol
m	€	€	€
bx	€	€	€

Adobe also has the Euro in PostScript fonts available for download, including sans and serif versions (with serifs top and bottom, too!), and there's already a Euro in the `china2e` package (a METAFONT font). Bitstream sells a standalone pi font with the Euro symbol, and will customize your existing fonts for you (for a charge). Linotype sells nearly 200 Euro symbols for DM 100 but makes the same mistake as Monotype in pretending the E is a C in the serif versions.

The EU has laid down that the official design is to be used regardless of the surrounding font (in both style and colour, see <http://europa.eu.int/euro/html/entry.html>). Fortunately I don't know anyone daft enough to want to follow that diktat.

As I write, each of my Euros appears to be worth \$1.30 Canadian, so I've started to save for the TUG meeting in Vancouver.

## L<sup>A</sup>T<sub>E</sub>X and glue

I've pretty much settled down to using L<sup>A</sup>T<sub>E</sub>X now. I don't make so many mistakes and I've stopped typing plain T<sub>E</sub>X on the rare occasions when I actually write a document in raw code. Most of my text is authored by other means and converted to L<sup>A</sup>T<sub>E</sub>X for formatting, so my misgivings about the default L<sup>A</sup>T<sub>E</sub>X appearance mean that most of what I do is writing or modifying style files to publishers' specs. I've started to turn some of the ideas which have spun off from this into class and package files in preparation for a project I mentioned online recently and which I will be presenting in Vancouver.

I had several responses to my suggestion that we ditch the weird concept that the default style for reports should have chapters, most of them supporting a change. It's probably inadvisable to change the source of `article.cls`, as too many people have too much private code rigged to cope with its peculiarities, and they rightly rely on the stability of T<sub>E</sub>X systems to maintain their text. What I'm aiming at is a package that repairs this and other leaks and seals them up so that authors have to spend less time fiddling and thus have more time for writing<sup>1</sup>. Articles should work more like authors and publishers expect them to, books more like books, and reports more like reports. Maybe this will even help stem the flow of FAQs about these problems on `comp.text.tex`.

And what was that about glue? When you repair a puncture in a tire, you glue a patch of rubber to the inner tube. In the early days of cycling and motoring, tire rubber needed heat treatment after repairs to ensure the patch was properly bonded, and this treatment was called 'vulcanising'. To avoid the bond degrading as the rubber flexes in use, the glue is made of our good friend latex (plus assorted chemicals). Those of you with long memories may recall childhood cycle repairs with 'self-vulcanising' glue, which replaced the need for heat-bonding. Hence by a tortuous path the name `vulcan` for a package which seals the leaks in L<sup>A</sup>T<sub>E</sub>X — come to Vancouver and see (tell your boss you're off to learn about latex bondage or something: Dan Quayle will explain).

## Backquotes

Maybe it's just the way that once you notice something once, you repeatedly see it all around you, but I've spotted the reversed quote (‘) several times,

<sup>1</sup> But where should we be without the inveterate fiddling author inventing new styles?



Figure 1: Type fac-simile of the SGML97 logo

including some extremely public displays which included the logo which appeared on all the posters, leaflets, proceedings, and assorted publicity for the XML'97 conference in Washington, DC. They got it right on the title page, and anywhere that it was reproduced from typed characters, but the logo itself, approximated in Figure 1 with CM fonts, used the reversed quote. I'm curious to know why so I've sent a message to the designer and I'll let you know.

### Get writing

The new journal I mentioned last time, *Markup Languages: Theory & Practice* [1], is up and running. It's quarterly, peer-reviewed, and the first one of its kind devoted to text markup. I'm therefore repeating my call for articles: as the markup we use for  $\text{\TeX}$  and  $\text{\LaTeX}$  was one of the major advances in the move towards logic-based or structural markup, I feel that there is plenty of scope. Dip your quill in the ink and start writing now.

### Postscript

Probably like many of you I've been using PostScript fonts for years. They're portable, convenient, reasonably accurate, and although the hinting isn't a full substitute for design-sizing, in most practical situations they work just fine. I don't do many jobs requiring extremely large sizes, so I haven't run into the problems that I am told exist in advertising work, for example.

But my guess is that most  $\text{\TeX}$  systems, particularly in research or academic sites, don't have PostScript fonts (a font survey would be interesting). There is a cost involved once you go beyond Charter and the other free PostScript fonts, and although it is small per font, it can be outside the budget of many individuals, especially students, and even some projects. The bigger stumbling-block is the installation: I had my own problems when I did the first few fonts, but I was lucky to have generous and helpful people on call who patiently explained what I needed to do; and this was long before the new  $\text{\TeX}$  Directory Structure.

I have therefore finally gotten around to writing a new PostScript font installer. The old `mkvf` program which I wrote to take the Virtual Font route was a shell script, and fairly crude; the `mkcd` batch

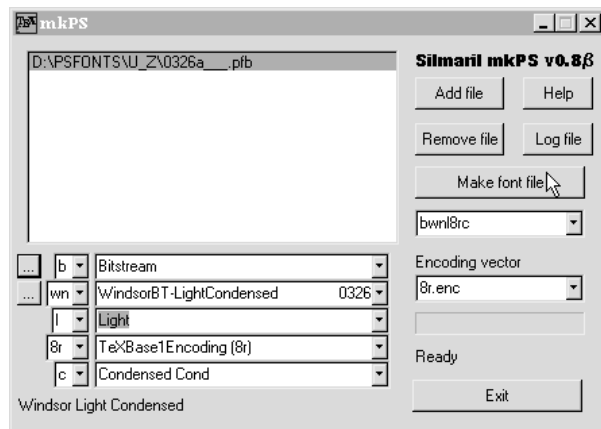


Figure 2: The MKPS PostScript font installer

file for DOS/Windows systems which followed was never satisfactory as the restricted operating environment available precluded it doing all that `mkvf` did. This time I have taken the plunge into Windows'95 and rewritten it as a windowing utility: it's the most prevalent platform I support. The tool I used, *Visual DisplayScript*, is a very simple and effective way of tying together a simple interface to make a little utility (see Figure 2).

It assumes the TDS, although you can change that if you store your fonts elsewhere, and it makes reasonably intelligent although by no means fool-proof attempts to deduce the Karl Berry fontname abbreviations from the extended descriptive name in the AFM file (part of this grew out of having to rescue a client's broken installation where all the AFMs were corrupt and I had to try and dig into several hundred PFBs for the same data). I hope to have a distributable beta release by the summer: if anyone can recommend a similarly simple environment for producing X Window mini-apps, I'd be happy to hear of it.

The core that does the work is about four lines: `afm2tfm`, `vptovf`, some file-copying, and the appending of the relevant line to `psfonts.map` (it does assume `dvips`: it's all I know about). What takes the time, as usual, is handling the configuration, the deducing, the file-loading, and working out the name (and checking in the fontname map files). The user should be able to drag and drop a PFB file onto it, check that it has correctly resolved Bonemontano, Inc's *Gracatia Sancta* Skinny Weird DemiBold into `zgsdw8ro` and then just go ahead and do everything, including creating a skeleton FD file. If you're tired of hearing people complain that  $\text{\TeX}$  systems have only got one font, and it's soooo hard to get it to

work with anything else, mail me to go on the beta list. And no, I'm not offering €1.00 per bug.

### References

- [1] *Markup Languages: Theory & Practice*. MIT Press, Cambridge, MA. ISSN:1099-6621.

◇ Peter Flynn  
Computer Centre  
University College Cork  
Ireland  
[pflynn@imbolc.ucc.ie](mailto:pflynn@imbolc.ucc.ie)  
<http://imbolc.ucc.ie/~pflynn/>

---

## Typesetting with Varying Letter Widths: New Hope for Your Narrow Columns

Miroslava Misáková

### Introduction

The line-breaking algorithm based on optimum fit, which serves as a basis of the  $\text{\TeX}$  typesetting engine is considered to be of very high quality. However, there are still a large number of line-breaking problems where the results are not satisfactory. Especially when typesetting text in narrow columns with justified line margins, its optimising criteria can usually be met only by enlarging the amount of white space allowed (`\emergencystretch`). This introduces unacceptable distortions in the overall grayness of the page appearance.

One way to tackle this problem is to go back to an ancient technique used by Gutenberg for typesetting his 42-line Bible: extend the set of font types by letters with width variations. If one succeeds in selecting optimal typefaces modified to suit individual lines, one can minimize the annoying “holes” which otherwise occur within the pages.

When considering this approach, we come to the METAFONT system that makes it possible to keep constant stem width even when the width of individual letters is modified, and to use the current optimum-fit algorithm of  $\text{\TeX}$  for finding suitable line breaks within the paragraphs to be typeset. A real implementation would require the typesetting system to be rewritten completely, especially its line-breaking algorithm. However, even with lower effort, we can happily explore this method and perform various (aesthetic) experiments.

This paper demonstrates the potential of a simple method of implementing the idea of extending font types by using letters with width variations. Selecting optimal typefaces modified to suit individual

lines should make it possible to minimize the annoying “holes” which otherwise occur within paragraphs.

We will present the results of paragraph breaking using  $\text{\TeX}$  and the improvements we can get using iterated line-breaking, based on variants of the fonts modified by width distortion. We will discuss benefits and limitations of this method.

### The average document

When a  $\text{\TeX}$ ist, on a lovely summer day, enters his `\bye` and leaves the real world for the gates of  $\text{\TeX}$ 's brackets, they will be surprised to find that the quality of the average document at the dawn of the twenty-first century is still less than satisfactory. They might analyse more texts and realize that, nevertheless, the situation is better than some five years ago. The initial enthusiasm over the mere existence of DTP systems declines and both the designers and users of those systems start to exhibit a certain self-discipline in re-acquiring the achievements of this 500-year-old science called typography.

The vast majority of small typesetting problems encountered in the process of plain composition that arise from the competition of three paradigms (uniformity, information and structure) can be solved by any program that aspires to being called “the typesetting system”. It is a must if we want to tackle hyphenation, ligatures, kerning, ties and various types of dashes. However, in the presence of this, there is much less progress in an area which attracts the user's attention very quickly and with great intensity:

How is it that this issue — so important to typesetting masters in the good old days — is so ignored by almost all present-day DTP systems? If we want to avoid sparse typesetting, perhaps we cannot apply just a simple algorithmic approach. With only a little exaggeration, we can say that, with respect to the goal of producing consistent grayness on the page, digital composers are still at the typewriter level. With despair we observe that even when  $\text{\TeX}$  is relatively better because of the optimum-fit line breaking algorithm, it cannot avoid all problems.

What, in fact, is sparse composition? We could say, for example, that it is plain composition, where the inter-word spacing is in the range of 66 to 150 per cent of the width of the ideal space, as specified by the font designer. But even in documents typeset with  $\text{\TeX}$ , we can often find spaces that exceed this limit by several magnitudes. Philip Taylor [6] shows how to try to improve such results but there is a general consensus that, for example, justified narrow columns are unsolvable if the regular level of grayness of the text is the main criterion. With

decreasing \hspace the problems of the line breaks suddenly jump out.

### Narrow columns today

Why do we need those narrow justified columns anyway? Isn't the natural answer simply to put 70 characters on a line — the most pleasant number for the eye of our reader [5]? No way! Typesetting is always a compromise between ergonomics and overall design that may require parts that are hard to produce (flowing around pictures, newspapers). Unfortunately, we cannot simply forget the existence of narrow columns. A more typical approach to this problem is letterspacing, a solution which is awful yet widely used. We can only wonder how a method so heartily frowned upon<sup>1</sup> has found its way into present-day typesetting. With today's greater cultural awareness, letterspaced words can particularly confuse readers used to certain national traditions that use it for *e m p h a s i z i n g* when appropriate italics was not at hand. Only by slowing down and asking "why did they emphasise *this* word" can we realize that it stands on a line by itself and is letterspaced only in order to justify the text. The flow of information is significantly disturbed.

A much more acceptable solution is to use raggedright lines rather than justify them. The reasons preventing the composer from picking up this style for any narrow column are, to be true, irrational. However, typography, as a discipline serving irrational beings, has to accept them. People simply *want* justified columns. It is like architects (often compared to typographers), who would have a hard time thrusting non-linear walls upon their customers; we treat books with unjustified margins with a certain disdain: we tolerate it only where justified lines would lead to much greater violation of the overall grayness than would unjustified lines.<sup>2</sup>

So we would like a different tool in our fight with sparse typesetting. One possibility is the approach presented in the remainder of this text — that is, to typeset using typefaces containing wider or narrower variants of characters while preserving all of their design characteristics. This way we give the typesetting algorithm one more degree of freedom in its search for optimum breaks; the algorithm

is not constrained to change only the width of interword spaces. Some situations viewed as critical with regular systems become easier to solve (for example, lines with a minimum number of spaces — the more letters we have on the line, the more we can slightly vary their widths and get a reasonable result). We have more letters than spaces in regular texts but we cannot alter their width as much as we can the white spaces. To find out how practical and applicable this idea might be, we used experiments exhibited later in this article.

### Is it moral to play with such a terrible thing?

Wider M's. Narrower O's. Isn't it a Greek gift which, in an attempt to make the page more regular, will break up the visual well-being of the reader because their brain will be confused with strange abnormalities in the shapes of letters? It's a weighty question. Superficial specification of the problem might even lead to the notion of a result that a master typographer will condemn — what's going on here might seem to be a *mixing of fonts* in its worst form because we suddenly have dozens of different typefaces, maybe even a different font for each line, whereas it is generally accepted to have at most three or four fonts in the whole document. However, here the intent is not to have the document as fancy as possible (goals of designers spitting around fonts and typefaces) but to stifle any irregularity. The modifications to the characters shouldn't exceed the limit beyond which they are recognized without a more thorough examination. This limit would need to be derived from empirical tests; it will vary for both different readers and different typefaces. The first estimate assumes modifications should not be greater than 5 per cent of the original width of the character. Another requirement is to maintain a uniform look to the whole line, which is the greatest unit that the reader really perceives.

It is hard to predict if there will regress appear frequent, an uncertain feeling of incorrectness or that it is simply harder to read.<sup>3</sup> We need to make many practical tests and we will probably not be able to generalize results to other font families.

It is useful to remember that we are primarily speaking about minuscules; the text of a title on

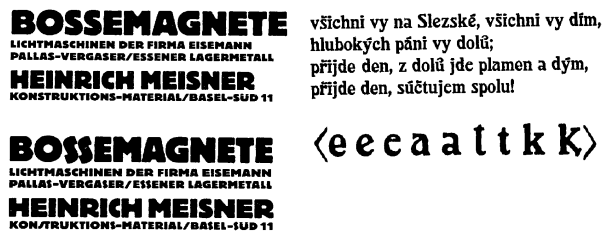
<sup>1</sup> "A man who would letter-space lowercase would steal a sheep." F. Goudy

<sup>2</sup> Another example of an algorithm broken by users' solutions is hyphenation in esperanto. The authors of this language are allowed to hyphenate at any point in a word; users of the language, however, have come up with various artificial constraints that have led to hyphenation patterns that are the same (or bigger) in size than those of other natural languages. . .

<sup>3</sup> The paths of human vision are strange. As an example, consider the long-standing dispute about sans serif typefaces: they ought to be more readable because they do not disturb the reader with serifs and lead the eye more quickly to the important shapes of the letter, and yet it seems to be less convenient because it lacks the bounding box of line that leads the reader's eye along.

which the eye will spend a longer time and thoughtlessly explore the shapes requires different principles than plain paragraph, where the main goal is to pass on the information and disappear.

**Historical reminiscences:** When in doubt, it is always good to look into the history, to experience gained by past generations. When studying historical contexts, we can see that some variations on this method were used by many typographers who needed precisely justified documents. Oldřich Hlavsa [4] gives an example of varying characters that can be found in a catalogue of type from 1920.

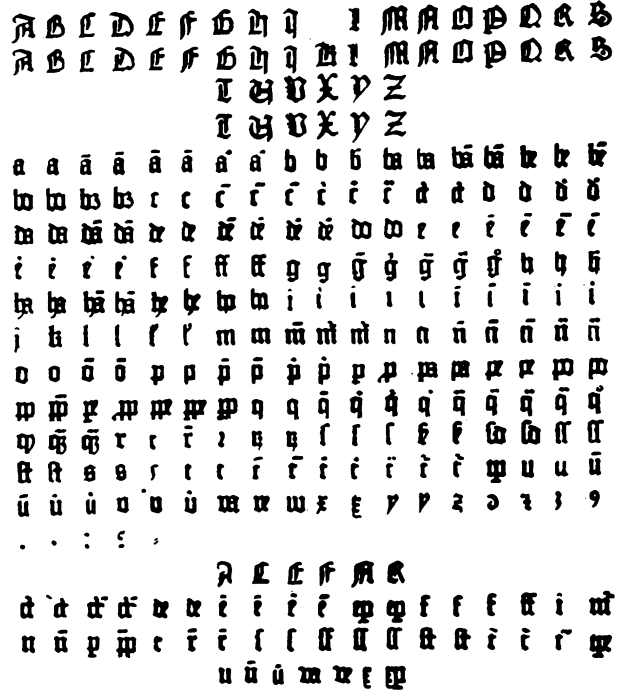


**Figure 1:** With variant typefaces, it is relatively easy to create justified but still closely tied advertisement. (*left*) Only a closer look at Preissig’s solution for the design of a book of poetry shows modified letters. (*right*)

Vojtěch Preissig has also added variants of some letters to his font to get lines with regular light and a more beautiful appearance.

It is also important to note that in traditional hot-metal typesetting it was quite common to have (almost linear) contraction of width, up to about 1 per cent. It was achieved by strong tightening of the screws, taking advantage of the elasticity of the typesetting alloy.

**What about “John-from-Good-Mountain”?** If we were to consider the above examples as sporadic fads, we can go deeper to the roots, to Johann Gutenberg’s workshop. The exact records about his “art of multiplication of books” are not known, but what we do know is that the admired uniform grayness of his 42-line Bible was accomplished by using dozens of ligatures, often abbreviations, placing punctuation to the middle of inter-word space and especially by using a vast set of character types. It was the selection of characters with variant widths which allowed him to typeset those perfectly justified lines that inspired Europe and that were so akin to good manuscripts. We can assume that his goal was nothing less than to achieve uniformly dis-



**Figure 2:** The typeset used in Gutenberg’s Bible had hundreds of items.

tributed white space in the whole document. The great amount of work that he devoted to the problem confirms how great a problem sparse typesetting was for the old typographers.

**The quest**

When exactly typographers lost the need to create pages with perfect uniformity in grayness is not known. Probably this tradition did not survive the switch over from texture typesetting to the rounded italic typefaces of the present. Leaving the distinct vertical casts of the letters, the effort to make the mirror of a page into the regular grid has vanished and a much simpler method for line justification has predominated along the centuries: widen the interword spaces. Other techniques, as we have shown, run through the history of typography; they were, however, never used widely. I believe it was not caused by aesthetic condemnation but by overwhelming technological difficulties. Not until electronic typesetting brought simpler ways for experiments with these micro-typographical effects and make it possible to include them in our documents.<sup>4</sup>

<sup>4</sup> The really practical and transparent use of variant-width characters would of course mean a really new generation of the line-breaking module to typesetting algorithms. Such a task is far beyond the scope of my thesis, which discusses these ideas. Nevertheless, URW started to work on

**Et ingressus angelus ad eam dixit. Ave gratia plena: dominus tecum: benedicta tu in mulieribus. Que cum audisset turbata est in sermone eius: et cogitabat qualis esset ista salutatio. Et ait angelus ei. Ne timeas maria: inveniisti enim gratiam apud deum. Ecce concipies in utero et paries filium: et vocabis nomen eius iesum. Hic erit magnus: et filius altissimi vocabitur. Et dabit illi dominus deus sedem dauid patris eius: et regnabit in domo iacob in eternum: et regnum eius non**

Figure 3: From the 42-line bible.

### Implementation

**Fonts:** When trying to find how to initiate the idea of varying-width characters, the problems with fonts is of the main importance. Essentially, there are two approaches: a) extend the typesetting with width variants of certain group of characters, or b) generate the necessary typeface on the fly, according to the requests from the typesetting system. The first solution, supposedly used in the hz-system, has both some drawbacks (limited flexibility that comes from the fixed set of available characters) and advantages: the set of characters will (should) be prepared by experienced designer, which will prevent possible excesses, that could appear during automatic generation; the disk usage is lower as well. The second solution requires very good cooperation of the typesetting system with the program generating the fonts. Also the number of fonts used in document will be enormous.<sup>5</sup> The need to change the shape of the characters and yet to keep all the

it. Its hz-system is, however, a typical commercial product: the information vacuum is impenetrable, and no test or any other version is available on the market. Yet the suspected existence of the hz-system was a source of inspiration and hope for us, hope that it would make sense to explore the

VLW approach. We did concentrate on the cooperation with the TeX typesetting system, the tools that would allow anybody to test the utility and limits of this method; hopefully one day somebody will implement it in a really systematic way.

<sup>5</sup> This disadvantage could be eliminated by different font management, similar to font servers that generate only characters needed, not the whole fonts.

main characteristics of the font (especially the stem width) implies the use of METAFONT.<sup>6</sup>

**Line breaking:** You barely get sparse typesetting with *optimum fit* algorithm. That was the thought during the first years of experience with TeX. The reality is slightly different. People are too lazy to aid the hyphenation algorithm or rewrite the text to get better line breaks. On the other hand, *optimum fit* and the box-glue-penalty paradigm itself is still a very strong concept.

Probably not very hard extension of it by *gluish box*, that would merge some features of both boxes and glues, would allow such a change of the line-breaking algorithm that would reflect the fact that even the material in the box has got some width variability. The badness of lines today is computed using the formula  $b = |r|^3 \times 100$ . If we could stretch or shrink both spaces and characters, the adjustment ratio  $r$  would come out as something like

$$\alpha \times \text{change of spaces} + \beta \times \text{change of characters}$$

Fine-tuning the balance between  $\alpha$  and  $\beta$ , the user could express if they prefer loose lines or lines containing “deformed” font. By proper setting of these parameters, one could even get the backward compatibility with TeX.

### How to simulate this approach in TeX

Let’s stop theorizing and see what we can do in the current TeX, to finally understand how this innovative typesetting looks like; how it works and what effect it has on readers. After considering various approaches (prototype system as a TeX change file, typesetting system independent on TeX, other ways) we opted for the method of postprocessing of DVI and a cooperation of Perl, TeX and METAFONT.

**Method:** When preparing such a system, several groups of problems needed to be solved. In the present solution line breaking that considers the flexible gluish boxes is simulated using existing TeXdata type: glue. Optimum fit in TeX considers the content of the `\rightskip` register (it contains the glue that should be placed on the right margin of the line). If one breaks a paragraph into lines with the setting

```
\rightskip=0pt plus 0.052\hsize minus 0.047\hsize
```

(`\hsize` holds the width of the page) we get the same result as if we allowed all objects on the line to stretch/shrink by 5 per cent. These broken lines will be wrapped (using suitable macro) with marks,

<sup>6</sup> Even if we can see some future in a Multiple Master system that could bring the needed metaness to the Postscript world.

showing the beginning and the end of each line. For this, we can use the `\special` primitive that allows to write out arbitrary marks into the DVI file.

Proper positioning of the material on a broken line is the phase when we leave `TEX` and the subsequent work is done on the output DVI file that is analyzed using a Perl program. It is kept intact up to a place marked with the `\specials`, showing the line boundaries. The distance between these marks defines the space that should contain the objects and minimize the variance from the required grayness. The Perl script computes the widths of the characters; it uses heuristics to decide if the skips in the DVI file come from spaces or kernings (kernings are kept intact, spaces will be used for modifications). It figures out by how much it needs to vary the font and re-sets the line using the new font. If the necessary metrics is not available, it waits for another script to generate it.

The preparation of the variant-width fonts consists of automatic generation of the source texts in METAFONT. We base our procedure on the DC family of fonts. The Perl script takes as a parametr the font name, which defines which typeface it is derived from and how much it differs (for example `dcr8+3w.mf` is an 8 pt font extended by 3 per cent). We modify the source code of the original font accordingly (the value of its width parametr `\u#`, to be exact) and using METAFONT we generate new metric and bitmap files. The implicit attempt is to prepare 10 width variants with the width differences from the original font being  $\langle -5, -4, \dots, 4, 5 \rangle$  per cent. The actual typesetting is then done using the font that is closest to the one requested. To have a special exact font for each line of the document wouldn't be feasible from the computational point of view. In special cases, but only on request, we can generate exact width-variant.<sup>7</sup>

**Equivalence of the proper and implemented solutions:** The solution presented is in many respects only an approximation of systematic approach. The most visible simplifications include:

The fact that  $x\%$  of the width of the line is not equal to the sum of  $x\%$  of widths of the flexible boxes that built it. The equality holds only if we can vary the width of *all* objects involved. The first goal was to get the document that has all the spaces identical, so the fact that we consider the modification of spaces is not a benefit. On the other hand, a method that changes letters but not spaces, smells too artificial. More problematic is the fact that the

<sup>7</sup> For example for my favorite task to "typeset the headline to exactly fit the specified width".

line can contain parts that must not be modified at all. The user has to have a tool to specify that certain hand-tuned typographical construction should not be changed even by a micron. To improve the result in the rest of the paragraph, we recommend to the user to enclose these dangerous parts by a couple of `\special` marks that will inform the justifying algorithm that this material should be typeset without any change. We however encounter one nuisance: the assumption about modifying the material in the line by  $x$  per cent fails, if there is some unchangeable part that occupies substantial width of the line.

There is only one way how to check the badness of the created line. In this solution, we simulate the flexibility of the boxes by adding a glue to the `\rightskip` variable. The only possibility is to compute the badness using the standing formula  $|r|^3 \times 100$ , not distinguishing the white space and deformed characters. For the same reason, when searching for the optimum line-breaks in a whole paragraph, we are not able to consider some variant of `\adjdemerits` that would penalize adjaced lines with stretched and shrunked characters.

We do not analyze the content of the `\hboxes`. The `\hbox` in the DVI file is usually represented by another stack level. Because the presence of such a construct often marks something untypical (the difference of the actual and declared width of objects, explicit shifts of the reference point back and forth, complicated objects build by the user), we keep these parts of the page intact.

### Results of aesthetic experiments

The individual examples are provided with comments and numeric characteristics, but we strongly encourage the reader to do some aesthetic observation before taking author's prejudice into account. The empirical findings show that the perception of microtypographical effects differs extremely for different individuals; we would probably need to make great series of psychological and ergonomic tests to get any objective valuation.<sup>8</sup>

In an attempt to quantify results of the work by some algorithmic way, we have chosen following metrics:

**Badness:** is shown with some examples that compare the result with the result produced by `TEX`.

<sup>8</sup> All remarks of kind readers about bad headaches encountered as a result of endless excitation of visual nerves that try to seize the alphabet the same way they have known it (i.e. with constant width of letters), are greatly appreciated.



Unfortunately, vast majority of narrow columns shown below fall through into the third pass of the line-breaking algorithm (where the stretchability of the line is extended by `\emergencystretch`). In this pass,  $\TeX$  doesn't consider this added glue in its final compilation log. Badness, as measure of quality of the paragraph, is therefore insufficient. That is why we show another metrics.

**Percentage difference from the ideal width of the space:** Negative value means shrinking for example white spaces in *overflow boxes* have the width of  $-33$  per cent. The paragraphs after iterated line-breaking, include the following:

**Percentage difference of the width of the font used:** The positive values mean that we have used a font wider than the original, negative denotes shrinking. By looking at adjacent fonts that differ by a great amount (for example  $+5\%$  and  $-4\%$ ), we can review the critical spots of this way of typesetting, because here the eye of the reader encounters the biggest difference in the shape of the letters.

The following examples are prepared with the standard settings of the plain format (especially `\pretolerance100`, `\tolerance200`, `\hfuzz0.1pt`, `\adjdemerits10000`).

**The first example:** shows that  $\TeX$  has substantial problems when breaking lines into really narrow columns. The allowed tolerance limit of 200 is relatively tough; on the other hand, this is not a mathematical text with many unbreakable formulas, nor a technical text where terms not typical for the Czech language could confuse the hyphenation algorithm. The line-breaking is so hard that even after the third pass there are some *overflow boxes* left. The amount by which here the white spaces were stretched out in the solution with variable width font (second columns at the bottom right) indicates that even typesetting with five per cent ragged-right margin did not prevent the third pass or `\emergencystretch`. The fonts used here have, nevertheless, made it possible to decrease the stretch of the white spaces by an order of magnitude. When we compare the sixth lines (bottom right and left) we notice that the same material typeset with extended spaces (34%) changes into a line where they are shrunk just a little ( $-1\%$ ). This paradox solution was chosen because the choice of the best of 11 possible width variants has left us with less white space than would be needed in the optimum case. By increasing the number of variants in a font, it would be possible to decrease the scale of these non-optimal spaces.

The sixth and seventh lines show adjacent fonts that differ by nine per cent, truly one of the critical places on the page. Careful inspection of *m*'s reveals that the differences are very noticeable.

When we compare the last thirds of the paragraphs, the new system evidently wins. Not even inherent scepticism can keep the author from appreciating the regular grayness and more compact ending with the more reasonable length of the broken lines (see Figure 4).

**The second example:** brings 6 lines with *badness* 10000. The ragged-right version shows that the opening lines of a paragraph can be broken only very short. And really, even after iteration, the spaces on the second line are still very wide (124%). The left side brings little comfort because  $\TeX$  itself was unable to typeset this paragraph at all.

The last part of the paragraph offers two different variants of italics for comparison. Even a glimpse suggests that this typeface makes the modifications more visible than roman. The sixth and seventh lines of the text differ by eight per cent, but this is far less perceptible than those with italics. Individual typefaces obviously have different limits of painless modification (see Figure 5).

**The third example:** shows a typical way of using the system:  $\TeX$  could typeset the paragraph using `\emergencystretch` but the possible ways to do so were so few that even the freedom added by allowing a ragged-right margin did not change the solution chosen. Using the variant-width fonts we only adjust the spaces — we actually try to relax very loose lines. Because of the upper limit of the font modification (5 per cent) the widths of spaces still remain “unacceptable” (to compare this, see the ideal spaces in the ragged-right example). The advantage of this solution is the fact that most of the lines have undergone a similar type of modification — a rather stretched font. We do not see the compatibility problems as in other cases (see Figure 6).

**The fourth example:** shows that when  $\TeX$  encounters a truly unfeasible situation, as with very long words (and at the beginning of a paragraph, words shorter than  $2\backslash\hsize$  are enough), even a big value for `\emergencystretch` does not help. The glue added in the third pass is considered and typeset at the right margin of the text (see second line at the bottom left). Even words that are theoretically reasonably long can cause extreme problems — see the 206 per cent spaces on the third line.

We can find faults in the iterated solution but it comes very well from the comparison. The difficult

5	18%	Norská runová jména			Norská runová jména
1	11%	jsou pozdější, z doby, kdy			jsou pozdější, z doby, kdy
8	-14%	bylo ve Skandinávii použi-			bylo ve Skandinávii pou-
10000	-33%	váno už pouze 16 run, takže			žíváno už pouze 16 run,
10000	-33%	kompletní seznam jmen run			takže kompletní seznam
20	29%	této oblasti nemáme. Ná-			jmen run této oblasti ne-
2	-9%	zvy, které runám daly jiné			máme. Názvy, které runám
10000	-33%	germánské národy, neznáme			daly jiné germánské ná-
10000	-33%	vůbec (ačkoliv některá pís-			rody, neznáme vůbec (ač-
9	-14%	mena gótské abecedy mají			koliv některá písmena gót-
10000	-33%	k jménům run jistý vztah).			ské abecedy mají k jmé-
7	20%	Ze 16 <i>přeživších</i> norských			nům run jistý vztah). Ze
15	26%	run jich většina odpovídá			16 <i>přeživších</i> norských run
10000	-33%	jejich anglosaským protějškům;			jich většina odpovídá jejich
28	32%	a tuto podmnožinu pova-			anglosaským protějškům;
87	47%	žujeme za runy nejstarší,			a tuto podmnožinu pova-
8	-14%	pocházející z dávných ger-			žujeme za runy nejstarší,
-	8%	mánských dob.			pocházející z dávných ger-
					mánských dob.
5	18%	Norská runová jména	+1	5%	Norská runová jména
1	11%	jsou pozdější, z doby, kdy	+1	4%	jsou pozdější, z doby, kdy
154	57%	bylo ve Skandinávii pou-	+5	8%	bylo ve Skandinávii pou-
329	74%	žíváno už pouze 16 run,	+5	40%	žíváno už pouze 16 run,
2005	135%	takže kompletní seznam	+5	62%	takže kompletní seznam
32	34%	jmen run této oblasti ne-	+5	-1%	jmen run této oblasti ne-
10000	-33%	máme. Názvy, které runám	-4	-3%	máme. Názvy, které runám
768	98%	daly jiné germánské ná-	+5	52%	daly jiné germánské ná-
5	18%	rody, neznáme vůbec (ač-	+2	-1%	rody, neznáme vůbec (ač-
35	-23%	koliv některá písmena gót-	-2	-1%	koliv některá písmena gót-
169	98%	ské abecedy mají k jmé-	+5	25%	ské abecedy mají k jmé-
72	44%	nům run jistý vztah). Ze	+5	9%	nům run jistý vztah). Ze
10000	238%	16 <i>přeživších</i> norských	-2	-2%	16 <i>přeživších</i> norských run
4391	176%	run jich většina odpo-	-4	-9%	jich většina odpovídá jejich
3029	155%	vidá jejich anglosaským	+3	13%	anglosaským protějškům;
536	87%	protějškům; a tuto pod-	+3	3%	a tuto podmnožinu pova-
2884	153%	množinu považujeme za	+5	-1%	žujeme za runy nejstarší,
0	0%	runy nejstarší, pocházející	-1	-3%	pocházející z dávných ger-
2591	147%	z dávných germánských	0	0%	mánských dob.
-	-	dob.			

**Figure 4:** The first example. Top left: format plain. Bottom left: with additional `\emergency-stretch1em`. Top right: ragged-right lines (ideal spaces, `\rightskip` plus minus 5%). Bottom right: ragged-right lines adjusted with modified fonts.

10000	-33%	Ani při návodu nemůžeme od-			Ani při návodu nemůžeme
10000	-33%	dělovat to, co je správné, od toho,			oddělovat to, co je správné,
12	-16%	co je pouze zdánlivě správné, po-			od toho, co je pouze zdánlivě
10000	-33%	něvadž právě to není sporným stra-			správné, poněvadž právě to není
14	-17%	nám nikdy předem známo. Proto			sporným stranám nikdy pře-
29	33%	zde uvádím <b>úskoky</b> bez ohledu			dem známo. Proto zde uvádím
10000	-33%	na objektivní pravdu či nepravdu,			<b>úskoky</b> bez ohledu na objektivní
9	-14%	neboť to člověk sám nemůže bez-			pravdu či nepravdu, neboť to člo-
15	26%	pečně vědět. Teprve sporem má			věk sám nemůže bezpečně vědět.
143	56%	být pravda zjištěna. A pak při			Teprve sporem má být pravda
86	47%	každé debatě nebo argumentaci			zjištěna. A pak při každé debatě
10000	-33%	vůbec se musíme shodnout na ně-			nebo argumentaci vůbec se mu-
175	60%	čem, odkud – jakožto od prin-			síme shodnout na něčem, odkud
0	1%	cipu – hodláme otázku, o kterou			– jakožto od principu – hodláme
111	51%	jde, zkoumat: <i>Contra negantem</i>			otázku, o kterou jde, zkoumat:
190	64%	<i>principia non est disputandum.</i>			<i>Contra negantem principia non</i>
0	1%	(Nechť se nediskutuje s tím, kdo			<i>est disputandum.</i> (Nechť se nedis-
10000	-33%	popírá platnost základních pojmů			kutuje s tím, kdo popírá platnost
-	0%	a vět.)			základních pojmů a vět.)
273	69%	Ani při návodu nemůžeme	+5	17%	Ani při návodu nemůžeme
80	46%	oddělovat to, co je správné, od	+5	124%	oddělovat to, co je správné,
10000	-33%	toho, co je pouze zdánlivě správné,	+5	56%	od toho, co je pouze zdánlivě
0	-4%	poněvadž právě to není sporným	+1	-3%	správné, poněvadž právě to není
552	88%	stranám nikdy předem známo.	+5	114%	sporným stranám nikdy pře-
219	65%	Proto zde uvádím <b>úskoky</b> bez	+5	19%	dem známo. Proto zde uvádím
145	56%	ohledu na objektivní pravdu či	-3	-12%	<b>úskoky</b> bez ohledu na objektivní
175	60%	nepravdu, neboť to člověk sám	-3	-4%	pravdu či nepravdu, neboť to člo-
72	44%	dem může bezpečně vědět. Teprve	-1	2%	věk sám nemůže bezpečně vědět.
14	26%	sporem má být pravda zjištěna.	+5	27%	Teprve sporem má být pravda
725	96%	A pak při každé debatě nebo	+1	0%	zjištěna. A pak při každé debatě
1248	116%	argumentaci vůbec se musíme	+3	0%	nebo argumentaci vůbec se mu-
1342	118%	shodnout na něčem, odkud –	+1	1%	síme shodnout na něčem, odkud
218	64%	jakožto od principu – hodláme	+0	2%	– jakožto od principu – hodláme
179	60%	otázku, o kterou jde, zkoumat:	+5	15%	otázku, o kterou jde, zkoumat:
133	57%	<i>Contra negantem principia non</i>	+5	-11%	<i>Contra negantem principia non</i>
124	56%	<i>est disputandum.</i> (Nechť se ne-	-3	-13%	<i>est disputandum.</i> (Nechť se nedis-
0	-3%	diskutuje s tím, kdo popírá plat-	-2	-4%	kutuje s tím, kdo popírá platnost
-	0%	nost základních pojmů a vět.)	0	0%	základních pojmů a vět.)

**Figure 5:** The second example. Top left: format plain. Bottom left: with additional `\emergencystretch1em`. Top right: ragged-right lines (ideal spaces, `\rightskip` plus minus 5%). Bottom right: ragged-right lines adjusted with modified fonts.

25	-20%	Eristická dialektika je umění			Eristická dialektika je umění
9	-15%	diskutovat, a sice tak diskutovat,			diskutovat, a sice tak diskutovat,
10000	-33%	aby člověk vždy získal pravdu, tedy			aby člověk vždy získal pravdu,
0	6%	<i>per fas et nefas</i> . Lze totiž mít ve			tedy <i>per fas et nefas</i> . Lze totiž
87	47%	věci samé pravdu objektivně, a			mít ve věci samé pravdu objek-
10000	-33%	přece se člověk v očích posluchačů,			tivně, a přece se člověk v očích
143	56%	ba leckdy i ve svých vlastních,			posluchačů, ba leckdy i ve svých
10000	-33%	ocitne v nepravu – tehdy, vyvrátí-			vlastních, ocitne v nepravu –
147	56%	li odpůrce můj důkaz a platí-li			tehdy, vyvrátí-li odpůrce můj
26	31%	toto vyvrácení již také jako vy-			důkaz a platí-li toto vyvrácení
19	-19%	vrácení tvrzení samého, jež přece			již také jako vyvrácení tvrzení
0	-4%	lze dokazovat ještě jinak; v tako-			samého, jež přece lze dokazovat
1	-7%	vém případě je ovšem poměr pro			ještě jinak; v takovém případě
1	10%	odpůrce opačný: získá vrch, jak-			je ovšem poměr pro odpůrce
5	-12%	koli je objektivně v nepravu. Jak			opačný: získá vrch, jakkoli je
-	0%	je to možné?			objektivně v nepravu. Jak je to
					možné?
25	-20%	Eristická dialektika je umění	-2	2%	Eristická dialektika je umění
9	-15%	diskutovat, a sice tak diskutovat,	-1	-4%	diskutovat, a sice tak diskutovat,
259	68%	aby člověk vždy získal pravdu,	+5	23%	aby člověk vždy získal pravdu,
66	43%	tedy <i>per fas et nefas</i> . Lze totiž	+5	17%	tedy <i>per fas et nefas</i> . Lze totiž
21	29%	mít ve věci samé pravdu objek-	+4	1%	mít ve věci samé pravdu objek-
37	36%	tivně, a přece se člověk v očích	+5	7%	tivně, a přece se člověk v očích
0	3%	posluchačů, ba leckdy i ve svých	+0	3%	posluchačů, ba leckdy i ve svých
1199	114%	vlastních, ocitne v nepravu –	+5	72%	vlastních, ocitne v nepravu –
2150	139%	tehdy, vyvrátí-li odpůrce můj	+5	79%	tehdy, vyvrátí-li odpůrce můj
341	75%	důkaz a platí-li toto vyvrácení	+5	31%	důkaz a platí-li toto vyvrácení
338	75%	již také jako vyvrácení tvrzení	+5	30%	již také jako vyvrácení tvrzení
29	33%	samého, jež přece lze dokazovat	+3	5%	samého, jež přece lze dokazovat
364	77%	ještě jinak; v takovém případě	+5	32%	ještě jinak; v takovém případě
1960	134%	je ovšem poměr pro odpůrce	+5	93%	je ovšem poměr pro odpůrce
1478	122%	opačný: získá vrch, jakkoli je	+5	80%	opačný: získá vrch, jakkoli je
17	27%	objektivně v nepravu. Jak je to	+3	6%	objektivně v nepravu. Jak je to
-	-	možné?	0	0	možné?

**Figure 6:** The third example. Top left: format plain. Bottom left: with additional `\emergency-stretchlem`. Top right: ragged-right lines (ideal spaces, `\rightskip` plus minus 5%). Bottom right: ragged-right lines adjusted with modified fonts.

second line is solved using a font with a customized width. Here it even came out greater than the five per cent limit — when examining the relevant line and lines around it we find to our surprise that adjacent lines that differ by 7.52% do not cause a big problem (see Figure 7).

The variant width of the fonts can be used not only for improving narrow columns but for many other typographic purposes. This example shows an attempt to typeset a paragraph of reasonable width longer by one line (let's say we need it to achieve some higher visual goal). T<sub>E</sub>X will find such a solution but the price is an increased tolerance from 200 to 1635. Amazing rivers are one of its side effects.

Our solution reduces these annoying consequences. With a similar approach we can use variant-width fonts to improve paragraphs that need to be typeset with specific `\parfillskip` values. When typesetting such texts the loose lines can usually be seen, even in rather wide lines.

The task of typesetting a headline with given wording and size at a given width sometimes brings problems as well. To alter the font by several per cent is sometimes the smartest solution (see Figure 8).

Now that we have gone through the above series of examples, let us consider some thoughts and conclusions. Adjacent lines with big differences in the type of font modification are the most problematic ones. However, such narrow and short paragraphs cannot be broken in too many ways, so it's hard to select a solution with more compatible adjacent lines — by increasing the value of `\adjdemerits` we only increase the total demerits of paragraphs but we do not get a clear improvement. Much better results can be achieved, in this respect, with paragraphs that were stretched by force (positive `\looseness`, lower `\parfillskip`), where this method just “shrinks the white spaces” and in most places where the stretched fonts are used.

One note about the aesthetic evaluations of the examples: ordinary people usually “do not see anything” (but this result might be ambiguous, of course). On the other hand, people with some experience with micro-typographical effects only support the feeling that the readers' notions can differ significantly.

### And in the end...

First, let me apologize for the many motivation notes in the first part of this text. This article is the final word to a successfully completed thesis which nobody will ever re-open! So, the purpose was to make expert T<sub>E</sub>X-programmers feel that variant-width fonts are an interesting tool that would be nice to have. Anybody who wants to do their own experiments, both for inspiration when polishing difficult documents or searching for ideas for programming projects, can make use of scripts and macros available at <http://www.fi.muni.cz/~imladris/vlw>. Any modifications, improvements or even complex solutions to ideas presented here will certainly be appreciated by those T<sub>E</sub>Xists who (like me) enjoy the never-ending playing with typography.

### References

- [1] Miroslava Misáková. *Kvalitní typografie v počítačové sazbě (in Czech)*. diploma thesis on Faculty of Informatics, MU Brno, 1997.
- [2] Martin Davies. *The Gutenberg Bible*. The British Library Board, 1996.
- [3] URW Software Hamburg. *hz-program: Microtypography for advanced typesetting*. 1993.
- [4] Oldřich Hlavsa. *Typographia 1–3*. 1976–1986. In czech language.
- [5] Philip Taylor. Electronic typesetting and T<sub>E</sub>X: Book design for T<sub>E</sub>X users. In *Sborník zvaných přednášek SOFSEM '93*, 1993.
- [6] Philip Taylor. Pragmatický přístup k odstavcům. *T<sub>E</sub>Xbulletin*, 94(3), 1994.
- [7] Adolf Wild. La typographie de la bible de Gutenberg. *Cahiers Gutenberg*, Septembre 1995.

◇ Miroslava Misáková  
Faculty of Informatics, Masaryk  
University  
Botanická 68a,  
Brno, 602 00  
Czech Republic  
[imladris@fi.muni.cz](mailto:imladris@fi.muni.cz)  
[http://www.fi.muni.cz/  
~imladris/](http://www.fi.muni.cz/~imladris/)

10000	-33%		Tvůj příklad Llanfairpwllgwyn-				Tvůj příklad Llanfairpwll-
10000	-		gyllgogerychwyrndrobwlllantysil-				gwyngyllgogerychwyrndrobwll-
157	58%		iogogogoch (čili Llanfairu P.G.,				llantysiliogogogoch (čili Llanfairu
200	63%		jak se prý běžně zkracuje toto				P.G., jak se prý běžně zkracuje
1	-7%		město ve Walesu) je přece jenom				toto město ve Walesu) je přece
84	47%		okrajový. Němčina taky nestojí				jenom okrajový. Němčina taky
10	23%		a nepadá s tím, že se v ní ,pro-				nestojí a nepadá s tím, že se v ní
85	47%		středí pro vývoj aplikací' řekne				,'prostředí pro vývoj aplikací'
10000	-		,'Anwendungsentwicklungsumgebung'.				řekne 'Anwendungsentwicklungs-
							umgebung'.
1342	118%		Tvůj příklad Llanfairpwll-	+5	38%		Tvůj příklad Llanfairpwll-
10000	-		gwyngyllgogerychwyrndrobwll-	+5.52	0%		gwyngyllgogerychwyrndrobwll-
7030	206%		llantysiliogogogoch (čili Llan-	-2	-5%		llantysiliogogogoch (čili Llanfairu
1831	131%		fairu P.G., jak se prý běžně	+5	1%		P.G., jak se prý běžně zkracuje
40	37%		zkracuje toto město ve Walesu)	+5	9%		toto město ve Walesu) je přece
159	58%		je přece jenom okrajový. Něm-	+5	13%		jenom okrajový. Němčina taky
3	-9%		čina taky nestojí a nepadá s tím,	+5	0%		nestojí a nepadá s tím, že se v ní
132	54%		že se v ní ,prostředí pro vývoj	+5	102%		,'prostředí pro vývoj aplikací'
1	12%		aplikací' řekne 'Anwendungsent-	-2	17%		řekne 'Anwendungsentwicklungs-
-	-		wicklungsumgebung'.	0	0%		umgebung'.

**Figure 7:** The fourth example. Top left: format plain. Bottom left: with additional `\emergency-stretchlem`. Top right: ragged-right lines (ideal spaces, `\rightskip` plus minus 5%). Bottom right: ragged-right lines adjusted with modified fonts.

---

## Symposium o tolerantnosti

V hloubi šedesátých let, kdy se na české půdě začala do úvah a rozhovorů vracet některá zakázaná nebo zapomenutá té-

---

## Symposium o tolerantnosti

V hloubi šedesátých let, kdy se na české půdě začala do úvah a rozhovorů vracet některá zakázaná nebo zapomenutá té-

---

## Symposium o tolerantnosti

V hloubi šedesátých let, kdy se na české půdě začala do úvah a rozhovorů vracet některá zakázaná nebo zapomenutá té-

---

## Symposium o tolerantnosti

V hloubi šedesátých let, kdy se na české půdě začala do úvah a rozhovorů vracet některá zakázaná nebo zapomenutá té-

---

## Symposium o tolerantnosti

V hloubi šedesátých let, kdy se na české půdě začala do úvah a rozhovorů vracet některá zakázaná nebo zapomenutá té-

**Figure 8:** The fifth example. First four solutions:  $\TeX$ . The fifth: headline shrunk by using the narrower font.

## Software

### Editorial: Enc $\TeX$ , by Petr Olšák

Barbara Beeton

The motto introducing the following article, by Petr Olšák, describes Donald Knuth's original vision for  $\TeX$ , to be used mainly by him and his secretary.

Things haven't turned out that way.

Publishers of scientific and mathematical journals now produce them with  $\TeX$ , from  $\TeX$  manuscripts prepared by the authors, adhering to uniform guidelines — any divergence causes problems in automated production, with associated delays and costs. For this reason and others — joint authorship with manuscripts shipped back and forth, preprint archives on the Web, . . . — there is enormous peer pressure in much of the  $\TeX$  community (at least in the English speaking part of it) to use standard implementations and macro sets. Portability has become paramount. However, as printed languages accumulate more and more accented letters, or use different alphabets,  $\TeX$ -out-of-the-box becomes less and less usable without workarounds, sometimes elaborate ones.

That is the environment in which Petr Olšák finds himself, and he is trying to solve the problems that will make  $\TeX$  as easy to use for a computer-literate, Czech-language-literate novice as it is for a similarly well-educated English-speaking novice. How much more difficult it would be to learn a different natural language before you could use a computer tool created with that language in mind.

Three potential reviewers were asked to look at this article. Two refused outright, stating personal biases that might color their opinions. The third agreed, but warned of a similar bias, and made a strong (and successful) effort to overcome his prejudice. All three strongly agreed that the article should be published, as it forms a solid basis for discussion of this knotty problem.

Work is now going on to extend  $\TeX$  to (ultimately) 16bit encodings; the transition has to be planned with care, and it is going more slowly than anyone really wants. The stability of  $\TeX$  and the conservatism with respect to adding features have been proven out by the fact that  $\TeX$  is still in active use after nearly 20 years, while most other systems of this vintage are long dead.

I invite discussion in particular from implementors of  $\TeX$ , its successors and adjuncts. Space will be reserved in the next issues for this discussion.

---

**EncTeX — A little extension of TeX**

Petr Olšák

**Motto:**

Certainly, if I were a publishing house, if I were in the publishing business myself, I would have probably had ten different versions of TeX by now for ten different complicated projects that had come in. They would all look almost the same as TeX, but no one else would have this program—they wouldn't need it, they're not doing exactly the book that my publishing house was doing.

Donald E. Knuth, Prague, March 1996

— \* —

This article describes a simple change to TeX which makes it possible to manipulate the internal TeX vectors `xord` and `xchr`. These vectors are used to convert input encoding into TeX internal encoding and vice versa. For example, emTeX users (DOS or OS/2) know the so-called `tcp` tables which are used to set `xord` and `xchr` values. On the other hand UNIX users have no chance to reset these values once the TeX binary is compiled. The modification of TeX described below enables something similar to emTeX `tcp` tables. It is independent of the operating system. You can implement it in all TeX systems where the TeX program is compiled from WEB source files. I have tested my modification on UNIX systems with the `web2c` implementation of TeX.

There exist so far two options how to work with reencoding on UNIX `web2c` implementations. The first one is Skarvada's patch [3]. This solution implements several reencoding tables directly into the TeX source and the user cannot change these tables at TeX run time. The encoding table is selected by an environment variable. It is not saved into generated formats during `iniTeX`. I think, this solution is not so flexible.

The second option was implemented through `tcx` files by Karl Berry. It is commented out in current `web2c` sources with the following note: "`tcx` files are probably a bad idea, since they make TeX source documents unportable. Try the `inputenc` L<sup>A</sup>TeX package." I know the `xord/xchr` reencoding solution is not compatible with the `inputenc` package, but nevertheless I disagree with the note above. I have the following arguments:

- The `inputenc` package is a solution for L<sup>A</sup>TeX only, but TeX is used via other formats too.
- The `log` files and the terminal outputs are not legible if an overfull/underfull box of Czech



text is reported. The  $\hat{\hat{}}$  notation is absolutely funny. If section 49 of `tex.web` is changed (via `tex.ch` of course) in the following way: `(k<" ")or(k=invalid_code)`, the  $\hat{\hat{}}$  notation no longer occurs and the text is legible. But if `inputenc` is used with different internal  $\TeX$  encoding, the Czech sentences in `log` files are still not legible.

- The reencoding is an *implementation* problem, it is not the problem of a naive user, who must write `\usepackage[bla]{inputenc}`. He or she has no knowledge about encoding used in his/her OS. He or she perceives the command `\usepackage[bla]{inputenc}` as very mystical.
- If the  $\LaTeX$  document is sent via e-mail with MIME (or similar methods of transport), the reencoding is done by e-mail agents and the document is properly encoded for the OS of the receiver. The `\usepackage[bla]{inputenc}` is not automatically changed in the  $\LaTeX$  header, thus if the sender and the receiver work in different encodings — oops — Houston, we have a problem. I think, the reencoding must be solved by software for transportation between different OSes (e-mail agents or WWW servers/clients, for example) and this problem should not be solved in the  $\LaTeX$  header.
- The `inputenc` package sets active `\catcodes` to accented characters. So `Olšák` is expanded to `Ol\v s\'ak`, and therefore you cannot define the control sequence `\Olšák`. Accented letters have `\catcode=13` but `\catcode=11` is needed.
- Donald Knuth has implemented the `xord/xchr` vectors into  $\TeX$  to separate the encodings used in an OS and the internal  $\TeX$  encoding (because text fonts used in  $\TeX$  are independent of OS). Administrators can set up `xord/xchr` values during the  $\TeX$  WEB source state, but they usually don't do it. This is because there are many  $\TeX$  implementations with binary  $\TeX$  only. Even if the  $\TeX$  WEB source is available, setting `xord/xchr` is somewhat difficult for some administrators. But if the `xord/xchr` setting is possible during the  $\text{ini}\TeX$  state, the administrators will be more flexible to choose the right setting for their OS.
- I think Donald Knuth did not take into account the possibility to reencode during the expand processor state, as it is done by the `inputenc` package. Just consider that the `\uppercase`, `\lowercase` primitives do their work on *{balanced text}* before expanding using

`\uccodes` and `\lccodes`, which are used in the hyphenation algorithm *after expanding*.

My solution to the reencoding problem is more general than the `tcp` tables or the `tcx` files, because the  $\text{enc}\TeX$  tables are read during  $\text{ini}\TeX$  simply by using `\input` and are defined by  $\TeX$  macros. I have implemented three new primitives into  $\TeX$ : `\xordcode`, `\xchrcode` and `\xprncode`. They enable to set and read the values of `xord` and `xchr` vectors and to set the “printability” attribute of any character. A new quality is introduced: the `xord/xchr` vectors may be set independently. This opens great new possibilities.

### A technical introduction

The `xord` vector is 256 bytes long and stores the reencoding information for inputting characters from a text file into  $\TeX$ . The `xchr` vector has the same length and stores the reencoding information for outputting characters from  $\TeX$  to the terminal, to logs and to the text output files created through `\write`, but does not influence output to `dvi` files. These vectors are built into the program. All text information during input or output is reencoded by these vectors. If the input character has an external code  $x$  and an internal code  $y$  in  $\TeX$ , the `xord` vector must be set the following way: `xord[x] = y`. The rules for the output characters are as follows: If the character with internal code  $y$  is not assumed to be “printable” then the  $\hat{\hat{}}$ code  $y$  is output, otherwise the character with code  $x = \text{xchr}[y]$  is written.

### The $\text{enc}\TeX$ package

The installation of  $\text{enc}\TeX$  was tested on `web2c` version 7. If we have the WEB sources of this implementation of  $\TeX$  then the command

```
patch <enctex.patch
```

in the directory with `tex.ch` installs the  $\text{enc}\TeX$  package. After that, the new compilation of the  $\TeX$  binary (`make tex`) is needed. For more details see the `INSTALL.eng` file.

The patch changes the `tex.ch` file only. No other files including the C libraries are changed. The `make tex` command runs `tangle` on the main source file `tex.web` and on the changed change-file `tex.ch`. The Pascal source file `tex.p` is created and it is converted into C by the `convert` script and afterwards it is compiled into the run time binary `tex`.

Different  $\TeX$  implementations (than `web2c`) can have different `tex.ch` files, thus the simple `patch` command (for `web2c`) may not be applicable. For that reason the `enctex.ch` file is included. All

encTeX specific changes are described in this file. You can modify your `tex.ch` file manually using information from this file.

The encTeX modification is independent of the operating system and of the TeX implementation because all the changes are done in a WEB change file exclusively.

After encTeX is installed, you can set and read the values of `xord` and `xchr` vectors by new primitives `\xordcode` and `\xchrcode`. You can set the “printability” attribute of the character by the new primitive `\xprncode`. The syntax of all three new primitives is the same as the syntax of the `\lccode` or `\uccode` primitives. For example:

```
\xordcode"AB="CD \xchrcode\xordcode"AB="AB
\the\xchrcode200
sets xord[0xAB]=0xCD, xchr[xord[0xAB]]=0xAB
and, as a result of the second line, the value of
xchr[200] is printed in this example.
```

The new primitive `\xprncode` enables to set the “printability” attribute of the character. The character with internal code  $y$  is “printable” if and only if  $y \in \{32 \dots 126\}$  or `\xprncode`  $y > 0$ . For example, if we set `\xprncode255=1`, then the character with code 255 is “printable” and it will be printed as a character with the code `xchr[255]`. On the other hand, setting `\xprncode` to zero does not cause “non-printability” because the code of the character “~” is in  $\{32 \dots 126\}$ . It is a kind of “self-defence instinct” against an unsound user who could set all characters to be “non-printable” and the printing ability of the program may be lost. The `\xprncode` primitive can take any value from the range  $0 \dots 255$ , but remember the rule — “printable” if `\xprncode` is positive.

There is an important difference between the new encTeX primitives and well-known primitives like `\lccode` or `\catcode`. The new primitives represent internal TeX registers and are *global* under any circumstances. You can set their values in a group and these settings are not changed at the group end. I rejected the possibility of local settings (via the `eqtb` table) in order to achieve more efficient code.

The initial values, when iniTeX starts, are the following:

```
\xordcode i = i for i in {128...255},
\xchrcode i = i for i in {128...255},
\xprncode i = 0 for i in {0...31, 127...255},
\xprncode i = 1 for i in {32...126}.
```

The values `\xordcode`  $i$  and `\xchrcode`  $i$  for  $i \in \{0 \dots 127\}$  depend on the operating system. If the system is using the ASCII standard (very common)

then `\xordcode`  $i = i$  and `\xchrcode`  $i = i$  for all  $i$ . If the operating system is using another (obscure) encoding standard, then 95 printable ASCII internal codes from  $\{32 \dots 126\}$  are mapped into appropriate codes through corresponding changes in `\xordcode` and `\xchrcode` initial values.

The values of `\xordcode`, `\xchrcode` and `\xprncode` are stored in the `fmt` format file and they are restored during the run of the production version of TeX.

### About the ambiguous encoding

This subsection will describe some issues with `xord` and `xchr` resetting. Let us construct an example. Say, we need to map the character `\'a` (having code 129 in the OS, for example) onto the internal TeX code 128. So let `\xordcode129=128`, `\xchrcode128=129` and `\xprncode128=1`. At the same time the input code 128 is not mapped because it is never used in the Czech alphabet, for example. What if I get some file from Poland containing the character with the input code 128? This character is mapped to the code 128 (internal in TeX) but it is returned to `\log` as the code 129. That means that TeX is no longer able to distinguish between codes 128 and 129 on its input.

We will describe this phenomenon more exactly. Let's use mathematical terminology. Let  $X = \{0 \dots 255\}$  be a set of input codes and  $Y = X$  be the same set (from mathematical point of view) but let's use this letter for a set of the internal codes in TeX. Let  $Y_p \subseteq Y$  be a set of all printable characters. We claim:

$$Y_p = \{y; \text{\xprncode } y > 0\} \cup \{32 \dots 126\}.$$

It is obvious that the values of the `xchr` vector on the set  $Y \setminus Y_p$  don't influence the behavior of the program output.

Let  $I : X \rightarrow Y$  be the “input” function defined by the `xord` vector and  $O : Y_p \rightarrow X$  be the “output” function defined by the `xchr` vector. The initial values of `xord` or `xchr` ensure that  $I$  is bijective and  $O$  is injective and  $O = I^{-1}$  on  $Y_p$ . This feature gets lost after the first change of `xord` or `xchr` values. For example, let  $x \neq y$  and  $x \in X$ ,  $y \in Y_p$ . Let us make a simple transposition:

$$\text{xord}[x] = y, \quad \text{xchr}[y] = x. \quad (1)$$

Now, neither  $I$  function nor  $O$  function are injective! You can see, `xord`[ $x$ ] = `xord`[ $y$ ], and a similar equation holds for the `xchr` vector. The following condition must be fulfilled so that our functions are injective after applying transposition (1)  $n$ -times. The sequence  $x_0, \dots, x_n$  must exist with the following

properties:

$$x_0 = x, \quad x_1 = y \quad \text{and} \quad \text{xord}[x_i] = x_{i+1}$$

for all  $i \in \{0 \dots n-1\}$  and the equation  $\text{xord}[x_n] = x$  holds. Similar conditions must be fulfilled for the `xchr` vector. The problem is, that we apply the transpositions (1) only on a certain subset of  $X$  (for example on the printable characters or on accessible characters in a given encoding). Then we need not be surprised that as a result of our settings neither  $I$  nor  $O$  are injective functions and therefore equations  $O = I^{-1}$  or  $I = O^{-1}$  are senseless. The inversion exists only if the function is injective.

### The encoding tables

There are two types of encoding tables in `encTeX`. Both tables are `TeX \input` files with auxiliary macros. The files have the common extension `tex`. It is recommended to use these tables (or to modify them to your needs). Don't use the new primitives `\xordcode`, `\xchrcode` and `\xprncode` directly unless you exactly know what you are doing.

#### The first type of encoding tables

These tables are used during the `iniTeX` run. The values of `xord/xchr` are set symmetrically during the `\input`, the transposition (1) is used repeatedly for setting of the `xord/xchr` values. The resulting settings are stored in the format file using `\dump` and used in the production version of `TeX`.

These tables declare the relation of internal `TeX` encoding and the encoding used on the host operating system. For example, our system encoding is ISO8859-2 and internal `TeX` encoding is chosen by the Cork standard (called T1 in `LATeX`). In this case, the encoding table name is `i12-t1.tex`. It redefines the `xord` vector to map ISO8859-2 to T1 and the `xchr` vector to map T1 back to ISO8859-2. A part of the `i12-t1.tex` table is shown in Appendix 1 at the end of this article.

The first thing every encoding table does is input the macro file `encmacro.tex`, which consequently defines macros `\setcharcode`, `\expandto`, `\texmacro`, `\texaccent`. See the `README.eng` file in the `encTeX` package for detailed documentation of these macros.

Secondly, the internal encoding-specific macro is read. An input of the `t1macro.tex` file is performed in our example. The encoding-specific macros (such as accent definitions) are placed here. These macros solve similar issues as the `fd` files for text fonts in `LATeX`.

See Appendix 2 for the overview of all tables of the first type included in `encTeX`. You can list these files by

```
ls *-csf.tex *-t1.tex
```

`EncTeX` contains many files prepared for `iniTeX` for plain-variant formats. For example, the command

```
initex plain-i12-dc
```

generates the plain format with ISO8859-2 as the input encoding. This format name is `plain-i12-dc`, it includes the hyphenation table in T1 and uses the `dc` fonts for text. The content of the `plain-i12-dc.tex` file is shown in Appendix 3. The `\input` of Knuth's original `plain.tex` and the table `i12-t1.tex` is performed here.

#### The second type of encoding tables

The tables of the second type perform reencoding only on the input side of `TeX`, so the `xord` values are changed but the `xchr` values are not. The name convention identifies these tables: the symbols like `t1` or `csf` are missing in the name, because tables of this type deal with reencoding from one operating system standard to another and therefore they are not related to the `TeX` internal code.

For example, the table `1250-i12.tex` maps the input characters from CP1250 to ISO8859-2. The CP1250 encoding becomes a new input encoding and we assume that the first type of encoding table `i12-*.tex` was used in `iniTeX`.

We can use the table of the second type when a part of the input document (or the whole document) has a different encoding from the encoding used by our operating system. The table of the second type establishes the relationship to the input encoding declared in the table of the first type.

For example, the `i12-t1.tex` table was used in `iniTeX` and we have obtained a document in CP1250. We can write:

```
\input 1250-i12
\input document
\restoreinputencoding
The ISO8859-2 is restored here.
```

The double reencoding is active when the `document.tex` is read: firstly from CP1250 to ISO8859-2 and secondly from ISO8859-2 to internal `TeX` T1 encoding. The text is output to `log`, to the terminal and to `\write` files in ISO8859-2 encoding. The ISO8859-2 input encoding is restored after the `\restoreinputencoding` command.

Attention: it is impossible to reread the `\write` files when the table of the second type is active. If, for instance, the file `document.tex` includes some `\write` activities (for index, table of contents and so on), we have to read these auxiliary files *before* `\input 1250-il2` or *after* `\restoreinputencoding`. That is the reason why `\dump` (the format generation) is senseless while table of the second type is active.

### About compatibility

The `encTeX` extension successfully passes the TRIP test with two exceptions: 1. The banner is changed. 2. The number of multiletter control sequences is greater than in original `TeX` by three.

All changes of `TeX` which do not change the behavior of original `TeX` and only add some new primitives are backward compatible with Knuth's original `TeX`. It means that if we have written a document for original `TeX` and it is processed through extended `TeX` we will get the same results. Here is one exception though: the macro construction of the type `\ifx\xordcode\undefined` has to be missing in such a document. But, I guess, the probability of existence of such constructions in documents for standard `TeX` is equal to zero.

We have to say that it is possible to write new macros and documents in extended `TeX` which are not backward compatible with original `TeX`. This is a disadvantage of all extensions of `TeX`. We face this situation both if the extension adds new primitives directly (as in `encTeX` or `pdfTeX`, for example), as well as if the access to new primitives is hidden and may be initialized by some trick at the format generation time (as in `e-TeX` or `MLTeX`). The issue is, how many users will use the new primitives and who will be a supervisor for the standardization of these primitives.

In case of my `encTeX` package, I have no claim to standardize its primitives into newly developed `TeX`s. I have made this extension for my needs and if somebody likes it, he/she can use it realizing that his/her documents may not be backward compatible if he/she uses the new primitives directly. On the other hand, I meant my primitives to be used primarily while generating formats and not to be used directly in real documents. Thus the documents can still be backward compatible.

If an administrator of a multi-user system installs a `TeX` format using `encTeX`, he/she can call some table of the first type and prohibit the usage of the new primitives before `\dump`:

```
\let\xordcode=\undefined
\let\xchrcode=\undefined
```

```
\let\xprncode=\undefined
```

Thus users can't access the backward incompatible extension of `encTeX`. I recommend this setup for public sites. If `encTeX` is used this way exclusively, then the `xord` and `xchr` vectors work as was meant by the author of `TeX`: They filter operating system specific encodings into internal `TeX` encoding.

The question in my mind is why Donald Knuth did not introduce primitives similar to mine. He probably wanted all `TeX` macros to behave the same way on various implementations. In this case the direct access to `xord/xchr` values was not acceptable for him. We can use a condition like `\ifnum\xordcode'@='@`, thus our macro processing depends on whether or not the operating system adopts the ASCII standard.

The same macro behavior is not exactly reached in standard `TeX` either. We can `\write` a character into a file and we can reread this character in the next run. If we set `\catcode'^=12` before rereading then we can conditionally continue processing based on the "printability" attribute of this character in a given operating system.

### Conclusion

Everybody can modify the `TeX` source for his needs (see the quotation from the author of `TeX` at the beginning of my article). Modifying the `TeX` source is simpler than it looks. In my case, I perused [1] in the evening and reconsidered all issues. The next morning, I implemented my ideas into a computer and performed a couple of tests. And I wrote this article in the afternoon (in the Czech language; the English version took me considerably more time :-). The goal was reached quickly thanks to the very well documented program `TeX`.

### References

- [1] Donald Knuth. *TeX: The program*, volume B of *Computers & Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.
- [2] Petr Olšák. *The encTeX package*, <ftp://math.feld.cvut.cz/pub/olsak/enc tex>
- [3] Libor Škarvada. The patch for web2c `TeX` <ftp://ftp.muni.cz/pub/tex/local/cstug/skarvada>

◇ Petr Olšák  
Department of Mathematics  
Czech Technical University  
Prague, Czech Republic  
[olsak@math.feld.cvut.cz](mailto:olsak@math.feld.cvut.cz)

## Appendices

**Appendix 1:** A part of the table of the first type `il2-t1.tex`.

```
%%% The encoding table, v.Sep.1997 (C) Petr Ol\v s\'ak
%%% input: ISO-8859-2, internal TeX: Cork

\input encmacro \input t1macro

% (1) Czech/Slovak alphabet
%
%          input TeX   lc   uc   sf cat prn          sequence
\setcharcode "C1 "C1 "E1 "C1  999 11  1  \texaccent \'A
\setcharcode "E1 "E1 "E1 "C1 1000 11  1  \texaccent \'a
\setcharcode "C4 "C4 "E4 "C4  999 11  1  \texaccent \"A
\setcharcode "E4 "E4 "E4 "C4 1000 11  1  \texaccent \"a
\setcharcode "C8 "83 "A3 "83  999 11  1  \texaccent \v C
\setcharcode "E8 "A3 "A3 "83 1000 11  1  \texaccent \v c
...
\setcharcode "A4 "1F  0   0   0  15  0  % =o=, not accesible
\setcharcode "A7 "9F  0   0   0   0  12  1  \texmacro \S
\setcharcode "D7 "03  0   0   0   0  13  0  \expandto {$\times$}
\setcharcode "F7 "07  0   0   0   0  13  0  \expandto {$\div$}
```

**Appendix 2:** A list of tables of the first type.

File name	input encoding	internal T <sub>E</sub> X encoding
<code>il2-csf.tex</code>	ISO8859-2	CS-font
<code>kam-csf.tex</code>	Kamenických	CS-font
<code>1250-csf.tex</code>	CP1250, MS-Windows	CS-font
<code>852-csf.tex</code>	CP852, PC Latin2	CS-font
<code>mac-csf.tex</code>	MAC-CZ, Macintosh	CS-font
<code>il2-t1.tex</code>	ISO8859-2	T1 alias Cork
<code>kam-t1.tex</code>	Kamenických	T1 alias Cork
<code>1250-t1.tex</code>	CP1250, MS-Windows	T1 alias Cork
<code>852-t1.tex</code>	CP852, PC Latin2	T1 alias Cork
<code>mac-t1.tex</code>	MAC-CZ, Macintosh	T1 alias Cork

The CS-font encoding and T1 are commonly used as the internal encoding of T<sub>E</sub>X for the Czech language. CP1250 is commonly used in MS Windows systems, ISO8859-2 in UNIX, CP852 or Kamenických encodings are used in DOS and MAC-CZ is used in Macintosh systems.

**Appendix 3:** The content of the `plain-il2-dc` file.

```
\input noprefnt % re-defines \font: \font\preloaded is ignored
\input plain    % format Plain
\restorefont    % original meaning of primitive \font
\input dcfnts   % loads text-style dc fonts
\input il2-t1   % input: ISO8859-2, internal TeX: Cork
\input hyphen.lan % czech / slovak hyphenation pattern
\input plaina4  % \hsize and \vsize for A4
\everyjob=\expandafter{\the\everyjob
  \message{The format: plain-il2-dc <Sep. 1997>.}
  \message{The cm+dc-fonts are preloaded and A4 size predefined.}}
\dump
```

---

## ConcTeX: Generating a concordance from TeX input files

Laurence Finston

### Abstract

ConcTeX is a package that generates a concordance from a plain TeX file. It has been designed specifically for books containing transcriptions of medieval manuscripts, such as facsimile editions, but it can be adapted for other types of material. ConcTeX consists of TeX code in the file `conctex.tex` and a Common Lisp program in the file `conctex.lsp`. The main advantage of ConcTeX is that the same TeX files are used for typesetting and for generating the concordance. It also performs alphabetization of arbitrary special characters and lemmatization. ConcTeX illustrates the power of using TeX and Lisp in combination. It is available under the normal conditions applying to free software.

### Introduction

Designing and typesetting a facsimile edition of a medieval manuscript is a formidable challenge. The plates with the facsimile itself will be photolithographs or, in books of the finest quality, collotypes, and will therefore require no typesetting. However, facsimile editions invariably contain parts which must be typeset, such as an introduction, a transcription and a concordance.

Manuscript transcriptions must be carefully designed and they generally require frequent font changes and numerous special characters. Typesetters, if there are any left, are unlikely to be able to read the language of the manuscript, and each manuscript has its own peculiarities of language and orthography, so the process of proofreading and correction is even more difficult than for ordinary books. Nowadays, authors or editors are often expected to supply camera-ready output to the publisher. This often means a printout from one of the popular word-processing packages, which are incapable of producing typesetting of sufficient quality for the task. In today's market, using mechanically set lead type is prohibitively expensive and the number of publishers willing to typeset difficult copy in this way decreases every year.

The task becomes even more daunting when a concordance is desired.<sup>1</sup> Compiling a concordance by hand requires so much labor that it is no longer economically feasible. Today, concordances are

---

<sup>1</sup> A concordance is a complete listing of all words occurring in a manuscript, lemmatized, with the main forms of the lemmata sorted alphabetically.

generated by means of computer programs, but, until now, this has required preparing a specially formatted file containing the transcript. Since a separate file, or a typescript, was used for typesetting, changing either the transcript or the concordance necessitated making the corresponding change in the other. In some cases, a conversion program could be used to automate this process. Where this was not possible, either because a typescript was used or no conversion program was available, every change had to be made in two places by hand: an editorial nightmare.

ConcTeX is a package that attempts to solve these problems. It includes a file of TeX code, `conctex.tex`, containing tools for designing a facsimile edition of a manuscript, and a program written in Common Lisp, `conctex.lsp`,<sup>2</sup> for generating a concordance from the TeX input files. The concordance program performs lemmatization and alphabetization of arbitrary special characters, and its output is another TeX input file containing the concordance. ConcTeX is designed for producing facsimile editions of manuscripts, but it can be adapted for other types of material.


Using ConcTeX makes it possible to benefit from the typographic capabilities of TeX and METAFONT, of which readers of *TUGboat* need not be convinced. Apart from this, its most significant advantage is that the TeX input files are used both for typesetting and for producing the concordance, so that any changes in the input files are automatically reflected in both the printed output and the concordance.

ConcTeX is not for novices. A certain amount of TeXpertise and knowledge of Lisp are necessary to use it successfully. The version I describe in this article has been designed for a particular project. Any other project will require some customization. Many of the individual features of ConcTeX as described here are the result of decisions regarding the design of a particular book. Other books will require other decisions. I have programmed most of the routines in a general way, so that details can be changed while the basic operation of ConcTeX remains the same.

Both the TeX code in `conctex.tex` and the Lisp program `conctex.lsp` use some fairly advanced techniques, so readers may find this article somewhat difficult. I expect the description of the Lisp program will present the most problems, because Common Lisp is likely to be unfamiliar to

---

<sup>2</sup> Technically, it is incorrect to speak of "the program" `conctex.lsp`. In Lisp a program is not a file. It is, however, convenient to refer to the file `conctex.lsp` as "the program".

most  $\TeX$  users.<sup>3</sup> Some of the more difficult and subtle points are in the footnotes; for others, I've borrowed Prof. Knuth's "dangerous bend" sign:  I sometimes use footnotes and "dangerous bend" paragraphs to refer to topics that are introduced later in the article. Most readers will want to skip these paragraphs on the first reading.

Generating a concordance is admittedly a special application; most  $\TeX$  users won't want to do such a thing. However, the techniques for extracting information from  $\TeX$  input files, described in this article, are of general applicability.

In the following description and examples, I use two transcriptions of Icelandic manuscripts of the 13<sup>th</sup> century, AM 234 fol and Holm perg 11 4<sup>o</sup>, each containing a *vita* of the Virgin Mary and a collection of miracles. I wish to thank Dr. Wilhelm Heizmann, my *Doktorvater* (dissertation advisor), who prepared the transcriptions, for permission to use them in this article.<sup>4</sup>

In the following, many phrases have special meanings. They are all explained at their first appearance, but to avoid confusion, they are listed in a glossary on page 402.

**Installation.** In order to use  $\text{Conc}\TeX$ , the file `conctex.lsp`, containing the Lisp program, must be in your working directory, and `conctex.tex`, containing  $\TeX$  code, must be either in your working directory or in a directory in  $\TeX$ 's load path. If you don't know what this is, or how to change it, ask your local  $\TeX$  wizard, or just put the file in your working directory. The line

```
\input conctex
```

must be at the beginning of your input file.

**Why not  $\LaTeX$ ?**  $\text{Conc}\TeX$  is designed for use with  $\text{plain}\TeX$ . It is theoretically possible to adapt it for use with  $\LaTeX$ , but I recommend against doing so. I use  $\text{plain}\TeX$  in preference to  $\LaTeX$  in  $\text{Conc}\TeX$  for the following reasons:

1. Making significant changes to one of  $\LaTeX$ 's pre-defined formats is difficult and time-consuming, whereas programming a format based on  $\text{plain}\TeX$  is relatively easy.
2.  $\LaTeX$  enforces a rigid structure on formats. This makes sense for formats that are intended

<sup>3</sup> The standard introduction to Lisp is Patrick Henry Winston and Berthold Klaus Paul Horn's *LISP*. Once you know how to program in Lisp, Guy L. Steele's *Common Lisp, The Language* is the one indispensable reference.

<sup>4</sup> I would also like to thank Günter Koch and Jürgen Hattenbach of the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany for help above and beyond the call of duty.

to be used by many people, many of whom may have minimal knowledge of how  $\TeX$  works. However, it is simply not worth the time and effort to write a  $\LaTeX$  format for a single book, when a  $\text{plain}\TeX$  format is just as good and can be written in a fraction of the time. And every book (or series) deserves its own design.

3.  $\LaTeX$  loads various files by default, and signals an error if it doesn't get them. Some of the things in these files might not even be necessary, but you've still got to wait until they're loaded.
4. The more macros you use, the more likely it is that they will start to interfere with each other. The problem is even worse when using a large package with macros you a) don't need and b) don't understand.  $\LaTeX$ 's macros are very difficult to understand because pieces of them are scattered all over the place. For this reason, it can be very difficult and frustrating trying to get  $\LaTeX$  to stop doing something you don't want it to do.  $\text{Conc}\TeX$  changes the `\catcode` of several characters and includes a large number of macros. Therefore, the likelihood is great that there would be interference between  $\text{Conc}\TeX$  and  $\LaTeX$ .

### The $\TeX$ input file

Multiple input files can be used for typesetting a document and generating a concordance, so the document can be divided into several files in the customary way. In the following, I will assume, for simplicity's sake, that there is only one input file. This file can have any name within reason.

Like any other  $\TeX$  input file, an input file for  $\text{Conc}\TeX$  will contain text, control sequences for typesetting and perhaps comments (using `%`). But it will also contain Lisp code used by the program `conctex.lsp`. Therefore,  $\TeX$  input files used for generating a concordance are subject to greater restrictions than is ordinarily the case with  $\text{plain}\TeX$ . Some parts of the input file will only be used by  $\TeX$ , some will only be used by `conctex.lsp`, and some will be used by both, or neither. Many of the complications of  $\text{Conc}\TeX$  have to do with making  $\TeX$  and/or Lisp ignore items in the input file.

**Environments.**  $\text{Conc}\TeX$  changes some category codes and redefines some control sequences in order to format the transcription correctly. This can make it difficult to type in "normal" text using  $\text{plain}\TeX$ 's familiar conventions. It might be useful to do this if sections of transcription alternate with

sections of commentary. The macro `\plain` defines an environment where `\catcodes` and macros are reset to their normal values. This environment ends with the macro `\endplain`.

In the `\trans` environment, which is the default, the `\catcodes` of characters and macro expansions are set to the values needed for *transcription lines*, i.e., the lines that contain the actual transcription. The macros `\endplain` and `\endtrans` are defined like this:

```
\let\endplain=\trans
\let\endtrans=\plain
```

The changes made by `\trans` and `\plain` (and `\endplain` and `\endtrans`) are global. A `\trans` in the input file need not be matched by an `\endtrans`, but `\plain` and `\endplain` must be matched, otherwise they will wreak havoc in `conctex.lsp`.

Another environment is used for *commentaries*, which are described below. Commentaries also reset some category codes and macros, but the commentary environment is not identical to the `\plain` environment. Commentaries will usually contain material similar to that in the transcription itself, whereas material in the `\plain` environment should be formatted differently. The `\catcodes` of several characters needed in math mode are also reset by the token list `\everymath`.

⚡ A line beginning with `\trans` or `\endtrans` in an input file will be ignored by `conctex.lsp`, but `\plain` and `\endplain` are replaced by `*`, so `conctex.lsp` treats text between `\plain` and `\endplain` as a commentary.

⚡ Having the `\trans` environment be the default isn't hard-wired into `ConcTeX`; however, the definitions of `\-` and `-` assume that `\trans` is the default, so this will need to be fixed if you want `\plain` to be the default environment.

⚡ In both the `\plain` environment and within commentaries, `-` is reset to `\catcode 12` and `\-` is used for discretionary hyphens, because line breaking is not performed explicitly, but rather by `TeX`'s line breaking routine.

**Transcription lines** are the lines in the input file that contain the text of the transcription itself. They are processed in whole or in part by both `TeX` and `conctex.lsp`. In order to do its job, `ConcTeX` redefines the `\catcode` of several characters. Each of these changes is explained in its proper place, but there is a list for reference on page 402. Transcription lines are formatted according to the settings in the `\trans` environment, and processed

by `conctex.lsp`. Apart from the text, they may contain items like comments, commentaries, and certain macros.

**Comments and commentaries.** `ConcTeX` makes a distinction between “comments” and “commentaries”. A comment can be a normal `TeX` comment using `%`, but it can also be code that looks like this:

```
\begincomment{This is a comment.}
\endcomment
```

The format defined in `conctex.tex` uses a conditional (defined with `\newif`) called `\ifdraft`. Whenever `\drafttrue`, that is to say, whenever `\ifdraft≡\iftrue`, certain things are done which are useful for editing purposes, but which aren't done for the final draft. One of these things is printing out comments. If `\drafttrue`, this line:

```
helgum m{\oe}nnum
\begincomment{This is a comment}%
\endcomment {\ae}{\dh}r i.~helgari
```

⇒

```
helgum mœnnum * This is a comment *
æðr i. helgari
```

If `\draftfalse`, it yields

```
helgum mœnnum æðr i. helgari
```

A commentary, on the other hand, is text which should be processed by `TeX` and appear in the output, but should be ignored by `conctex.lsp`, i.e., not be used for generating the concordance. This is for editorial remarks within the transcription itself. Commentaries can be coded in several different ways. Usually, a commentary begins and ends in `*`.<sup>5</sup>

```
\catcode'\*=9
...
herbergi heilag{\slong} anda.~%
* This passage is particularly
interesting * {\oe}llvm\
helgum m{\oe}nnum {\ae}{\dh}r
i.~helgari\
{\tirok} haleitari.~er komin
at k{\ydot}n\
```

⇒

```
herbergi heilagf anda. This passage is particu-
larly interesting œllvm
helgum mœnnum æðr i. helgari
a haleitari. er komin at kÿn
```

<sup>5</sup> The control symbol `\` in the following example is used for breaking the lines. It's explained on page 381. The letter `{\slong}` → `f`, used in the following examples, is an alternative form of “s”. It is simply the “f” from Computer Modern Roman, with the crossbar removed.



In this example, T<sub>E</sub>X simply ignores the character \*, because its `\catcode` has been changed from 12 (other) to 9 (ignored), so the commentary is printed in the same font as the transcription. If I want the commentaries to be printed in a different font, I can code something like this:

```
\catcode'\*=\active
\font\ssf=cmss10

\def\startcommentary{\begingroup\ssf}
\let\finishcommentary=\endgroup
\let\docommentary=\startcommentary

\def*{\docommentary
\ifx\docommentary
\startcommentary
\global\let\docommentary=%
\finishcommentary
\else
\global\let\docommentary=%
\startcommentary
\fi}
```

This makes \* an active char whose expansion switches back and forth between `\startcommentary` and `\finishcommentary`. The example above then comes out looking like this:

```
herbergi heilagf anda. This passage is particularly
interesting œllvm
helgum moennum æðr i. helgari
a haleitari. er komin at kÿn
```

If you want \* to do more than just change the font, you can put more code into the expansions of `\startcommentary` and `\finishcommentary`. The macros `\begincommentary` and `\endcommentary` are both `\let` to \*, so using them is exactly the same as using \*. They can be useful for marking longer commentaries, e.g.,

```
herbergi heilag{\slong} anda.~{\oe}llvm\
\begincommentary
\<This commentary is so long that it's
nice to mark it with
{\tt\string\begincommentary}
and {\tt\string\endcommentary}
to make it easier
to see in the input file, since \* and \*
might be easy to overlook.\>
\endcommentary\space
helgum m{\oe}nnum {\ae}{\dh}r
i.~helgari\
{\tirok} haleitari.~er komin
at k{\ydot}n\
```

```
herbergi heilagf anda. œllvm
\begincommentary
\endcommentary
\<This commentary is so long that it's nice to mark
it with \begincommentary and \endcommentary
to make it easier to see in the input file,
since * and * might be easy to overlook.\>
helgum
moennum æðr i. helgari
a haleitari. er komin at kÿn
```

The explicit `\space` following `\endcommentary` is necessary, because ordinary spaces following `\endcommentary` would just be swallowed up in the usual way. It would be easy enough to change by redefining `\finishcommentary`, so that it always inserts a space into the current list.

```
\def\finishcommentary{\endgroup\space}
```

However, it might be desirable to have commentaries end within a word sometimes.


#### Ignored lines.

- Lines that begin with % are ignored both by T<sub>E</sub>X and `conctex.lsp`. (A % in the middle of a line causes both T<sub>E</sub>X and `conctex.lsp` to discard the rest of the line.)
- Entirely blank lines are processed normally by T<sub>E</sub>X (the second `\return` in a row is converted to `\par`), and ignored by `conctex.lsp`.
- Lines that begin with \ are treated in the normal way by T<sub>E</sub>X and ignored by `conctex.lsp`, with a few exceptions. This makes it possible to include *undelimited macros* (control sequences that are not surrounded by braces) in the input file without worrying about how the Lisp program is affected by them. These can be skips, font changes or other commands. Beginning a line with `\relax` or `\empty` makes it possible to type anything in the input file and have `conctex.lsp` ignore it.



Certain undelimited macros, such as `\lineno` and macros like `\putontop` and `\overstroke` that are used for text, do not cause `conctex.lsp` to ignore a line, so they can appear at the beginning of text lines. Other exceptions may be programmed.

**Evaluated lines.** Lines in the input file can contain Lisp code to be evaluated by `conctex.lsp`. The first non-blank character in these lines must be the commercial “at” symbol, @. The `\catcode` of @ has been changed to 14 (comment), so T<sub>E</sub>X ignores these lines. An evaluated line should contain a complete balanced expression (s-expression or sexp); multiline Lisp expressions are not permitted in the input file. If an @ is not the first non-blank character in a *text line*, both T<sub>E</sub>X and `conctex.lsp` discard the rest of the line following the @, and Lisp does not evaluate it.

 The most common use of evaluated lines is to reset the *text position* for a new leaf, side or column.

```
@ (set-position "28va")
```

They can also be used for adding occurrences for words which  $\text{T}_{\text{E}}\text{X}$  can't format in the normal way, perhaps because a word is written vertically and extends over multiple lines (see page 398).

```
@ (add-occurrences "drotning" "1va~1-7")
```

**The character ¥.** The program `conctex.lsp` considers all letters (`\catcode 11`) and some characters of type “other” (`\catcode 12`) in text lines as “word elements”. Some “other” characters are considered “word separators”: in particular, blanks and punctuation. The character ¥ (decimal 165, octal 245, hexadecimal A5)<sup>6</sup> is a special kind of word separator: it is ignored by  $\text{T}_{\text{E}}\text{X}$  (`\catcode 9`). The program `conctex.lsp` will ignore most lines that begin with an *undelimited macro*, but if the line begins with ¥, `conctex.lsp` will process it.

```
¥\vskip12pt DROTNING himins ok
iar{\dh}ar ...
```

The character ¥ may look different on your terminal.<sup>7</sup> However it looks, it should always be used rather than `^^a5` in your input file;<sup>8</sup>  $\text{T}_{\text{E}}\text{X}$  will recognize `^^a5` as referring to a single token, but `conctex.lsp` will treat it as a string of 4 characters. It's never necessary to use ¥, it is merely a convenience.

**Curly braces.** As far as `plain`  $\text{T}_{\text{E}}\text{X}$  is concerned, the characters `{` and `}` are simply set to categories 1 and 2, beginning and end of group, respectively. They can be set to other categories, and other characters can be set to categories 1 and 2. `ConcTEX` doesn't support this generality; the `\catcodes` of `{` and `}` should not be changed

<sup>6</sup> It's sometimes convenient to know the decimal, octal or hexadecimal notation for an integer in another of these radices. `ConcTEX` includes a lagniappe, a C program that converts between decimal, octal and hexadecimal integers easily and quickly, and several Emacs-Lisp functions for calling this program from within Emacs.

<sup>7</sup> If you're using Emacs, you can enter any character by typing `Control-q` followed by an octal integer, so I can type `Control-q 245` to get ¥. You may get another character, though, depending on your editor, operating system, etc. A better way to type in special characters is to define a key sequence or an abbreviation to do it. There's more about this in the documentation supplied with `ConcTEX`.

<sup>8</sup> Any character can be represented in a  $\text{T}_{\text{E}}\text{X}$  file as `^^` followed by two lowercase hexadecimal digits (*The T<sub>E</sub>Xbook*, p. 45). If necessary, `conctex.lsp` can be made to recognize this notation.

and other characters should not be used in their place. The reason for this is that the program `conctex.lsp` uses `{` and `}` explicitly in strings. It would be possible to change this and have a general mechanism for recognizing a beginning-of-group and an end-of-group character, but I did not consider that this was worthwhile.

Braces are used in *transcription lines* for their normal purpose: to delimit macros and their arguments.  $\text{T}_{\text{E}}\text{X}$  normally permits “unmotivated braces”, i.e., braces that are typed into the input file for no particular reason.

```
Th{is} line has {unm}otiv{ated} br{ace}s.
```

⇒

This line has unmotivated braces.

The unmotivated braces will only have an effect if they separate parts of a ligature, as in `waf{f}le` → waffle, or prevent kerning, as in `{V}A` → VA. Sometimes, of course, as in `{shelf}ful` → shelfful,<sup>9</sup> this is useful, but then the braces aren't unmotivated. `ConcTEX` does not permit them, and the function `letter-function` in `conctex.lsp` will signal an error when it finds an unmotivated `}`. If cases like “shelfful”, where the word looks better without the ligature, are desired, they must be accounted for in `conctex.lsp`.

**Changing fonts.** Manuscript transcriptions often require frequent font changes, sometimes within a single word. For example, editorial emendations may be printed in italics: `prestr` (Engl. “priest”), and perhaps enclosed in brackets, too: `p[re]str`. Some letter forms may be represented in the transcription as small capitals, such as the “R” in `p[re]str`. Different fonts may be used for various other purposes, too, according to the book design. For instance, initials, large and/or decorative letters, and letters written in a different color ink in the manuscript may all be indicated by a special font in the transcription. Font changes are handled in the normal way by  $\text{T}_{\text{E}}\text{X}$  and discarded by `conctex.lsp`. Font changes can extend over multiple transcription lines.

**Special characters and other macros.** A fundamental decision in editing a manuscript transcription is, to what degree the transcription should correspond to the actual appearance of the manuscript, e.g., how special characters are represented, whether abbreviations are expanded, etc. With  $\text{T}_{\text{E}}\text{X}$  and `METAFONT`, it is possible to imitate the appearance of the manuscript to a greater degree than is possible with other methods; however it is up

<sup>9</sup> *The T<sub>E</sub>Xbook*, pp. 19 and 306.

to the editor and the book designer whether to make use of this capacity or not. I believe that a transcription will usually be of greater interest if it's not normalized, and if the abbreviations are not expanded. No matter what style of transcription is chosen, a wide range of special characters is usually required.

In T<sub>E</sub>X, special character macros can be typed with or without enclosing braces, e.g., the word “ætt” (Engl. “a quarter of the heavens, one’s family”)<sup>10</sup> can be coded as “{\ae}tt” or “\ae<sub>tt</sub>”, or even “\ae<sub>tt</sub>” since spaces following a control word are ignored. The second and third alternatives are impractical even under normal circumstances, because it is unclear in the input file that “\ae” and “tt” belong together. In ConcT<sub>E</sub>X, however, all special character codings must be enclosed in braces (“{\ae}tt”). T<sub>E</sub>X will handle “\ae<sub>tt</sub>” and “\ae<sub>tt</sub>” in the normal way, but `conctex.lsp` will signal an error when it reads “\ae” without enclosing braces.<sup>11</sup> All macros used in transcription lines must be accounted for in `conctex.lsp`. Others will cause an error.

In most medieval manuscripts, words are often abbreviated. There are several methods of abbreviation. One is to write some of the letters of a word and put a stroke over them, like “mm” for “mōnnum.” It may be desirable to expand these abbreviations in the transcription. One way of doing this is to print out the characters that do not appear in the manuscript, but in italics and underlined, e.g., “mōnnum”. (Other solutions may be used according to the book design.) The macro `\ustroke` is used to do this:

```
m{\ustroke{{\ohook}nnu}}m
⇒
mōnnum
```

You can type “`m{\ustroke{{\ohook}nnu}}m`” or “`m\ustroke{{\ohook}nnu}m`”, i.e., the macro `\ustroke` can be delimited or undelimited. Either way, `conctex.lsp` discards the macro and its braces, and the argument is treated as part of the word, so in this example, “mōnnum” is printed in the output and “mōnnum” will appear in the concordance (under the main form “maðr”, Engl. “man”).

Since such abbreviations occur frequently, the `\catcode` of the underline character `_` (decimal 95, octal 137, hexadecimal 5F) has been reset to `\active` and `\let` to `\ustroke`. So now you can type “`m_{\ohook}nnu}m`” to get “mōnnum”, which

<sup>10</sup> CLEASBY-VIGFUSSON, p. 760.

<sup>11</sup> It is `letter-function` that signals the error, when it reads the `\`.

makes the input file somewhat easier to type and read. The `\catcode` of `_` is reset to 8 (subscript) in math mode, so it's available for making subscripts, and also in the `\plain` environment, so that it's possible to load files with the character `_` in their names. But it's not reset in commentaries, which might very well want to use `_` for `\ustroke`.



The macro `\ustroke` is defined as follows:

```
\def\ustroke#1{%
\def\subustroke{\leavevmode
\ifx\next.% It's a period
\setbox0=\hbox{{\it#1}}\else
\ifx\next,% It's a comma
\setbox0=\hbox{{\it#1}}\else
% It's neither a period nor a
% comma
\setbox0=\hbox{{\it#1/}}}%
\fi\fi
$\underline{\box0}$}%
\futurelet\next\subustroke}
```

Since the underlined text is put in italics, it's nice to have `\ustroke` insert the italic correction (`\)` automatically, if and only if `\ustroke`'s argument is followed by something other than a period or a comma. In order to find this out, it's necessary to peek at the following token, using `\futurelet`. If `\ustroke` uses a non-slanted font, it can be defined more simply. It uses the `\underline` macro, which is available only in math mode. This makes the underlines go below the bottom of the lowest character in `\ustroke`'s argument.

```
_ {ypq} _ {abc}
_{\ehook}{\ohook}{\oehook}}
⇒
ypq abc eqq
```

It might be nicer to have the underlines all at the same depth, preferably close to the baseline, but unfortunately, this doesn't turn out to look very good. If `\ustroke` is defined like this:

```
\def\ustroke#1{%
\def\subustroke{\leavevmode
\ifx\next.% It's a period
\setbox0=\hbox{\it#1}\else
\ifx\next,%
% It's a comma
\setbox0=\hbox{\it#1}\else
% It's neither a period
% nor a comma
\setbox0=\hbox{\it#1/}\fi\fi
% This makes a .25pt rule.
\hbox to 0pt{\vrule height -.8pt
```


```

        depth 1.05pt
        width \wd0\hss}\box0}%
\futurelet\next\substroke}
then
  _{ypq} _{abc}
  _{{\ehook}{\ohook}{\oehook}}
⇒
  ypq abc eoo

```

I think it looks worse to have the underline stroke go through the descenders of *y*, *p*, and *q*, and the ogoneks (̇) of *ę*, *ø*, and *o*, than it does to have the underline stroke be at different heights. To do this properly, it would really be necessary to design fonts with the underline stroke included in the individual letters.<sup>12</sup>

Sometimes manuscript transcriptions will require the use of special characters which are not available in existing fonts. It may be possible to create them by manipulating existing sorts.<sup>13</sup> For example, in Holm perg 11 4<sup>o</sup>, many words have letters with smaller letters placed over them. A logotype<sup>14</sup>  $\overset{b}{a}$ , coded as `{\bOVERa}`, can be defined using the existing letters “a” and “b”, boxes, and glue. For other sorts, like  $\mathfrak{z}$ , the Tironian symbol for Latin “et” (“and” in English and “ok” in Old Icelandic), it may be necessary to program a font using METAFONT.

 `\bOVERa` is actually defined as `\putontop{a}{b}{-}{-}`. The macro `\putontop` is defined like this:

```

1. \def\putontop{\begingroup
2. \catcode\-=12
3. \def\subputontop##1##2##3##4##5{%
4. \setbox1=\hbox{##1}%
5. \setbox3=\hbox{##3}%

```

<sup>12</sup> Cf. *The T<sub>E</sub>Xbook*, p. 323.

<sup>13</sup> “Sort” is a technical term for a typographical unit, synonymous with “character”. The term “character” is ambiguous in the context of T<sub>E</sub>X, because the characters in the input file differ in nature from the characters in the fonts used for typesetting, and the coding of the latter in terms of the former is subject to modification. Therefore, I sometimes prefer the unambiguous term “sort” for a character when I mean a typographic unit belonging to a font. WILLIAMSON defines a character or sort as a “single figure, letter, punctuation mark, symbol or word-space cast as a type or generated by photocomposition, CRT [cathode-ray tube] or digital system, or typed.” (368).

<sup>14</sup> WILLIAMSON defines “logotypes” (or “logos”) as follows: “letters joined to each other & cast on a single shank.” (p. 379.) He also notes that there is a shortage of logotypes in some photocomposition systems, where they are replaced by separate characters.

```

6. \setbox4=\hbox{##4}%
7. \setbox5=\hbox{##5}%
8. %
9. %% These are the default dimensions.
10. %% For shifting the top letter
11. %% to the right or left:
12. \dimen3=-.8\wd1
13. %% For shifting the top letter
14. %% up or down:
15. \dimen4=1.2\ht1
16. %% For increasing or decreasing
17. %% the kern following \putontop:
18. \dimen5=0pt
19. %
20. \ifdim\wd3>0pt
21. \advance\dimen3 by ##3\fi
22. %
23. \ifdim\wd4>0pt
24. \advance\dimen4 by ##4\fi
25. %
26. \ifdim\wd5>0pt
27. \advance\dimen5 by ##5\fi
28. %
29. \setbox2=\hbox{\kern\dimen3
30. \raise\dimen4
31. \hbox{\small ##2}}}%
32. \leavevmode\box1
33. \hbox to 0pt{\hss\box2\hss}%
34. \kern\dimen5\endgroup}\subputontop}

```

The macro `\putontop` takes its second argument and puts it above its first argument in a smaller size. The third and fourth arguments can be empty, or they can be used to shift the position of the second argument. The fifth argument can also be empty, or it can be used to increase or decrease the space following `\putontop`. The first and second arguments to `\putontop` need not be single characters. Here are some examples of using `\putontop`.

```

\putontop{abc}{def}{-}{-} ghi
⇒
def
abc ghi

```

This shifts the raised letters to the right.

```

\putontop{abc}{def}{10pt}{-} ghi
⇒
def
abc ghi

```

This shifts them down.

```

\putontop{abc}{def}{-}{-20pt}{-} ghi
⇒
abc ghi
def

```

This increases the kern following “abc”.

```
\putontop{abc}{def}{-}{-}{20pt} ghi
```

⇒

```
  def
abc    ghi
```

And this reduces it.

```
\putontop{abc}{def}{-}{-}{-10pt} ghi
```

⇒

```
  def
abghi
```

At the beginning of `\putontop`, the `\catcode` of `-` is reset temporarily to 12 (other). This makes it possible to use negative dimensions in its third, fourth, and fifth arguments.

◊ If a certain combination of letters like  $\overset{b}{a}$  appears frequently in a transcription, it’s easier to define a macro like `\bOVERa` for it rather than using `\putontop` explicitly over and over. The function `replace-items` can replace `{\bOVERa}` with “ab” or any other string, so that a word like “ $\overset{b}{r}$ abit” will appear in the concordance as “rabbit”. Another possibility is to account for `{\bOVERa}` in `letter-function`. In this case, “ $\overset{b}{r}$ abit” will be written to the concordance, unless “`r{\bOVERa}bit`” has been specified as a variant of “rabbit” in the lemmatization dictionary.

```
(generate-entry "rabbit"
 :variants "r{\bOVERa}bit")
```

◊ If a word like “eptir” (Engl. “after”) is abbreviated as `\putontop{e}{p}{-}{-}{2pt}{-}tir` → “ $\overset{p}{e}$ tir” in the input file, the first two arguments shouldn’t be discarded, because they belong to the word. However, the string “`\putontop`” and its other arguments, whether they’re empty or not, will cause an error in `letter-function`, if they reach it. The function `process-macro` inside the function `discard-items` discards the string “`\putontop`”, takes the first two arguments out of their braces and discards the rest.<sup>15</sup>

◊ This works, if words using `\putontop` always conform to this pattern. When they do, `\putontop` can be used explicitly in the input file. However, when they don’t, a different approach is required. Sometimes, for instance, letters that are put over other letters stand for parts of the word which are left out. If the word “drotning” is abbreviated frequently in the manuscript as “ $\overset{r}{d}$ ”,

<sup>15</sup> The function `process-macro` receives two arguments which tell it which arguments to take out of their braces and which to discard.

then it would make sense to define a macro `\dr` as follows:

```
\def\dr{\putontop{d}{r}{-}{-}{-}}
```

Then, `{\dr}` can be made a variant of “drotning” in the lemmatization dictionary.

```
(generate-entry "drotning" :variants "{\dr}")
```

or

```
(add-variants "drotning" "{\dr}")
```

Alternatively, the function `replace-items` can replace all occurrences of “`{\dr}`” in the input file with “drotning” before `current-line` is passed to `process-line`. Note that “`{\dr}`” should always appear within braces, like the special character macros.<sup>16</sup>

If there are a lot of cases of this type, and the transcription does not expand the abbreviations, then it would be a good idea to add another argument to `\putontop`.

```
\putontop{ma{\dh}r}{m}{r}{-}{-}
```

In this example, the word  $\overset{m}{ma}r$  (Engl. “man”) is abbreviated as  $\overset{m}{m}$ . Then, `discard-items` can call `process-macro` in such a way that the first argument, containing the whole word as it should appear in the concordance, is taken out of its braces so that it reaches `process-line` and `read-word`, and the other arguments are discarded. In this case, `\putontop` should discard the first argument when `TEX` is run, so that only the abbreviation is printed to the output.

◊ If the word “ $\overset{m}{ma}r$ ” appears frequently in the manuscript, which is likely, it might make sense to define a macro as follows:

```
\def\madhr{\putontop{}{m}{r}{-}{-}}
```

Here, the first argument can be empty, since `TEX` ignores it. The string “`{\madhr}`” must be made a variant of “ $\overset{m}{ma}r$ ” in the lemmatization dictionary or replaced by “`ma{\dh}r`” in the function `replace-items`, as above. The macro `\madhr` can be used in words like “`{\madhr}inn`” → “ $\overset{m}{m}inn$ ” for “ $\overset{m}{ma}drinn$ ” (nominative singular with enclitic article). Cases where a single special character is used to represent a frequently used word, like `{\hxbarhk}` → “ $\overset{h}{h}$ ” (abbreviation for “hans”, Engl. “his”, personal pronoun, masculine singular genitive) and `{\thxbarhk}` → “ $\overset{h}{t}$ ” for the word

<sup>16</sup> The control word `\dr` could even be accounted for in `letter-function`, which would make “ $\overset{r}{d}$ ” appear in the concordance. If it was assigned the list (d-value r-value o-value t-value n-value i-value n-value g-value), it would even be alphabetized correctly. I don’t think this would be useful for a concordance, but it might be for some other application.

“þessu” (demonstrative pronoun, feminine singular dative), can be handled in the same way.

An advantage in using  $\TeX$  is the ability to use temporary definitions for special characters. If I hadn’t programmed  $\mathfrak{z}$  yet, I could define it as follows:

```
\def\tirok{\&\$^{\rm Tir.}\$}
```

Then, a line like the following:

```
{\tirok} haleitari.~er komin at
k{\ydot}n
```


will be printed like this:

```
&Tir. haleitari. er komin at k˙n
```

When I’ve gotten around to programming  $\mathfrak{z}$ , it will be printed like this:

```
a haleitari. er komin at k˙n
```

but I won’t have to change my input file.<sup>17</sup>

 This is the  $\TeX$  code that’s necessary for using special characters like  $\mathfrak{z}$ , assuming the font is called `specialfont`.

```
\font\specialfontnine=specialfont9
\font\specialfonttten=specialfont10
\font\specialfontttwelve=specialfont12
...
```


Now, `\specialfont` must be defined where `\rm`, `\large`, `\small` etc. are defined, so that `\specialfont` accesses the correct size of the font `specialfont`, e.g.

```
\def\rm{\let\specialfont=%
\specialfonttten ...}
\def\small{\let\specialfont=%
\specialfontnine ...}
\def\large{\let\specialfont=%
\specialfontttwelve ...}
```

Finally, `\tirok` is defined like this:

```
\def\tirok{{\specialfont\char'140}}
```

The character “ $\mathfrak{z}$ ” is in position octal 140 (decimal 96, hexadecimal 60) in `specialfont`.

 In a similar way,  $\TeX$  makes it possible to distinguish words in the input file which should not be distinguished in the output. For instance, there are two forms of “m” in Holm perg 11 4° which are used interchangeably. Depending upon the degree of normalization desired in the transcription, these two letters can be distinguished in the output or not, as desired. One form can be coded as `m` →

“m” and the other as `{\mone}` (for “m-one”) which also prints as “m” and is defined as

```
\def\mone{m}
```

The function `replace-items` can convert each occurrence of the string “`{\mone}`” in the input file to “m” before current-line is passed on to process-line, so `{\mone}` will never appear in the concordance. If, later, a different letter form should be used to represent `{\mone}` in the output, `\mone` can be redefined, e.g.,

```
\def\mone{{\specialfont\char'001}}
```

**Math mode material** is always treated in the normal way by  $\TeX$ . It is treated in different ways by `conctex.lsp`. Display math material is processed normally by  $\TeX$  and discarded by `conctex.lsp`.<sup>18</sup> Math mode is very likely to appear in transcription lines, mainly for sub- and superscripts and certain special characters. Sometimes, the math mode material will be irrelevant for the concordance, but at other times it may be an essential part of a word, or even serve to distinguish between two otherwise identical words. If math mode material is attached to a word, that is, if it isn’t separated from the word by blank space or another word-separator (like most punctuation marks), it is considered to be part of the word. If not, `conctex.lsp` discards it. For example,

```
drotning\pi
```

is treated as a word, “drotning <sup>$\pi$</sup> ”. It will appear as such in the concordance and will be a separate entry from “drotning”, if this word occurs in the transcription. If this appears in the transcription,

```
drotning\pi
```

`conctex.lsp` will discard the math mode material, “drotning <sub>$\pi$</sub> ” will appear in the output and “drotning” will appear in the concordance. To have `conctex.lsp` discard the math mode material, but not separate it from the word in the output, type:

```
drotning\pi
```

In this case, `conctex.lsp` also discards the math mode material, and “drotning” appears in the concordance, but “drotning <sup>$\pi$</sup> ” appears in the output, because  $\mathfrak{z}$  is ignored by  $\TeX$  (`\catcode'\mathfrak{z}=9`), and treated as a word separator by `conctex.lsp`. The math mode material can also be put on the next input line following a %.

```
drotning%
\pi himins ...
```

<sup>17</sup>  $\TeX$  is generous with space for storing macros; there’s room for over 2000, so there should be enough for all the special characters you need.

<sup>18</sup> The program `conctex.lsp` converts  $\mathfrak{z}$  to \*, so display math mode material is treated as a commentary.

⇒

drotning<sup>π</sup> himins ...

and “drotning” appears in the concordance.

The `\catcodes` of some characters, such as `-`, `_` and `<`, that have been changed for use in transcription lines, should have their normal values in math mode. The token list `\everymath` resets them.

```
\everymath={\catcode'\-=12\catcode'\_ =8
\catcode'\*=12\catcode'\<=12\relax}
```

The `\relax` at the end of `\everymath` is necessary because assignments, such as changes to `\catcodes`, only take effect when T<sub>E</sub>X is building a list.<sup>19</sup> The same effect could be achieved with `\vbox{}` or `\hbox{}`, or a harmless macro (but not `\empty`).

⚡ Math mode material should not extend over more than one transcription line. If it does, `conctex.lsp` will signal an error. Normally, it shouldn't be necessary anyway, since most manuscripts don't contain multiline equations. Even if yours does, the elements of the equation probably don't belong in the concordance anyway, so the multiline math mode material can be put in a commentary. In this case, it may be necessary to reset the line number explicitly with `set-position`. If your application requires a lot of math mode, it would be advisable to reprogram the routines for handling it. Display math mode material is treated as a commentary, so it can extend over any number of lines.

**Line ends.** The treatment of line ends is an important factor in ConcT<sub>E</sub>X. A typeset manuscript transcription will generally have lines corresponding to the lines in the manuscript, so it is undesirable to have T<sub>E</sub>X do the line breaking for the text lines. (It would also be a lot of work to write a hyphenation dictionary for a medieval manuscript.) However, it will rarely if ever be desirable to use `\obeylines`, because there are many other items which can appear in the input lines, making it impractical for the lines in the input file to correspond to the lines in the output: font changes, index entries, footnotes, comments, etc. Therefore, the ends of transcription lines must be indicated.

Most transcription lines should end in `\`. The control symbol `\` is `\let` to `\par` in `conctex.tex` and recognized as a line end by `conctex.lsp`. The program `conctex.lsp` keeps track of the line numbers, so this is important. A transcription line

<sup>19</sup> *The T<sub>E</sub>Xbook*, p. 373.

in the input file that is followed by a blank line is treated as if it ended in `\`.

Normally, the lines in the output should correspond to the lines in the manuscript, so they should be wide enough, i.e., the value of `\hsize` should be large enough so that transcription lines don't have to be broken. In exceptional cases, however, this may not be possible, for example, if a line includes a long commentary.

```
\lineno{8} himins _{ok} *{\<{\ss This
is an extremely long commentary which
causes this otherwise short line to
be broken so that it results in more
than one line in the output.}\>*
iar{\dh}ar.~s{\ae}l {ok} dyr{\dh}\-
\lineno{9} lig m{\o}r Maria.~mo{\dh}ir
d_{fro}tti_{n}s\
```

⇒

8: himins ok (This is an extremely long commentary which causes this otherwise short line to be broken so that it results in more than one line in the output.) iarðar. sæl ok dyrð  
9: lig mør Maria. moðir drottins

⚡ The control symbol `\` expands to `\par` and the value of `\parskip` and `\parindent` have been set to `0pt` in order that multiple blank lines won't cause excessive vertical space or unwanted indentation in the output. If `\` expanded to `\hfil\break`, a blank line following `\` would cause two `\baselineskips` to appear in the output, one from the `\break` and one from the second `^M` (`\return`), which is converted to `\par`. On the other hand, a `\par` following a `\par` is harmless.<sup>20</sup> The text in manuscripts is generally not divided into paragraphs, so it's unnecessary to use blank lines to indicate paragraphs, and it's convenient to allow them for the sake of making the input file more readable.

⚡ The control symbol `\` is not needed in the `\plain` environment, because T<sub>E</sub>X does the line breaking there. The values of `\parskip` and `\parindent` can also be changed in the definition of `\plain`, for text that should be formatted differently from the transcription itself.

**Manuscript positions** are defined according to leaf (folium), side (recto or verso), column (if there are multiple columns), and line number. The

<sup>20</sup> The first `\par` puts T<sub>E</sub>X into vertical mode, where the second `\par` has no effect, “except that the page builder is exercised ... and the paragraph shape parameters are cleared.” (*The T<sub>E</sub>Xbook*, p. 283.)

leaves of Holm perg 11 4° and AM 234 fol have two columns, so a position is identified as follows: 2ra 17 is leaf 2, recto, column a, line 17. When a new leaf, side or column begins, the input file must contain a line that looks like this:

```
@ (set-position "28vb")
```

If that column starts with line 15, perhaps because the top of the leaf is missing or unreadable, the line should look like this:

```
@ (set-position "28vb" 15)
```

Lines beginning with @ are ignored by T<sub>E</sub>X and evaluated by `conctex.lsp`. The Lisp program counts the lines and keeps track of the position in the manuscript. However, it will usually be useful to indicate line numbers in the transcription itself, so a line in the input file might look like this:

```
31: f_{ra} savgn_{n} Mathevs
```

In the output, it will look like this:

```
31: fra savgn Mathevs
```

However, “31” should not appear as a lemma in the concordance. Therefore, `conctex.lsp` discards numbers, punctuation and blanks at the beginning of transcription lines. Another possibility is to use the macro `\lineno` to make them print or not, according to the value of a conditional.

```
\ifprintlinenumbers
\def\lineno#1{...}\else
\def\lineno#1{\relax}\fi
\lineno{1} Drotning etc.
```

```
\newif\ifprintlinenumbers
\printlinenumberstrue
```

⇒

```
1: Drotning etc.
```

```
\printlinenumberfalse
```

⇒

```
Drotning etc.
```

The macro `\lineno` is an exception to the rule about `conctex.lsp` discarding lines that begin with undelimited macros. It can be defined as follows:

```
\ifprintlinenumbers
\def\lineno#1{\leavevmode
\setbox0=\hbox{33:\space}%
\hbox to \wd0{\hss#1:\space}%
\hangindent\wd0\hangafter 1}
\else
\def\lineno#1{\relax}\fi
```

Line numbers, whether explicit or in `\lineno`, are discarded by `conctex.lsp`. Since `\`, `\-` and `-`

(explained below) and a blank line all cause a `\par` to be added to the current list, `\lineno` always begins a paragraph (assuming `\printlinenumbers-true`). Usually, this paragraph will consist of one output line, just as it corresponds to one line in the manuscript, but if the paragraph is longer than one line, the `\hangafter` and `\hangindent` macros cause the following lines to be indented so that they begin directly below the text of the first line in the paragraph.

◆ The same effect could be achieved by using the token list `\everypar`, and separate `\everypars` could be maintained for the `\plain` and `\trans` environments.

◆ The “magic number” 33 in `\box0` in the definition of `\lineno` is there simply to make the box an appropriate size. Numbers wider than “33” extend into the left-hand margin, so the colons (or periods, or whatever) always line up.

In some applications, positions may not require line numbers. For instance, positions in a concordance of the Bible should be given as book, chapter and verse. In this case, T<sub>E</sub>X could take care of the line breaking, and `\` could be used to indicate the end of a verse. It could be `\let` to `\relax` in `conctex.tex` and cause `conctex.lsp` to increment a “verse-counter” instead of line-counter (explained below). The exact way that ConcT<sub>E</sub>X handles line numbers in the input file can and should be set for each particular project.

◆ While it is convenient to define macros in this way:

```
\ifprintlinenumbers
\def\lineno#1{...}\else
\def\lineno#1{\relax}\fi
```

it is sometimes preferable to define them like this:

```
\def\lineno#1{\ifprintlinenumbers
... \else \relax \fi}
```

or like this:


```
\def\linenoprint#1{...}
\def\linenonoprint#1{\relax}
```

```
\ifprintlinenumbers
\let\lineno=\linenoprint
\else\let\lineno=\linenonoprint
\fi
```

The advantage of the last two alternatives is, that it is possible to put the definitions of `\linenoprint` and `\linenonoprint` in a file



used for generating a preloadable format file with INITEX. If you try to generate a format file using the first example, where the definitions themselves are within the conditional construction, one version will be skipped when INITEX runs, depending on the current value of `\ifprintlinenumbers`. However, in the second example, the conditional construction is within the definition, so it will expand according to the value of `\ifprintlinenumbers` when `\lineno` is called. This means that the expansion of `\lineno` can switch back and forth during the T<sub>E</sub>X run. In the third example, the two definitions are assigned to two different macros, and `\lineno` is `\let` to one of them according to the value of `\ifprintlinenumbers` at the time. This conditional construction can be put in an ordinary macro file, loaded after the preloaded format. The assignment must occur before `\lineno` is called for the first time. A subsequent change to the value of `\ifprintlinenumbers` later in the T<sub>E</sub>X run has no effect on the expansion of `\lineno`. You can, however, switch the definition explicitly by saying `\let\lineno=\linenoprint` or `\let\lineno=\linenonoprint` at any time. If you have a lot of macros, it can be useful to create a preloadable format in this way.<sup>21</sup>

 The Lisp program doesn't use the explicit line numbers in the input file for its line counting routine, although it could. Explicit line numbers are optional, so `conctex.lsp` needs its own line counting routine anyway. It does, however, check that the line numbers from its routine and the explicit line numbers in the input file match up. If they don't, it issues a warning, but doesn't signal an error. Warnings are written to standard output when `conctex.lsp` is run, but also to the file `warnings`. If `conctex.lsp` is set up to write code like this to `warnings`:

```
echo "At line 21 of input.tex:
Line numbering is incorrect.
(\\lineno: 12) (line-counter = 14)"
```

and `conctex.lsp` is run using a UNIX shell script, then the shell script can execute `warnings` by calling `sh warnings`, causing all of the warnings to be printed to standard output after `conctex.lsp` is done.

**Broken words.** Some lines may end in words that are continued on the next line. In AM 234 fol, some of the broken words have hyphens and some do not. It is necessary to distinguish between these two cases in the input file. I use - (hyphen) to indicate

a broken word with an explicit hyphen and `\-` to indicate a broken word with no hyphen. Using `\-` indicates broken words in the input file, and causes `conctex.lsp` to treat such a broken word as a unit, but no hyphen appears in the typeset output. For example:

```
tok j vpphafi {\slong}n{\slong}
Gv{\dh}{\slong}pia-
llz.~at telia {\ae}tt drottin{\slong}
ie{\slong}us
```

⇒

```
tok j vpphafi fnf Gvðfpia-
llz. at telia ætt drottinfnf iefus
```

Whereas

```
tok j vpphafi {\slong}n{\slong}
Gv{\dh}{\slong}pia\
llz.~at telia {\ae}tt drottin{\slong}
ie{\slong}us
```

⇒

```
tok j vpphafi fnf Gvðfpia
llz. at telia ætt drottinfnf iefus
```

If a line in the input file ends in - or `\-`, this obviously indicates the end of a line in the manuscript, so it would be redundant to type `-\` or `\-\`.<sup>22</sup>

The program `conctex.lsp` recognizes - and `\-` at the end of a line as line ends for its line numbering routine. T<sub>E</sub>X also recognizes them as line ends at the end of a line and inserts a `\`. However, - can also occur within a line, or even at the beginning of a line. Here it should not be recognized as a line end, either by T<sub>E</sub>X or by Lisp. A - at the beginning of or within a line is simply printed as - (what else?). As far as T<sub>E</sub>X is concerned, a `\-` within a line is harmless, but doesn't make any sense, since it doesn't print anything. Therefore, T<sub>E</sub>X issues a warning but doesn't signal an error. The program `conctex.lsp` discards `\-`s that are neither at the end of an input line nor followed by `\`. The strings -- and --- still yield - (en-dash) and — (em-dash) respectively, but they do not cause line ends, either in T<sub>E</sub>X or in Lisp.<sup>23</sup>

The following code is necessary to accomplish this:

```
1. \catcode'\@=\active
2. \let@=\-
3.
```

<sup>22</sup> Although it's redundant, it is nonetheless permitted.

<sup>23</sup> If a concordance is to be generated for a manuscript which uses more characters to indicate broken words, like = or =, both `conctex.tex` and `conctex.lsp` must be modified to account for these cases.

<sup>21</sup> *The T<sub>E</sub>Xbook*, p. 344.

```

4. \def\hyphen{-}
5. \def\dash{--}
6. \def\Dash{---}
7.
8. \catcode\-=\active
9.
10. \begingroup
11. \catcode\^M=\active
12. \global\let^M=\empty
13.
14. \gdef\invisiblehyphen{
15. \begingroup\catcode\^M=\active
16. \def\subhyphen{\ifx\next^M\\else
17. \ifx\next\ % Do nothing
18. \else
19. \message{Ignoring \noexpand\-. %
20. Don't use \noexpand\-. %
21. in the middle of a line.}
22. \fi\fi\endgroup}
23. \futurelet\next\subhyphen}
24.
25. \global\let\-=\invisiblehyphen
26.
27. \gdef-{\begingroup\catcode\^M=\active
28. \def\aeathypens##1{
29. \futurelet\next%
30. \bEathypens}
31. \def\beathypens{
32. \ifx\next-\Dash
33. \expandafter\cEathypens\else
34. \dash\endgroup\fi}
35. \def\cEathypens##1{
36. \futurelet\next\endgroup}
37. \def\subhyphen{
38. \ifx\next^M\hyphen\\% It's a return.
39. \let\eathypens=\endgroup\else
40. \ifx\next-% It's a hyphen
41. \let\eathypens=%
42. \aeathypens\else
43. %% It's not a return or a hyphen.
44. \hyphen
45. \let\eathypens=\endgroup
46. \fi\fi
47. \eathypens}
48. \futurelet\next\subhyphen}
49.
50. \endgroup

```

The primitive `\-`, which is ordinarily used for inserting discretionary hyphens, must be redefined. Since the user decides where transcription lines are to be broken, discretionary hyphens are unnecessary there. In case they are needed, however, the character © (decimal 169, octal 251, hexadecimal

A9) is made active and `\let` to `\-` before `\-` is redefined (lines 1 and 2). Make sure you use © and not `^a9`. T<sub>E</sub>X will treat `^a9` as a single token, but `conctex.lsp` will not, just as with `¥` and `^a5` (unless `conctex.lsp` is modified to allow this). In commentaries and the `\plain` environment, `\-` reverts to its primitive self.

T<sub>E</sub>X's ligature routine won't work for `-`, `--`, and `---` after the `\catcode` of `-` has been changed to `\active` (line 8), so first I put `-`, `--` and `---` into the expansion of the new control words `\hyphen`, `\dash` and `\Dash`, so I can still use them. Now I define `-` and `\invisiblehyphen` and `\let\-` to `\invisiblehyphen`.<sup>24</sup> They check whether the token following `-` or `\-` is a `<return>`, ASCII code 13, which can be notated as `^M`, in which case `\` should be inserted to break the line.<sup>25</sup>

Under normal circumstances, i.e., when `\catcode\^M=5`, it wouldn't be possible to tell whether the character following an active char is a `<return>` or not. For example:

```

\def\abc#1{\show#1}
\abc
d

```

causes

```

> the letter d.
<argument> d
\abc #1->\show #1

```

to be printed to the screen, or standard output, to be precise, and

```

\abc

```

d

causes an error.

```

?
Runaway argument?
! Paragraph ended before
\abc was complete.

```

<sup>24</sup> The macro `\invisiblehyphen` is defined in order to make it possible to switch the definition of `\-` back and forth when changing environments. This is necessary, because `\-` is already a control symbol (in fact, it's a primitive), whereas `-` has `\catcode 12` in the `\plain` environment, commentaries, and math mode. Therefore, the active character `-` can be defined globally. When the environment is switched, it suffices to change the `\catcode` of `-`.

<sup>25</sup> Incidentally, you might be wondering why I'm talking about "`<return>`s". I use a computer running UNIX where the end-of-line character is `<newline>`, ASCII 10, and not `<return>`, ASCII 13. The reason is that T<sub>E</sub>X converts the external coding scheme of the input file to its own internal coding (*The T<sub>E</sub>Xbook*, p. 43), so UNIX' `<newline>`s are converted to T<sub>E</sub>X's `<return>`s.

```
<to be read again>
      \par
```

The first time `\abc` was invoked, it skipped the first `^M`, which had been converted to a space, and found the “d” in the next line. The second time it was invoked, `\abc` skipped the first `^M`, but found the second one, which had been converted to a `\par` token (*The T<sub>E</sub>Xbook*, p. 47). A normal macro can’t accept arguments that contain `\pars`, so an error was signalled. Here’s another version.

```
\long\def\abc#1{\show#1}
\abc
```

```
d
```

```
⇒
```

```
> \par=\par.
<argument> \par
```

```
\abc #1->\show #1
```

A macro defined using `\long\def` won’t signal an error if a `\par` occurs in its arguments, but the first `^M` is still skipped. So simply reading in arguments is useless for discovering whether a macro or an active character is followed by a `^M`. In addition, `\abc\par` will produce the same screen output as the last example, since T<sub>E</sub>X can’t distinguish between an explicit `\par` and one that was converted from a `^M`.

The same behavior can be demonstrated using `\futurelet`:

```
\def\abc{\futurelet\next\subabc}
\def\subabc{\show\next}
\abc
a
```

```
⇒
```

```
> \next=the letter a.
```

```
and
```

```
\abc
```

```
a
```

```
⇒
```

```
> \next=\par.
```

In order to check whether the character following a control sequence or an active character is a `^M`, it is necessary to change the `\catcode` of `^M`. One possibility is to change it to `\active`. I do this locally within the definitions of `-` and `\-`, so that `^M` otherwise behaves normally. However, in order that the `^M`s within the actual definitions of `-` and `\-` are handled correctly while they’re being

read into T<sub>E</sub>X’s memory, it is necessary that `\catcode\^M=\active` and that `^M` is set globally to `\empty` (line 11–12). It could also have been set to `\relax`.

Serendipitously, setting `^M` to `\empty` makes it unnecessary to use `%` at the end of lines that end in `{` or `}`, such as lines 14, 21–22, etc. However, `%` is still necessary in the `\message` command in lines 19–20, otherwise there would be no space between “\.” and “Don’t” in the message, i.e., any trailing spaces before the `^M` would be swallowed up. A `%` is also necessary in the `\futurelet` construction in line 29, otherwise `\next` would be set to `\bEathyphens`, the token following the `^M`, which is expanded.

The way T<sub>E</sub>X handles `^M` at ordinary times, i.e., when `\catcode\^M=5`, causes the global definition of `^M` as a macro to be reset to `\par`. Therefore it’s necessary to set it globally in line 12, so that it expands to `\empty` in `-` and `\-`. Setting `\catcode\^M=\active` in line 11 is only in effect inside the group that ends on line 50; when `-` or `\-` is invoked, *it* must begin a group and reset `\catcode\^M` to `\active`. However, this being done, the `\global\let^M=\empty` is in effect while `-` or `\-` is expanding. It doesn’t work to get rid of the `\global\let\^M=\empty` and put `\let\^M=\empty` in the definitions of `-` and `\-`. I’m afraid I don’t understand why not, but it is apparently connected with peculiarities of category code 5.

Since the definitions of `-` and `\invisiblehyphen` are inside of a group, they must be `\gdefs` (global definitions), or they wouldn’t be available outside the group. The same applies to `\-`, which is `\let` globally to `\invisiblehyphen`. The definitions inside `-` and `\invisiblehyphen`, however, are local to the groups begun in these macros.

◆ Another possibility is to set `\catcode\^M=\active` and `\let^M=\empty` globally (i.e., get rid of the group begun in line 10 and ended in line 50, change `\gdef` and `\global\let` to ordinary `\def` and `\let`, and reset `\catcode\^M=5` after the definitions.

If `-` were defined like this:

```
\gdef-#1#2{...}
```

i.e., with arguments, it wouldn’t be possible to check whether the tokens following `-` in the input file were `^M` or not, because the arguments would have been read in and expanded before `-` had had a chance to change `^M`’s `\catcode`. Once a token has been read in as an argument this is no longer possible. So `-` and `\invisiblehyphen` change the `\catcode` of

`^^M`, peek at the following token using `\futurelet`, and turn over control to `\subhyphen`. Both `-` and `\invisiblehyphen` have their own version of `\subhyphen`.

In `\invisiblehyphen`, `\next` contains the token following the `\-` in the input file. This token is examined by `\subhyphen`. If `\next` is `^^M`, `\subhyphen` inserts `\` to break the line. If it's `\`, `\subhyphen` does nothing (`\-` is redundant but permitted in the input files). If it's anything else, `\subhyphen` issues a warning that `\-` shouldn't be used in the middle of a line. Then `\subhyphen` ends the group begun by `\invisiblehyphen` in line 15.

The active character `-` functions in a similar, but more complicated way. It too uses `\futurelet` and `\next` to examine the following token.

⊥ If `\next` is `^^M`, `\subhyphen` inserts a hyphen with `\hyphen` (defined in line 4), a `\` to break the line, and `\lets` the control word `\eathypens` to `\endgroup`, to end the group begun in line 27.

⊥ Else, if `\next` is not `-`, `\subhyphen` adds a `\hyphen` to the current list, but no `\`, and `\eathypens` is `\let` to `\endgroup`.

⊥ Else, if `\next` is `-`, `\subhyphen` `\lets` `\eathypens` to `\aEathypens`.

Now, `\eathypens` takes over control. In the first two cases, i.e., `\next` was `^^M` or anything else other than `-`, `\eathypens` merely ends the group begun in line 27. However, if `\next` was `-`, matters are more complicated.

If `-` is followed by `-`, this could be `--`, which should be replaced with an en-dash, or `---`, which should be replaced with an em-dash. In neither of these cases should the line be broken. Therefore, it's necessary to peek at the token following the second `-`, but first we have to dispose of the second `-`, because it's not possible to peek at the next token but one, and the second `-` is still to be read by `TeX`'s parser. The macro `\aEathypens` takes one argument, which disposes of the second `-`, then peeks at the following token using `\futurelet` and `\next`, and passes control to `\bEathypens`. This macro examines `\next`. If it's not `-` (in the `\else` construction), `\bEathypens` inserts a `\dash` into the current list, and ends the group begun in line 27. However, if `\next` is `-`, it inserts a `\Dash` into the current list and calls `\cEathypens`. The `\expandafter` is necessary to prevent `\cEathypens` from reading in `\else` as its argument. Instead, it reads in and thereby disposes of the third `-`. It could just end the group now, but it doesn't. It peeks at the token following the third `-` using `\futurelet` and `\next` yet again. This has

the effect that `--` or `---` at the end of a line is not followed by a blank space.

Normally, in plain `TeX`, or when using the `\plain` environment in `ConcTeX`, `--` or `---` at the end of a line is followed by a blank space.

```
... in lines 299--
547 ...
```

```
There was a sudden crash---
then silence.
```

⇒

```
... in lines 299_547 ...
```

```
There was a sudden crash_ then silence.
```

However, in the `\trans` environment, the spaces disappear.

```
... in lines 299-547 ...
```

```
There was a sudden crash—then silence.
```

This is nice, since one doesn't want spaces following en- or em-dashes. If you do want them, it's easy enough to redefine `\bEathypens` and `\cEathypens` so they're added before `^^M`, or you can type

```
... in lines 299--\space 547 ...
```

```
There was a sudden crash---\space
then silence.
```

However, you may be puzzled as to why they disappear. If `\cEathypens` just eats the third `-` and ends the group, a following `^^M` will cause a space to be inserted into the current list, so it's the `\futurelet` that causes the space to disappear.<sup>26</sup> When `\futurelet` peeks at a token, it fixes its `\catcode` even though it doesn't expand it.<sup>27</sup> When the group ends and `-` is finished, `TeX`'s parser reads the `^^M` that follows the second or third `-`. The `\catcode` of this `^^M` was fixed at 13, so it expands to `\empty`, its expansion within `-` rather than a blank space.

Changing the category code of `-` to `\active` has some minor, unpleasant side effects in the `\trans` environment. It is no longer possible to use `-`, `--` and `---` in the arguments to some macros, such as `\beginchapter` and `\beginsection`, which are defined in `conctex.tex`, and code like `\vskip-2pt` will cause an error, because `-` must have `\catcode 12` (other) for this to work. The control words `\hyphen`, `\dash` and `\Dash` can be used in macro

<sup>26</sup> In the case of `--`, it's the `\futurelet` in `\aEathypens` that causes the space to disappear.

<sup>27</sup> *The TeXbook*, p. 381: "... `TeX` stamps the category on each character when that character is first read from a file."

arguments instead; other cases should be put into a commentary or the `\plain` environment, where `-` has `\catcode=12`, e.g., `*\vskip-12pt*`.

◊ If certain macros that use `-`, `--`, or `---` are used frequently in the `\trans` environment, and you don't want to put them in commentaries or the `\plain` environment, you can redefine them using a similar technique to that used in the definitions of `-` and `\invisiblehyphen`.

```
\begingroup
\catcode'\-=12
\gdef\mymacro{\begingroup
\catcode'\-=12
\def\submymacro##1##2##3{%
... \endgroup}}
\endgroup
```

The macro `\putontop` is defined like this. It doesn't work if you try to redefine `\vskip`, though.

**Homograph identifiers.** *Homographs* are words that are spelled the same, but belong to different lemmata. They must be indicated explicitly in the input file. In order to do this, I have defined the characters “<” and “>” to delimit *homograph-identifiers* in the `\trans` environment. For instance, in Old Icelandic, the word “á” can be a noun, meaning “river” or a preposition, meaning “at, on,” etc. In the input file, I can distinguish between them by typing the preposition as `{\ 'a}<p>` and the noun “á” as plain `{\ 'a}` or `{\ 'a}<n>`. `TeX` handles them in different ways, depending on the value of `\ifdraft`. For rough drafts, it will be useful to see the homograph-identifiers, whereas they should not appear in the final draft.

To accomplish this, the `\catcode` of `<` has been changed to `\active`. The definition of `<` depends on the value of `\ifdraft`:

```
\catcode'\<=\active

\def\eatit#1{\ifx#1>\else
\expandafter\eatit\fi}

\def\donteatit${\${%
\def\subdonteatit##1{%
\ifx##1>${\}$\else##1%
\expandafter\subdonteatit\fi}%
\subdonteatit}

\ifeathomoids
\let<=\eatit
\else
\let<=\donteatit\fi
```

The macro `\eatit` simply reads an argument, tests to see if it's “>” and if it isn't, calls itself again. The arguments it reads are discarded, hence the name. The macro `\donteatit` functions in a similar way, but has the added complication that the “{” should only be put into the current list once, so the main work of not eating the string is taken over by `\subdonteatit`. When `\draftfalse`, `<` is `\let` to `\eatit`. When `\drafttrue`, the homograph-identifiers shouldn't be eaten, so `<` is `\let` to `\donteatit`, which puts the text between `<` and `>` inside curly braces, e.g., “á{p}”.

◊ Homograph identifiers can be as long as you want, because `\eatit` and `\donteatit` process the tokens (or groups) within the angle braces one-by-one. This prevents a long argument from burdening `TeX`'s memory. When `\eatit` or `\subdonteatit` calls itself recursively within the conditional construction, the `\expandafter` allows the conditional, and hence the current invocation of `\eatit` or `\subdonteatit`, to complete execution before the recursive call is expanded. Therefore, only one invocation of `\eatit` or `\subdonteatit` is active at any time. This is called “tail recursion” (cf. *The TeXbook*, p. 219). It would also be possible to define `\eatit` and `\donteatit` like this:

```
\def\eatit#1>{\relax}
\def\donteatit#1>{\${\${#1$}\$}
```

In this case, extremely long homograph identifiers will burden `TeX`'s memory, but in practice, it would be just as good, since it's unlikely that anyone would want to type in extremely long homograph identifiers.

### The Lisp program `conctex.lsp`

The Lisp program `conctex.lsp` consists of several parts:

1. A lemmatization dictionary.
2. A parser that reads the input files, discards extraneous material, and passes the transcription lines on for further processing.
3. A routine for extracting information from the transcription lines, accessing Lisp symbols, and storing the information in data structures.
4. An output routine that writes the `TeX` file containing the concordance.

ConcTeX does not use the page numbering information generated by `TeX`'s output routine, so it doesn't require a preliminary pass. The concordance shows the position of individual words in the *original manuscript*, not the page numbers of the printed transcription. This makes it possible

to generate the concordance directly from the  $\text{\TeX}$  input file without running  $\text{\TeX}$  at all.

⦿ Making a concordance using the page numbers generated by  $\text{\TeX}$ 's page breaking routine is a more difficult matter. If the input files contain explicit commands for page breaking, i.e.,  $\backslash\text{eject}$ ,  $\backslash\text{supereject}$ , and/or macros that call one or the other of these macros,  $\text{Conc}\text{\TeX}$  can be used with only trivial modifications. However, it will not work if  $\text{\TeX}$  is supposed to break pages automatically. One way around this problem would be to let  $\text{\TeX}$  break the pages automatically, and insert code for  $\text{conctex.lsp}$  in the final version, at the page breaks  $\text{\TeX}$  found, so  $\text{conctex.lsp}$  knows to increment a page counter. This would not be entirely satisfactory, however.

⦿ If explicit page breaks are used, or code is entered in the input file indicating where pages are broken, then explicit line breaks (using  $\backslash\text{break}$  and/or  $\backslash\text{par}$ , or macros using one or the other of these commands) would make it possible to generate a concordance using line numbering information referring to the lines in the output. If  $\text{\TeX}$  does the line and page breaking automatically, however, and page breaks are not explicitly indicated, then  $\text{conctex.lsp}$  won't know when to increment the page counter and reset the line counter to 1. *There is no way to generate line numbering information without explicitly breaking the lines.*

⦿ It is therefore not possible to use  $\text{Conc}\text{\TeX}$  for making a concordance using line and page numbers that result from  $\text{\TeX}$ 's line and page breaking routines, i.e., without explicit line and page breaking commands. But it might be possible to design a concordance program that could do this by having it process the  $\text{dvi}$  file instead of the input file. However, if this sort of concordance were desired, a better approach would be to write a new version of  $\text{\TeX}$  that incorporates the features of  $\text{Conc}\text{\TeX}$  in the  $\text{WEB}$  program itself, thus making it unnecessary to use an auxiliary program. However, since  $\text{conctex.lsp}$  uses several features peculiar to Lisp, an entirely different program structure would be necessary.

**Lemmatization.** The most difficult problem in generating a concordance is lemmatization. There are three cases that need to be accounted for:

1. *Homographs.* Words that are spelled the same, but belong to different lemmata, like the word "spring", which can be one of several nouns or a verb.

2. *Subsidiary forms.* Distinct words that belong to the same lemma, like "man", "man's" and "men" in English.
3. *Variants.* Words that are spelled differently, but are really the same, and should appear under the same heading, like "color" and "colour" in English.

A lemma in the context of generating a concordance is a set of one or more distinct words which belong together, like the various forms in a morphological paradigm, e.g., the nominative, genitive, dative, and accusative; singular and plural, of "drotning" (Engl. "queen").

	<i>Sg.</i>	<i>Pl.</i>
<i>Nom.</i>	drotning	drotningar
<i>Gen.</i>	drotningar	drotninga
<i>Dat.</i>	drotningo	drotningom
<i>Acc.</i>	drotning	drotningar

One of these words is denoted the *main form* of the lemma; with nouns, it's usually the nominative singular, with verbs, the infinitive, etc. The *subsidiary forms* should appear in the concordance below the main form, indented, with their occurrences, arranged according to grammatical criteria.


drotning (2) 1va 12, 3rb 14.  
 drotningo (2) 2rb 25, 12va 30–31.  
 drotninga (1) 13ra 16.

*Homographs* must be distinguished in the concordance. Sometimes, they can belong to the same lemma, like the three forms of "drotning" that are spelled "drotningar": the genitive singular and the nominative and accusative plural. Sometimes they can belong to different lemmata, like the noun "á" and the preposition "á" in Old Icelandic.

Lemmatization must also account for *variants*. Where there are variants, all of the occurrences are assigned to a single form, which appears in the concordance. One way variants can arise is from the use of interchangeable letter forms in a manuscript. The manuscript transcription might use a macro or a font change to indicate the use of an alternative letter. For example, the word "prestr" may sometimes appear in the transcription as "prestr" → "prestr" and sometimes as "prestr{\r}" → "prestr", depending on the form of "r" used in that passage in the manuscript, so "prestr" may be termed a variant of "prestr" (or vice versa). Only one of them, say "prestr", should appear in the concordance, and the occurrences of "prestr" should be listed under "prestr". Subsidiary forms can also have variants.

Lemmatization is the process of sorting individual words, so that each one is put into its proper

lemma. There are several possible solutions to this problem. The one described here is a *lemmatization dictionary*, which must account for all subsidiary forms and variants.

 Running `conctex.lsp` without loading a lemmatization dictionary produces a complete list of all the words in the transcription, with their occurrences, but each entry is considered a main form, and only those variants are accounted for which `replace-items` or `process-line` catch automatically. For a concordance this is clearly inadequate, but it might be useful for some other purpose.

If concordances are to be generated for normalized transcriptions, or for a group of manuscripts whose orthographic conventions are very similar, a master lemmatization dictionary can be used. In most cases, however, the orthography and the form of the language of a given manuscript will diverge enough from that of others that it will be necessary to create a special lemmatization dictionary for that particular manuscript.

One of the first things that `conctex.lsp` does is to load the lemmatization dictionary. This is one or more files of Lisp code with invocations of the functions `generate-entry`, `harmless`, and `add-variants`. These functions cause data to be stored in `word structures`.

The data for the entries in the concordance are stored in `structures` of type `word`, defined like this:

```
(defstruct word
  main-form
  occurrences-mariu-a
  occurrences-mariu-s
  sort-string
  tex-string
  forms
  root
  variants
)
```

More slots can be added for other applications; in particular, `occurrences` slots can be added for additional manuscripts.

**The function `generate-entry`.** The basic function for generating a lemmatization dictionary is `generate-entry`. It's invoked like this:

```
(generate-entry "David")
```

or like this:

```
(generate-entry "David" noun)
```

The string, which is the first, required argument to `generate-entry`, is used, perhaps with some modifications, as the name of a symbol which is bound to a `word-structure`. The original string is stored

in the `main-form` slot of the `word structure`. In this example, the string "David" is surrounded by `||`, and the symbol is accessed with `read-from-string`.<sup>28</sup>

```
(set (read-from-string "||David|")
     (make-word :main-form "David"))
```

Surrounding a string with `||` has the effect of escaping all the characters within the string. This makes it possible to have symbol names with lowercase letters and other characters, which are normally not allowed in symbol names in Lisp. Evaluating

```
(read-from-string "David")
```

without `||` returns a symbol named `DAVID`, because Lisp converts lowercase to uppercase letters in symbol names unless they are escaped using `\` or `||`.

The second, optional argument to `generate-entry`, `noun` in the example above, is discarded by `generate-entry`. It is allowed for compatibility with another program that uses the same dictionary files for a different purpose.<sup>29</sup>

**Alphabetization.** The first argument to `generate-entry` always refers to the main form of a lemma. It is treated as a heading in the concordance. Lemmata are sorted in the concordance in alphabetical order of the main forms. `ConcTeX` includes a special sorting routine.<sup>30</sup> It makes it possible to sort arbitrary special characters in a user-defined order by replacing ordinary characters and special character macros with 0 or more characters from Lisp's code table, which is based on the ASCII code table. In its current form, it allows up to 256 positions; however the actual number of special characters that can be sorted is much greater. This is because some characters occupy no position at all, others share a position with another character, and still oth-

<sup>28</sup> What `generate-entry` really does is a bit more complicated, but the effect is the same.

<sup>29</sup> This other program is called `LexTeX` and I plan to document it in a subsequent article. It's for generating dictionaries, vocabulary flashcards and similar things from files of Lisp code.

<sup>30</sup> The alphabetization routine for `ConcTeX` is essentially identical to the one in my `Spindex` package. For a more complete discussion of the alphabetization routine, see my previous article "Spindex—Indexing with Special Characters", *TUGboat* 18(4)/1998, pp. 255–273. Since `ConcTeX`, unlike `Spindex`, does not require a preliminary `TeX` run in order to generate page numbering information by means of `TeX`'s output routine, and therefore uses no `\write` commands, special character macros in the input file can be coded in the normal way (but with obligatory braces).

ers use the positions of more than one other character.

The string “David” is passed to the function `generate-info`, which passes each letter or special character coding to `letter-function` in order to generate a `sort-string`. Each letter or special character in the string is used to create a new string using characters from Lisp’s code table. The `sort-string` generated for “David” might be `"^D^A^[^K^D"`, depending on the way the alphabetization routine is set up.<sup>31</sup> The `sort-string` is put into the car of a cons cell, with the symbol bound to the word structure in the cdr.

```
("^D^A^[^K^D" . |David|)
```

This cons cell is then put into an association list (or alist) called `var-alist`. This alist will contain a cons cell for each of the word structures created by `generate-entry` in the lemmatization dictionary. If this is the lemmatization dictionary:

```
(generate-entry "David")
(generate-entry "eiga")
(generate-entry "eptir")
```

`var-alist` will look like this:

```
((("^D^A^[^K^D" . |David|)
 ("^F^K^H^A" . |eiga|)
 ("^F^U^Y^K^W" . |eptir|))
```

⚠ Formatting commands like `\it`, `\bf`, `\sc`, etc. are ignored when the `sort-string` is generated, otherwise, they would cause an error in `letter-function`.

Words in the lemmatization dictionary can contain special characters.

```
(generate-entry "dyr{\dh}ligr")
```

Note that two backslashes are necessary for the `{\dh}` in `generate-entry`’s argument. The resulting symbol that is bound to the word structure is `|dyr{dh}ligr|` (Engl. “glorious”), and the backslashes disappear in the symbol name.

⚠ The braces surrounding special character macros are needed so `generate-info` knows where they begin and end. `TeX`’s parser is cleverer than `generate-info`. When `TeX` finds an escape character, it reads the following tokens and uses them to make the longest control sequence possible. The function `generate-info`, on the other hand, needs delimiters

<sup>31</sup> Each project will have its own requirements with respect to alphabetization, so this must be customized by the user. The documentation supplied with `ConcTeX` explains in detail how to do this. The string `"^D^A^[^K^D"` consists of non-printing characters in Lisp’s printed representation.

for control sequences within words so it can pass them as a whole to `letter-function`.

**Subsidiary forms.** The function `generate-entry` can also have keyword arguments.

```
(generate-entry "dyr{\dh}ligr"
 :forms "dyr{\dh}lig")
```

The `:forms` argument tells `generate-entry` that “`dyrðlig`” is a subsidiary form of “`dyrðligr`” (in fact, it’s the feminine singular nominative and neuter plural nominative and accusative form). The string `"dyr{\dh}lig"` is used to access a symbol, `|dyr{dh}lig|`, which is bound to another word structure. The symbol `|dyr{dh}lig|` is added to the list in the forms slot of the word structure `|dyr{dh}ligr|`,<sup>32</sup> and the symbol `|dyr{dh}ligr|` is put into the root slot of the word structure `|dyr{dh}lig|`. The function `generate-info` is used to generate a `sort-string` for `|dyr{dh}lig|`, and the cons cells

```
("^D^^^W^E^P^K^H^W" . |dyr{dh}ligr|)
```

and

```
("^D^^^W^E^P^K^H" . |dyr{dh}lig|)
```

are added to `var-alist`. All of the word structures are added to `var-alist`, not just the ones for main forms of lemmata.

The `:forms` argument to `generate-entry` needn’t be a single string. It can also be a list of strings.

```
(generate-entry "dyr{\dh}ligr"
 :forms '("dyr{\dh}ligs" "dyr{\dh}ligum"
 "dyr{\dh}ligan"))
```

In this case, the subsidiary forms listed are the masculine singular genitive, dative, and accusative strong forms “`dyrðligs`”, “`dyrðligum`”, and “`dyrðligan`”. Each of the forms is used to access a symbol and bound to a new word structure. The symbol `|dyr{dh}ligr|` is put in the root slot of each of these word structures, and the symbols for the subsidiary forms are put into the list in the forms slot of `|dyr{dh}ligr|`.

```
(|dyr{dh}ligs| |dyr{dh}ligum| |dyr{dh}ligan|)
```

The forms are stored in the order in which they appear in the invocation of `generate-entry`. They will appear in this order in the concordance. They are not sorted alphabetically, because subsidiary forms should be arranged in the concordance according to grammatical criteria.

**Variants.** Some words in a manuscript may differ from the standard form that should appear in the concordance. For example, the word for “king”

<sup>32</sup> In Lisp, `nil` is the same as the empty list `()`, so it’s possible to add an element to the list `nil`.



in Old Icelandic is “konungr”, but sometimes the form “kongr” is used (cf. Modern Danish “konge”). If a manuscript uses both forms, “konungr” and “kongr”, but only “konungr” should appear in the concordance, `generate-entry` can be called using the `:variants` keyword argument.

```
(generate-entry "konungr" :variants "kongr")
```

The string “kongr” is used to access a symbol, `|kongr|`, but this symbol is not bound to a word structure; instead, it’s bound to the symbol `|konungr|`, i.e.,

```
(setq |kongr| ' |konungr|)
(symbol-value ' |kongr|) → |konungr|
```

(Again, what `generate-entry` really does is more complicated, but this is the effect.) The occurrences of the word “kongr” in the transcription, if any, will appear under “konungr” in the concordance.

The `:variants` keyword argument to `generate-entry` may also be a list of strings.

```
(generate-entry "konungr"
 :variants '("kongr" "ko{\n}gr"
 "kvnvngr" "konongr"))
```


In this case, all of the strings in the list are used to access symbols which are bound to `|konungr|`.

The symbols that are made from the string or strings in the `:forms` keyword argument to `generate-entry` can also have variants.

```
(generate-entry "eiga" :variants "ei{\g}a"
 :forms '("{\ 'a}"
 ("attv" :variants '("a{\t}v" "atto"))))
```

Here, there are two forms, “á” and “attv”, and the latter has two variants, “aTv” and “atto”. The `:forms` argument can be a list comprising a combination of simple strings and/or lists such as `("attv" :variants '("a{\t}v" "atto"))`.

```
(generate-entry "eiga" :variants "ei{\g}a"
 :forms '("eigir"
 ("eigi" :variants "ei{\g}i")
 "eigom"
 ("attv" :variants
 '("a{\t}v" "atto"))))
```

 The function `generate-entry` takes lists like `("eigi" :variants "ei{\g}i")` in its `:forms` ar-

gument and conses<sup>33</sup> (or puts) the symbol `sub-generate-entry` onto the front so it looks like this:

```
(sub-generate-entry "eigi"
 :variants "ei{\g}i")
```


Then it appends the list

```
(:root "eiga" :recursive t)
```

to the end of it, resulting in

```
(sub-generate-entry "eigi"
 :variants "ei{\g}i"
 :root "eiga" :recursive t)
```

and evaluates this list.<sup>34</sup>

 The lists in `generate-entry`’s `:forms` argument can also have `:forms` arguments.

```
(generate-entry "abc"
 :forms '("def" :forms "ghi"))
```

This is meaningless in the context of a concordance, since nesting in grammatical paradigms only has two levels. It might be useful, if variants were to appear in the concordance. Then they could be indented to the value of `\parindent`, and their forms could be indented to the value of `\parindent\parindent`. It might also be useful for some purpose other than a concordance. Changes to `process-line` and `export-entry` would be necessary to make deeper nesting work.

**Homograph identifiers** can be assigned to a word structure in two different ways: by using the `<string>` syntax, as in the input file,

```
(generate-entry "afl<nsg>")
```

or by using the `:homograph-identifier` keyword argument.

```
(generate-entry "afl"
 :homograph-identifier "nsg")
```

The string “nsg” stands for “neuter, singular”. In the concordance, this word will appear as “afl [nsg]”. If you want “afl [n. sg.]” or “afl [neut. sing.]” to appear in the concordance instead, you can write

```
(generate-entry "afl<n.~sg.>")
```

or

<sup>33</sup> In Lisp, a list is a chain of cons cells. Cons cells contain two elements, the `car` and the `cdr`, in that order. In the list (a b c d), the first cons cell has the symbol a in its `car` and a pointer to the following cons cell in its `cdr`. This in turn has the symbol b in its `car` and a pointer to the next cons cell in its `cdr`. The last cons cell in the list has d in its `car` and nil in its `cdr`: (d . nil). So the list (a b c d) is really (a . (b . (c . (d . nil)))).

<sup>34</sup> Since the original invocation of `generate-entry` called `sub-generate-entry`, as explained below, the latter is called recursively for subsidiary forms.

```
(generate-entry "afl"
  :homograph-identifier "n.~sg.")
```

or

```
(generate-entry "afl<neut.~sing.>")
```


or

```
(generate-entry "afl"
  :homograph-identifier "neut.~sing.")
```

instead. But “afl<nsg>” is more convenient to type in the input file than “afl<neut.~sing.>”, so it’s possible to use a cons cell as the :homograph-identifier argument to generate-entry.


```
(generate-entry "afl"
  :homograph-identifier
  '("nsg" . "neut.~sing."))
```

This way, you can type “afl<nsg>” in the input file, but “afl [*neut. sing.*]” will appear in the concordance.

 If you use both the <string> syntax and an explicit :homograph-identifier argument,

```
(generate-entry "afl<n.~sg.>"
  :homograph-identifier
  '("nsg" . "neut.~sing."))
```

the explicit :homograph-identifier argument takes precedence, and the homograph-identifier using the <string> syntax is discarded. It doesn’t matter whether it’s a simple string or a cons cell in the :homograph-identifier argument.

 When you use a homograph-identifier, no matter how you create it, the name of the symbol bound to the word structure includes the homograph-identifier. For instance, in the last example, the word structure is bound to the symbol |afl<nsg>|, its main-form is "afl", its tex-string is "afl [{{\it neut.~sing.\\\/}}]", and its homograph-identifier (i.e., the object stored in its homograph-identifier slot) is "nsg".

When the main form of a lemma is a homograph and has subsidiary forms, the latter may or may not also need homograph-identifiers. In Old Icelandic, there are two words spelled “afl”, a masculine noun meaning “an ironsmith’s forge” and a neuter noun meaning “physical strength”. If generate-entry is called like this:

```
(generate-entry "afl<nsg>"
  :forms '("afls" "afl"))
```

the genitive and dative singular forms, “afls” and “afl”, may appear in the input file with no homograph-identifier, and they will appear in the concordance as subsidiary forms of the neuter noun

“afl”. This is not very satisfactory, however, because the genitive and dative singular forms of the masculine “afl” are identical to the genitive and dative singular of the neuter “afl”. So it’s necessary to type in

```
(generate-entry "afl<nsg>"
  :forms '("afls<nsg>" "afl<nsg>"))
```

or

```
(generate-entry "afl"
  :homograph-identifier "nsg"
  :forms '("afls<nsg>" "afl<nsg>"))
```

or

```
(generate-entry "afl"
  :homograph-identifier "nsg"
  :forms '("afls"
    :homograph-identifier "nsg")
    "afl<nsg>"))
```

or one of the other possible alternatives. Note that the various possibilities of formulating the arguments are always allowed.

If no homograph-identifier is indicated for a subsidiary form of a lemma whose main form has one, the subsidiary form may be typed in the input file with no homograph-identifier. However, generate-entry automatically creates a variant using the homograph-identifier of the main form.

```
(generate-entry "afl<m>" :forms "aflar")
```

This results in a symbol |aflar| which is bound to a word structure, and a symbol |aflar<m>| which is bound to the symbol |aflar|. The word “aflar”, masculine plural nominative, is unambiguous and needs no homograph-identifier, since no other form of “afl” (masculine) or “afl” (neuter) is spelled the same way.

Another problem is that words in the same lemma sometimes have identical forms. The nominative and accusative, singular and plural of the neuter noun “afl” are all spelled “afl”. If it’s desirable to keep the morphological forms of lemmata separate in the concordance, it’s necessary to have different homograph-identifiers for the forms that are spelled the same.

```
(generate-entry "afl<nsg>"
  :forms '("afls<nsg>" "afl<nsg>"
    "afl<nsga>" "afl<npln>"
    "afl<npla>"))
```

Most of the time, the homograph-identifiers of subsidiary forms are not printed to the concordance. Since the homograph-identifier of the main form is printed to the concordance, indicating for instance that “afl” is a neuter noun, it’s not necessary to

indicate this fact for the subsidiary forms that appear below it. Sometimes, though, it's desirable to print something. The homograph-identifiers for subsidiary forms are printed to the concordance only when an explicit `:homograph-identifier` argument is used, not when the `<string>` syntax is used. The homograph-identifiers for main forms of lemmata are always printed to the concordance.

```
(generate-entry "afl<neut.>"
  :forms '("afls<nsg>" "afl<nsg>"
    '("afl"
      :homograph-identifier
      '("nsga" . "sg.~acc."))
    '("afl"
      :homograph-identifier
      '("npln" . "pl.~nom."))
    '("afl"
      :homograph-identifier
      '("npla" . "pl.~acc."))))
```

⇒

```
afl [neut.]
afls
afl
afl [sg. acc.]
afl [pl. nom.]
afl [pl. acc.]
```

Note that not all the information that's necessary for the homograph-identifier in the input file must be printed. The string `"afl<npln>"` in the input file indicates this form of "afl" unambiguously, i.e., neuter, plural, nominative. But only "pl. nom." need be printed to the concordance, because it's obvious it's the neuter noun.

The **tex-string** is what is written to the concordance. It is usually determined by a combination of `generate-entry`'s first argument and the homograph-identifier, if any. It is possible to override this by using the `:tex-string` keyword argument to `generate-entry`.

```
(generate-entry "abc" :tex-string "xyz")
```

This causes a symbol `|abc|` to be created and bound to a word structure, but "xyz" will be printed to the concordance. Occurrences of "abc" in the input file will appear under the heading "xyz", and the sort-string will be `^X^Y^Z` (depending on the values assigned by `letter-function`), i.e., based on the string "xyz" and not on "abc".

A `tex-string` can also be set by using a cons cell for the first argument to `generate-entry`.

```
(generate-entry '("eiga" . "agie"))
```

is equivalent to

```
(generate-entry "eiga" :tex-string "agie")
```

However, if you type

```
(generate-entry '("eiga" . "igea")
  :tex-string "agie")
```

the explicit `tex-string` keyword argument takes precedence over the `tex-string` in the `cdr` of the cons cell, which is discarded. I'm not sure whether the `tex-string` feature will prove to be useful, but it's available if needed.

◆ The function `generate-entry` allows the use of other keyword arguments. It doesn't matter what they are, `generate-entry` simply ignores them. This is for compatibility with the program `LexTeX`, which uses the lemmatization dictionary files for another purpose (see page 389), or for other programs that might be written. For instance,

```
(generate-entry "nema" verb
  :forms '("nam" "n{\ohook}mum"
    '("nvmenn" :variants "nomenn"))
  :class 'strong-4 :definition-english "take")
```

contains the keyword arguments `:class` and `:definition-english`, and their values. They are irrelevant to the function `generate-entry` in `ConcTeX`, which ignores them, but are used by the function called `generate-entry` in `LexTeX`.

◆ `ConcTeX` could be extended to do more than just produce a concordance. One possibility is to sort the words into a file of `TeX` code according to grammatical criteria. For instance, the words could be written to an output file nouns first, then adjectives, pronouns, verbs, adverbs, etc. Within these categories, the words could be sorted according to class. In order to accomplish this, more information must be entered in the lemmatization dictionary, i.e., `generate-entry` must be modified to process additional arguments. Alternatively, the homograph-identifiers could be used for this purpose.

◆ Actually, `generate-entry` isn't really a function at all. It's a macro, defined with `defmacro`. One reason for this is that a macro doesn't evaluate the arguments that are passed to it. If `generate-entry` were defined like this,

```
(defun generate-entry
  (main-form &optional word-type forms
    ...) ...)
```

then this would cause an error:

```
(generate-entry "{\ae}tt" noun
  :forms "{\ae}ttar")
```

because Lisp would try to evaluate the symbol `noun`, which is unbound. Since `generate-entry` is a macro,

```
(defmacro generate-entry (&rest arg) ...)
```

it's possible to manipulate its arguments before they're passed to the *real* function, which is called `sub-generate-entry`. Another reason for defining `generate-entry` as a macro is to make it easier to extend `ConcTeX`, or change the way it behaves. The arguments can be manipulated and passed to other functions, instead of or in addition to `sub-generate-entry`.

**The functions `harmless` and `add-variants`** are two other functions that can be used in the lemmatization dictionary. These are merely convenience functions, you can accomplish the same results using `generate-entry`.

The function `harmless` is for cases where none of `generate-entry`'s keyword arguments are needed. Some words, like prepositions and conjunctions (in the Germanic languages, at least) have no inflected forms, and some words may have no variants in a particular manuscript. (So far, I've never needed a `tex-string`.)

```
(harmless "{\\ 'a}<p>" "af"
          "{\\ '\\i}" "ok" "yfir")
```

is equivalent to

```
(generate-entry "{\\ 'a}<p>"
(generate-entry "af")
(generate-entry "{\\ '\\i}")
(generate-entry "ok")
(generate-entry "yfir"))
```

The function `harmless` takes one or more strings as its arguments, and calls `(generate-entry <string>)` for each of the strings. Note that a homograph-identifier is permitted, using the `<<string>>` syntax, as for `{\\ 'a}<p>`.

Calling `generate-entry` with a lot of forms and variants can be confusing,

```
(generate-entry "dyr{\\dh}ligr" adjective
  :forms '(("dyr{\\dh}lig"
  :variants "dyrdlig")
  ("dyr{\\dh}ligi" :variants "dyrdligi")
  ))
```

so the function `add-variants` can be used instead. This is equivalent to the previous example:

```
(generate-entry "dyr{\\dh}ligr" adjective
  :forms '("dyr{\\dh}lig" "dyr{\\dh}ligi"))

(add-variants "dyr{\\dh}lig" "dyrdlig")
(add-variants "dyr{\\dh}ligi" "dyrdligi")
```

All of the arguments to `add-variants` are strings. The first is used to access a symbol. This symbol might be bound to a `word structure`, which would be

fine. If, however, it's unbound, then a `word structure` is created, using the string as the only argument to `generate-entry`. Otherwise, if it's bound and not a `word structure`, `add-variants` issues a warning, and exits. Assuming the symbol is now a `word structure`, the other strings are used to access symbols which are set to the symbol derived from the first string, just as `generate-entry` sets variants (as described on page 390f).<sup>35</sup>

The order of the subsidiary forms is important. It will usually not be desirable to have them sorted in alphabetical order in the concordance, although it's possible; usually, the order of the subsidiary forms will be determined by the conventional order of the forms within the paradigm, e.g., singular nominative, genitive, dative, accusative, then plural nominative, genitive, dative, accusative for nouns in Old Icelandic. If you prefer the order nominative, accusative, genitive, dative, just type the forms in this order.

**The parser.** After the lemmatization dictionary has been loaded, `conctex.lsp` opens the input file and the function `main` reads it line-by-line. Entirely blank lines, and lines that begin with `%` or `\` are discarded (except for lines beginning with `\lineno`, `\putontop`, `\overstroke`, or other exceptions, if any). Lines that begin with `@` are Lisp code, the `@` is discarded, and the rest of the line, which must be a balanced expression (`sexp`), is evaluated. Other lines are text lines, and passed to the function `process-line`. This function strips off leading and trailing blanks, discards line numbers and punctuation from the beginning of the line, discards certain macros and perhaps some or all of their arguments, discards unattached math mode material, and replaces some strings with others. If a `@` or `%` appears in the middle of a line, `process-line` discards it together with the rest of the line. It also tests to see whether the end of the input line corresponds to the end of a transcript line. If the input line ends with `\\`, `-`, or `\-`, or if the following line is blank, `process-line` treats the end of the current line as the end of a transcript line and increments the line counter (called `line-counter`) when it's done with the current line. (Advancing the leaf, side or column requires an explicit call to `set-position`, as does advancing more than one line at a time, which is sometimes necessary, as when part of a leaf is missing or unreadable.) Then `process-line` passes the line to the function `read-word`. This function strips characters off the line until it has stripped off a whole word. Then it returns the word, `current-`

<sup>35</sup> Actually, `generate-entry` uses `add-variants` to set variants.

word, and the rest of the line to `process-line`. Now the routine for access and data storage takes over.

**Access and data storage.** The function `read-word` returns `current-word` as a string. The function `process-line` appends `|` to the beginning and end of the string, so that, for example, "`{\ae}tt`" becomes "`|{\ae}tt|`".<sup>36</sup> Now the string is used to access a symbol (or variable) using `read-from-string`, and the symbol `current-var` is set to this symbol, e.g., `|{\ae}tt|`. (In C, you'd say "current-var is a pointer to `|{\ae}tt|`".)

This is one reason why unmotivated braces are not permitted in the input file. The braces are needed to delimit the special characters for `generate-info` and to distinguish between special character codings and ordinary characters in the symbol names.<sup>37</sup> The vertical strokes surrounding the symbol name act to escape all of the other characters, which causes `\` to disappear when the string "`|{\ae}tt|`" is converted to the symbol `|{\ae}tt|`. If the input file contained the string "`{\ae}tt`"  $\rightarrow$  "aett" and "`{\ae}tt`"  $\rightarrow$  "aett" (using "ae" instead of the ligature), and unmotivated braces were permitted, they would both map to the same symbol, namely `{\ae}tt`, and they would not be kept separate in the concordance.

Now `process-line` checks to see if the *value* of `current-var` (the symbol `|{\ae}tt|`, in our example) is already bound. If it is, this means either that the word "aett" is accounted for in the lemmatization dictionary, or that it's occurred previously in the input file. If `|{\ae}tt|` is unbound, `process-line` checks whether the symbol `|{\Ae}tt|` is bound. If it is, `current-var` is set to `|{\Ae}tt|`, otherwise `process-line` checks whether the symbol `{\AE}TT` is bound.<sup>38</sup> If it is, `current-var` is set to `{\AE}TT`.<sup>39</sup>

<sup>36</sup> Backslashes in strings read in from a file are automatically escaped.

<sup>37</sup> By uncommenting two lines in `letter-function`, it's possible to allow unmotivated braces. I believe it's safer to make them signal an error, because it's a good way of discovering mistakes in the input file. Any cases where braces are required can be accounted for in `conctex.lsp`.

<sup>38</sup> No vertical strokes are needed around `{\AE}TT` because `{`, `}`, and capital letters don't need to be escaped.

<sup>39</sup> Manuscripts are often orthographically inconsistent, and transcriptions often use uppercase and lowercase letters to represent different letter forms, although the original letters do not really correspond to our upper- and lowercase. If the editor wants a particular word to be represented in the concordance in lowercase, capitalized or all capitals, and the first occurrence of this word in the transcription does not have the desired form, then the word must be entered in this form into the lemmatization dictionary. Of course, capitalized, upper- and lowercase strings which map

Actually, the functions `string-downcase`, `string-capitalize` and `string-upcase` are applied to (`symbol-name current-var`). The symbol `current-var` is evaluated once to get the symbol `|aett|`. If I wanted to examine the `symbol-name` of `|aett|` directly, I would have to type (`symbol-name '|{\ae}tt|`) to prevent evaluation of `|{\ae}tt|`. Note that applying `string-capitalize` to a symbol name beginning with a coding like `{\ae}` fails, because the capitalized string would need to be `{\AE}`. It is, of course, possible to have `\Ae` expand to "Æ" in  $\TeX$ , so that `{\Ae}` would yield the correct result, but it seems hardly worthwhile, since words should be accounted for in the lemmatization dictionary anyway.

**Unbound symbols.** If `|{\ae}tt|`, `|{\Ae}tt|`, and `{\AE}TT` are unbound, a word structure is created, and `|{\ae}tt|` is bound to it. A `sort-string` is generated and stored in `|{\ae}tt|`'s `sort-string` slot, and a `tex-string` is stored in its `tex-string` slot, just as for word structures created by the lemmatization dictionary. If the word read in from the file includes the symbols `<` and `>`, the string they enclose is considered to be the homograph-identifier, which is stored in the `homograph-identifier` slot, and affects the form of the `tex-string`.

After a word structure has been created for the symbol `|{\ae}tt|`, the symbol itself is put into the `cdr` of a cons cell, while its `sort-string` is put into the `car`, e.g.,

```
("^A^F^X^X" . |{\ae}tt|)
```

This cons cell is now appended to the association list (or alist) `word-alist`, *not* `var-alist`, as are the word structures from the lemmatization dictionary. The alist `word-alist` is used to store all of the entries that are main forms of lemmata, i.e., not subsidiary forms or variants, that actually occur in the transcription. They will appear as top-level entries in the concordance. Whenever a word in the input file maps to a symbol that is unbound, i.e., if it's the first occurrence in the input file and not in the lemmatization dictionary, this word is considered to be the main form of a lemma, so all lemmata with subsidiary forms and most variants must be accounted for in the lemmatization dictionary. Variants that differ only in the use of capitalization and upper- and lowercase letters, or where certain special character macros are replaced, are handled automatically (as above).

**Positions and occurrences** The word structure has an `occurrences` slot, where manuscript positions are stored. It might even have multiple

to different word structures are also possible, if they're bound in the lemmatization dictionary.

occurrences slots, if ConcTeX is being used for more than one manuscript. When a word structure is created in the lemmatization dictionary, there are no occurrences to be stored yet. When an entry is created for a word in the input file, the `position-string` is stored as a string in a list in the proper occurrences slot, e.g., ("28va~7"), using `access-occurrences`.

The current position in the transcription (and the manuscript) will have been generated by the function `set-position`, which is used in the input file, and `process-line`'s line counting routine. In addition, `process-line` can tell whether a particular word is the first or last word in the transcription line. It can sometimes be useful to indicate that a word is at the beginning or end of a line in a manuscript, because the use of unusual forms is often explainable because of the position. For instance, words at the beginning of lines may have fancy initials, and words at the end may be abbreviated in order that they may be squeezed into the remaining space on the line. If a word occurs at the beginning or end of a line, this may be indicated in the concordance, if the book designer wishes. For example, if this is a line at leaf 1, verso, column a:

```
\lineno{15} seg_{ir} e{\n} gaufgi
ken_{n}i~{\m}_{a{\dh}r} [ok enn]\
```

⇒

```
15: segir eN gaufgi kenni Maðr [ok enn]
```

then one of the positions for "segir" will be "1va 15<sup>b</sup>" and one of the positions for "enn" will be "1va 15<sup>e</sup>".

The program `conctex.lsp` doesn't print `^b$` and `^e$` explicitly to the concordance file. Instead, it uses the token lists `\linebeginstring` and `\lineendstring`,

```
\newtoks\linebeginstring
\newtoks\lineendstring
\linebeginstring={^b$}
\lineendstring={^e$}
```

and prints `\the\linebeginstring` and `\the\lineendstring` to the concordance file. To suppress "b" and "e" in the concordance, all that's necessary is to redefine the token lists `\linebeginstring` and `\lineendstring`.

```
\linebeginstring={}
\lineendstring={}
```

This feature will probably be turned off, because it is not customary for concordances to indicate the position of words in a line.

A concordance should only be generated for one manuscript at a time. This is merely the way I've set things up; it's not hard-wired into the program. The definition of the Lisp macro `access-occurrences`

depends on the value of the variable `manuscript`. It is used to access the appropriate occurrences slot for `setf`.<sup>40</sup>

Multiple occurrence slots are useful if you want to use the information generated by `conctex.lsp` in another program. The word structures can be written to a file using Lisp's read syntax. Another Lisp program can load this file, provided structures of type `word` are defined, and the data produced by the last run of `conctex.lsp` will be available. This could be useful for dictionaries or linguistic studies involving multiple sources.

**Bound symbols.** If the symbol pointed to by `current-var` (`|{ae}tt|` in our example) is bound, this means it was either bound in the lemmatization dictionary, or that the word "ætt" has already occurred in the input file, or both.<sup>41</sup> If it has already occurred in the input file, and it's a main form, `word-alist` already contains the cons cell ("<sup>A</sup>F<sup>X</sup>X" . `|{ae}tt|`), but if it's only bound because it appears in the lemmatization dictionary, or it's a subsidiary form or a variant, this cons cell will not be in `word-alist`. So, `process-line` checks to see whether a cons cell containing `|{ae}tt|` is in `word-alist`, using:

```
(rassoc current-var word-alist)
```

which searches for a cons cell in `word-alist` using the `cdr` as the search key. If it's there, it means that "ætt" has already occurred in the input file.

The situation is more complicated if the symbol is not already in `word-alist`. There are three possibilities: it could be the main form of a lemma, a subsidiary form, or a variant.

‡ If the symbol is a word-structure, it is either a main form or a subsidiary form.

‡ If the root slot of the word-structure is nil, then it's a main form. In this case, the cons cell containing the symbol and its sort-string is copied from `var-alist` into `word-alist`.

‡ Else, if the root slot of the word structure is non-nil, the symbol is a subsidiary form. In this case, its root slot is accessed in order to get its main form, which is another symbol bound to a word-structure. If the cons cell containing

<sup>40</sup> It's worth looking at the way `access-occurrences` is defined, because it illustrates a limitation of Lisp's `setf` access function.

<sup>41</sup> Or there's been a terrible mistake. It is unlikely that a symbol derived from a word in the transcription would conflict with a symbol that's already defined by `conctex.lsp` or the Lisp interpreter itself. However, if this problem arises, it would be easy enough to write safety routines to catch the problem symbols.

this symbol and its sort string is not already in `word-alist`, it's copied to it from `var-alist`.

- ‡ If the symbol evaluates to another symbol, it's a variant. In this case, the symbol is replaced by this other symbol, and this process is repeated for the latter.

The Lisp macro `access-occurrences` then appends the position-string for the new position, e.g., "28ra~15" to the appropriate occurrences slot in the appropriate `word-structure`.

**Exporting the concordance.** After every line in the input file has been processed, the loop in `main` returns nil. If you are generating a concordance using multiple input files, the first input file is closed, the next one is opened and processed in the same way. When all of the input files have been processed and closed, the alists `word-alist` and `var-alist` are sorted.

```
(setq word-alist
      (sort word-alist #'string< :key #'car))
(setq var-alist
      (sort var-alist #'string< :key #'car))
```

This puts the cons cells in `word-alist` and `var-alist` in alphabetical order according to their `cars`, i.e., their sort-strings. Now the sort-strings are now longer needed, so new lists are generated by popping off the cons cells and putting the `cdrs`, i.e., the symbols, into new lists, `word-list` and `var-list`. Only main forms of lemmata are members of `word-list`, and only of those lemmata, in which at least one form (which can be a variant) occurs in the transcription. Other main forms may be bound from the lemmatization dictionary, but they will not be in `word-list`. The list `word-list` is now passed to the function `export-words`, which writes the `TEX` file for the concordance. Exactly what `export-words` writes is a matter for the editor and the book designer, so this part of `ConcTEX` can be changed easily.

The function `export-words` pops the symbols off of the front of `word-list` one-by-one. The `tex-string` is written to the concordance file (here called `concordance.tex`, but any name within reason can be chosen), not indented. If there are no occurrences of the main form, the `tex-string` is enclosed in parentheses. If there are occurrences, the number of occurrences is printed after the `tex-string`, enclosed in parentheses, followed by the occurrences. Subsidiary forms are printed to the concordance if and only if there are occurrences for them.

If there are no occurrences and no subsidiary forms with occurrences, something is terribly wrong and `conctex.lsp` will signal an error. The way

`conctex.lsp` is set up, occurrences at the beginning and end of a line are indicated with a superscript, e.g., "27ra 31<sup>b</sup>" and "34vb 12<sup>e</sup>". On the other hand, if a word occurs multiple times within a line, not at the beginning or end and not broken, the position string is only printed out once, and the number of occurrences is indicated within parentheses, e.g., "51ra 5 (2)". If the word is broken, what is printed depends on whether it's broken across a line, column, side or leaf. In the extremely unlikely event that the word occurs at the beginning or end of a line, or broken, *and* multiple times within the line, the different cases are listed separately, e.g., "11ra 5<sup>b</sup>, 11ra 5 (2), 11ra 5-6".

When a symbol is popped off of `word-list`, `export-words` checks whether forms are present in the `word-structure`'s `forms` slot. The forms should be in the order in which they should be listed in the concordance. The function `export-words` accesses the symbol for each form, and checks the appropriate occurrences slot. If it's non-nil, the `tex-string` for that form is written to `concordance.tex`, with its occurrences, as above, but indented. If a form has no occurrences, nothing is printed to `concordance.tex`. The Lisp program currently does not print out variants in any way. This would require some additional programming, but it is possible.

◆ The program `conctex.lsp` can be extended to extract grammatical information from manuscript transcriptions. One way of doing this would be to use standardized homograph-identifiers. Another would be to add one or more slots to the `word-structure`. Then, `export-words` could be modified to write files for the various grammatical categories, which could then be concatenated. After `word-list` has been generated, `word-alist` and `var-alist` are no longer needed, and `var-list` is never needed by `conctex.lsp`. They are kept or generated only for debugging purposes.

**Other data.** When `conctex.lsp` is run, it generates some data in addition to the concordance. It counts the number of lines and words in each input file, the total number of lines, the total number of words, the total number of distinct words and the total number of lemmata. It prints this information to the `TEX` file `count_info.tex`.

◆ The number of lines in an input file is simply the value of `line-counter` at the end of the file. The number of words in an input file is the number of times `read-word` returns a non-empty string while that file is being read. The total number of distinct words is the number of times `export-occurrences`

is called and the total number of lemmata is the length of `word-list`.

**Hints on using ConcTeX.** Trying to generate a concordance from an entire manuscript transcription all at once is a recipe for disaster. I recommend testing the program with one column or side at first, gradually increasing the amount of text. In this way, the user can discover which lemmata are needed, which subsidiary forms need to be assigned to a lemma, which variants need to be accounted for, and so on.

The program `conctex.lsp` writes an additional TeX file containing only the words that are *not* included in the lemmatization dictionary, and their occurrences. This makes it easy to see which words should be lemmatized or declared harmless, so the lemmatization dictionary can be built up gradually, as longer and longer portions of the transcription are read by `conctex.lsp`.

A format for a manuscript transcription is likely to be rather complex and the user will probably discover places where new special characters, fonts and macros are needed. Every new control sequence defined in `conctex.tex` and used in transcription lines will require some alteration to the Lisp program, either in `process-line`, `read-word` and/or `letter-function`. I've intentionally programmed `conctex.lsp` in such a way as to allow changes to reflect different book designs, so it may be necessary to change `export-words`, too. The best way to proceed is to take small parts and run TeX and `conctex.lsp` on the input file to make sure that everything is working properly. When you've got it running smoothly, and you've built up a complete lemmatization dictionary, then you can try running TeX and `conctex.lsp` on the whole manuscript.

**Running TeX on the concordance.** Since ConcTeX generates the concordance from the TeX input file without running TeX, it's no problem to include `concordance.tex` (the TeX file output by `conctex.lsp` which contains the concordance) in the input file itself:

```
\input concordance
```

But it's safer to include it like this:

```
\newread\concordance
\openin\concordance=concordance
\ifeof\concordance
\message{concordance.tex doesn't exist.
  Not inputting it.}\else
\closein\concordance
\input concordance
\fi
```

If you use UNIX, you can ensure that the concordance is always up-to-date by using a shell script.

```
# This runs the concordance program.
gcl<conctex.lsp
# Now I run tex on my input file.
tex transcription
# This executes the file "warnings"
# It prints the warning messages
# from conclsp.lsp to standard output.
sh warnings
```

It's a good idea to use a shell script anyway, or the equivalent in your operating system, if you're using two-pass features, i.e., for an index, table of contents, page references, etc.

### A special case

Manuscripts often have peculiarities that are difficult to represent in set type. Often, these peculiarities occur too rarely for it to be worthwhile writing TeX macros and/or Lisp functions to cope with them, but it is often possible to invent an *ad hoc* solution. For instance, Holm perg 11 4° has words that are written vertically, and are therefore, so to speak, broken over several lines. This problem can be solved in the following way:

```
\newskip\tempskip
\newskip\normalbaselineskip
\tempskip=.75\baselineskip
\normalbaselineskip=\baselineskip
\font\enormous=cmr17 scaled 7500
\baselineskip=\tempskip
\setbox0=\hbox{\enormous D}}%
* \lineno{1\dash 7}\copy0
\vskip-\ht0\vskip-2pt
\dimen0=\wd0\advance\dimen0 by 18pt
\hangindent\dimen0\hangafter-7
r\break o\break t \break n \break i
\break n \break g\par
\baselineskip=\normalbaselineskip *
\vskip.667\baselineskip

@ (add-occurrences "drotning" "1va~1--7")

\lineno{8} himins _{ok}
iar{\dh}ar.~s{\ae}l {ok} dyr{\dh}\-
\lineno{9} lig m{\o}r Maria.~mo{\dh}ir
d_{ro}tti_{n}s\
```

Produces the following output:



1-7: **D**<sub>r  
o  
t  
n  
i  
n  
g</sub>

8: himins ok iarðar. sæl ok dyrð

9: lig mör Maria. moðir drottins

Since `process-line` ignores all lines within a commentary, the complex construction for the word “Drotning” isn’t read by `process-line`, so this occurrence needs to be set explicitly.

`@(add-occurrences |drotning| "1va~1-7")`

The function `add-occurrences` uses `access-occurrences` to access the appropriate `occurrences` slot of the `word` structure bound to the symbol in its first argument, and appends its second argument, a string, to the list in the `occurrences` slot. It can be any string, so `access-occurrences` can be used to make non-standard occurrences by hand. An occurrence like “1va 1-7” cannot be created by `conctex.lsp`’s ordinary routines.

### Final remarks

ConcTeX demonstrates the power of Lisp and TeX in combination. Designing and typesetting a manuscript transcription, usually as part of a facsimile edition, is a challenge under the best of circumstances, and generating a concordance is always a time-consuming task. I do not promise miracles with ConcTeX, but I do believe that it can make both of these tasks easier, and indeed possible for non-professionals.

ConcTeX’s most significant advantages are:

1. It makes it possible to take advantage of the typographic capabilities of TeX and METAFONT.
2. It uses the very same file for typesetting and generating the concordance.
3. It performs alphabetical sorting on arbitrary special characters

ConcTeX is designed to be extendable. It would be possible to adapt it for use with other languages. For languages that are written left-to-right, it will only be necessary to cope with the usual difficulties with fonts and character encoding. For right-to-left text or a mixture of left-to-right and right-to-left text, the difficulties are greater, but it should be possible to solve them. Many of its features are of general utility and could be used for other kinds of programs that extract data from TeX input files.

However, there is one significant problem from the point of view of book design. The virtues of

Computer Modern and its offspring notwithstanding, there simply aren’t enough METAFONT fonts in the public domain suitable for use in fine printed books. Understandably, most of the other fonts available either contain special symbols or alphabets for non-Western languages, and are generally designed to be compatible with Computer Modern. But even if the Computer Modern fonts were the most beautiful fonts in the world, not every book should be printed in them. I admire Knuth’s accomplishment and I like Computer Modern, but Monotype Modern 8A, on which it is based, is not universally admired.<sup>42</sup> It is possible to use PostScript fonts with TeX, but it’s inconvenient, and although they are well-designed, they have the serious defect that most sizes are produced by simple magnification or reduction. I consider it an important desideratum that more METAFONT fonts are created and made available, but I don’t see how this goal can be accomplished by amateur programmers writing free software. But until such fonts are available, books typeset without the financial and technical backing of a publisher will continue to suffer from a poverty of fonts.

I am far from being an authority on the subject, but I doubt very much that any photolithographic printing technique will ever be able to equal the quality of impression of lead type. My dream is to use METAFONT for designing lead type for use on a TeX-driven, Linotype-like linecasting machine. I say a linecasting machine only because I think that it would be easier to implement glue on a linecasting machine. It might, however, be an interesting exercise to design a TeX-driven Monotype-like typesetting machine, or even a typesetting machine based on a different principle, such as casting an entire page at once. Perhaps with such machines, we can finally equal and perhaps even surpass the great typographic achievements of the past.

### Sample texts and concordances

Concordances become long very quickly, so this section contains three mini-concordances and the texts from which they were generated.

The first text is the beginning of the transcription of Holm perg 11 4<sup>o</sup>, prepared by Dr. Wilhelm Heizmann. It illustrates the use of *line footnotes*. These are footnotes which refer to lines in the

<sup>42</sup> WILLIAMSON, in reference to the English Monotype Corporation’s Modern Extended series 7: “Not particularly distinguished in letter form, the face has become familiar to readers of scientific works; for some years, this was one of the few series equipped with a full range of mathematical and other special sorts.” (p. 130)

transcription. Unlike ordinary footnotes, there is no indication in the running text, such as an asterisk, a dagger, or a superscripted numeral, but the line number or numbers appear in the footnote where the footnote indicator usually goes.

Note that a different definition of `\ustroke` is used here than in the foregoing article. The emendations are in italics and underlined, but not enclosed in brackets.

Holm perg 11 4<sup>o</sup>  
1va

próloGus

1–7: **D**  
r  
o  
t  
n  
i  
n  
g

- 8: himins ok iarðar. sæl ok dyrð  
9: lig mör Maria. moðir drottins  
10: Iesus Xristz. blomi hreinlifis.  
11: herbergi heilags anda. øllvm  
12: helgum mønnum æðri. helgari  
13: ok haleitari. er komin at kyn  
14: ferðj af kongligri ætt eptir þvi sem  
15: segir en gaufgi ken*ni* Maðr [ok enn]

Concordance to Holm perg 11 4<sup>o</sup>

- æðri (1): 1va 12.  
ætt (1): 1va 14.  
af (1): 1va 14.  
(allr)  
øllvm (1): 1va 11<sup>e</sup>.  
(andi)  
anda (1): 1va 11.  
at (1): 1va 13.  
blomi (1): 1va 10.  
drotning (1): 1va 1-7.  
(drottinn)  
drottins (1): 1va 9<sup>e</sup>.  
(dyrðligr)  
dyrðlig (1): 1va 8–9.  
enn (2): 1va 15, 15<sup>e</sup>.  
eptir (1): 1va 14.  
er (1): 1va 13.  
(göfugr)

1–7 Drotning, bis auf das initiale D sind die Buchstaben untereinander angeordnet.

8/9 dyrðlig, dafür in Ausg. zumeist dýrlig.

10 Iesus, 233 Jesu, Ausg. für den Genitiv immer Jesu; Xristz, 233 cristi, Ausg. für Xrist- fast immer Crist-, einige Male auch Krist-.

- gaufgi (1): 1va 15.  
(haleitr)  
haleitari (1): 1va 13.  
(heilagr)  
heilags (1): 1va 11.  
helgari (1): 1va 12<sup>e</sup>.  
helgum (1): 1va 12<sup>b</sup>.  
herbergi (1): 1va 11.  
(himinn)  
himins (1): 1va 8.  
(hreinlifi)  
hreinlifis (1): 1va 10<sup>e</sup>.  
Iesus (1): 1va 10<sup>b</sup>.  
(iørðr)  
iarðar (1): 1va 8.  
kennimaðr (1): 1va 15.  
(koma)  
komin (1): 1va 13.  
(kongligr)  
kongligri (1): 1va 14.  
(kynferð)  
kynferðj (1): 1va 13–14.  
(maðr)  
mønnum (1): 1va 12.  
Maria (1): 1va 9.  
moðir (1): 1va 9.  
mör (1): 1va 9.  
ok (4): 1va 8 (2), 13<sup>b</sup>, 15.  
(sæll)  
sæl (1): 1va 8.  
(segja)  
segir (1): 1va 15<sup>b</sup>.  
sem (1): 1va 14<sup>e</sup>.  
(Xrist)  
Xristz (1): 1va 10.  
þvi (1): 1va 14.

The Bible

Genesis

1. In the beginning God created the heavens and the earth. 2. The earth was without form and void, and darkness was upon the face of the deep; and the Spirit of God was moving over the face of the waters. 3. And God said, “Let there be light”; and there was light. 4. And God saw that the light was good; and God separated the light from the darkness. 5. God called the light Day, and the darkness he called Night. And there was evening and there was morning, one day.

Concordance to the Bible

and (11): *Gen.* 1:1, 2 (3), 3 (2), 4 (2), 5 (3).

be (1): *Gen.* 1:3.  
 was (7): *Gen.* 1:2 (3), 3, 4, 5 (2).  
 beginning (1): *Gen.* 1:1.  
 (call)  
 called (2): *Gen.* 1:5 (2).  
 (create)  
 created (1): *Gen.* 1:1.  
 darkness (3): *Gen.* 1:2, 4, 5.  
 day (2): *Gen.* 1:5 (2).  
 deep (1): *Gen.* 1:2.  
 earth (2): *Gen.* 1:1, 2.  
 evening (1): *Gen.* 1:5.  
 face (2): *Gen.* 1:2 (2).  
 form (1): *Gen.* 1:2.  
 from (1): *Gen.* 1:4.  
 God (6): *Gen.* 1:1, 2, 3, 4 (2), 5.  
 good (1): *Gen.* 1:4.  
 he (1): *Gen.* 1:5.  
 (heaven)  
 heavens (1): *Gen.* 1:1.  
 in (1): *Gen.* 1:1.  
 let (1): *Gen.* 1:3.  
 light (5): *Gen.* 1:3 (2), 4 (2), 5.  
 morning (1): *Gen.* 1:5.  
 (move)  
 moving (1): *Gen.* 1:2.  
 night (1): *Gen.* 1:5.  
 of (3): *Gen.* 1:2 (3).  
 one (1): *Gen.* 1:5.  
 over (1): *Gen.* 1:2.  
 (say)  
 said (1): *Gen.* 1:3.  
 (see)  
 saw (1): *Gen.* 1:4.  
 (separate)  
 separated (1): *Gen.* 1:4.  
 spirit (1): *Gen.* 1:2.  
 that (1): *Gen.* 1:4.  
 the (14): *Gen.* 1:1 (3), 2 (6), 4 (3), 5 (2).  
 there (4): *Gen.* 1:3 (2), 5 (2).  
 upon (1): *Gen.* 1:2.  
 void (1): *Gen.* 1:2.  
 (water)  
 waters (1): *Gen.* 1:2.  
 without (1): *Gen.* 1:2.

## Die Bibel

## Das 1. Buch Mose (Genesis)

**(With homograph identifiers.)**

1. Am Anfang schuf Gott Himmel und Erde.  
 2. Und die{fsn} Erde war wüst und leer, und es war finster auf der{fds} Tiefe; und der{mns} Geist Gottes schwebte auf dem{nds} Wasser. 3. Und Gott

sprach: Es werde Licht! Und es ward Licht. 4. Und Gott sah, daß das{nas} Licht gut war. Da schied Gott das{nas} Licht von der{fds} Finsternis 5. Und nannte das{nas} Licht Tag und die{fas} Finsternis Nacht. Da ward aus Abend und Morgen der{mns} erste Tag.

## Die Bibel

## Das 1. Buch Mose (Genesis)

**(Without homograph identifiers.)**

1. Am Anfang schuf Gott Himmel und Erde.  
 2. Und die Erde war wüst und leer, und es war finster auf der Tiefe; und der Geist Gottes schwebte auf dem Wasser. 3. Und Gott sprach: Es werde Licht! Und es ward Licht. 4. Und Gott sah, daß das Licht gut war. Da schied Gott das Licht von der Finsternis 5. Und nannte das Licht Tag und die Finsternis Nacht. Da ward aus Abend und Morgen der erste Tag.

## Konkordanz zur Bibel

Abend (1): *Gen.* 1:5.  
 (an)  
 am (1): *Gen.* 1:1.  
 Anfang (1): *Gen.* 1:1.  
 auf (2): *Gen.* 1:2 (2).  
 aus (1): *Gen.* 1:5.  
 da (1): *Gen.* 1:5.  
 (das [n.]  
 das [akk. sg.] (2): *Gen.* 1:4, 5.  
 dem [dat. sg.] (1): *Gen.* 1:2.  
 der [m.] (2): *Gen.* 1:2, 5.  
 die [f.] (1): *Gen.* 1:2.  
 die [akk. sg.] (1): *Gen.* 1:5.  
 der [dat. sg.] (2): *Gen.* 1:2, 4.  
 Erde (2): *Gen.* 1:1, 2.  
 (erst)  
 erste (1): *Gen.* 1:5.  
 es (3): *Gen.* 1:2, 3 (2).  
 finster (1): *Gen.* 1:2.  
 Finsternis (2): *Gen.* 1:4, 5.  
 Geist (1): *Gen.* 1:2.  
 Gott (2): *Gen.* 1:1, 3.  
 Gottes (1): *Gen.* 1:2.  
 Himmel (1): *Gen.* 1:1.  
 leer (1): *Gen.* 1:2.  
 Licht (4): *Gen.* 1:3 (2), 4, 5.  
 Morgen (1): *Gen.* 1:5.  
 Nacht (1): *Gen.* 1:5.  
 (nennen)  
 nannte (1): *Gen.* 1:5.  
 (schaffen)

schuf (1): *Gen.* 1:1.  
 (schweben)  
 schwebte (1): *Gen.* 1:2.  
 (sein [*verb*])  
 war (2): *Gen.* 1:2 (2).  
 (sprechen)  
 sprach (1): *Gen.* 1:3.  
 Tag (2): *Gen.* 1:5 (2).  
 Tiefe (1): *Gen.* 1:2.  
 und (10): *Gen.* 1:1, 2 (4), 3 (2), 5 (3).  
 von (1): *Gen.* 1:4.  
 Wasser (1): *Gen.* 1:2.  
 (werden)  
 werde (1): *Gen.* 1:3.  
 ward (2): *Gen.* 1:3, 5.  
 wüst (1): *Gen.* 1:2.

### Category code list

- `\catcode'\<=\active`. For homograph identifiers. Reset to 12 (other) in math mode.
- `\catcode'\@=14` (Comment). Treated as a comment by  $\TeX$ , equivalent to `%`. If `@` is the first non-blank character in an input line, the Lisp interpreter evaluates the rest of the line, which should contain a balanced expression (`sexp`). Otherwise, if it's in a text line, `conctex.lsp` discards it and the rest of the line following it.
- `\catcode'\#9` (Ignored). The character `¥` (decimal 165, octal `245`, hexadecimal `A5`) is ignored by  $\TeX$  and treated as a *word separator* by Lisp.
- `\catcode'\*=9` or `\active`. Used for commentaries. Reset to `\catcode 12` in math mode.
- `\catcode'\_=\active`. The underline character is `\let` to `\ustroke`. Reset to 8 (subscript) in the `\plain` environment, so `_` can appear in the names of files loaded using `\input`, and math mode, so it can be used for subscripts.
- `\catcode'\-=\active`. The hyphen character `-` is used for line ends where words are broken. Reset to 12 (other) in commentaries, the `\plain` environment, and math mode.
- `\catcode'\^~M=\active`. This change is local to the expansions of the active character `-` and the redefined control symbol `\-`, where `^~M` is `\let` to `\empty`. Otherwise, it has its normal `\catcode` of 5 (end of line).
- `\catcode'\©=\active`. The copyright symbol (decimal 169, octal `251`, hexadecimal `A9`) is `\let` to `\-` before the latter is redefined, so it

can be used for discretionary hyphens within transcription lines, if necessary.

### Glossary

**Braces, unmotivated:** Braces that delimit unnecessary groups in the input file. Not permitted in *transcription lines*.

**Comment:** Comments can be normal  $\TeX$  comments that follow a `%`. Usually, however, “comment” refers to invocations of the macros `\begincomment` and `\endcomment`.

```
\begincomment{This is a comment.}
\endcomment
```

Comments appear in the output only if `\drafttrue`. Comments are ignored by `conctex.lsp`.

**Commentary:** A commentary contains text which should appear within the *transcription lines* in the *output*, but which should not be used for generating the concordance. Commentaries can be coded in several ways.

```
* This is a commentary. *
\begincommentary This is also
  a commentary.\endcommentary
\plain Yet another commentary.
\endplain
```

**Evaluated lines:** Lines in the *input file*, where `@` is the first non-blank character. Such lines must contain a balanced Lisp expression, which is evaluated by the Lisp interpreter.  $\TeX$  ignores lines beginning with `@`.

**Homograph identifiers:** In  $\TeX$ , strings of the form `<<(string)>>`. Used for indicating homographs in the input file. Printed out or not, depending on the value of `\ifdraft`. For the Lisp program, homograph-identifiers can be set in various ways in the lemmatization dictionary.

**Ignored lines:** Lines in the *input file* that are ignored by  $\TeX$ , `conctex.lsp`, or both. Completely blank lines are ignored by `conctex.lsp` and treated normally by  $\TeX$ . Lines beginning with `%` are ignored both by  $\TeX$  and `conctex.lsp`. If a line contains a `%` in any other position, the rest of the line is discarded by both  $\TeX$  and `conctex.lsp`. The character `@` is equivalent to `%` in  $\TeX$ . If a line contains an `@` that's not the first non-blank character in the line, `conctex.lsp` discards the rest of the line.

**Input file:** A file containing text to be typeset for a book containing a facsimile of a manuscript, or

something similar. The input file or files are also used for generating a concordance.

**Lemmatization dictionary:** A file of Lisp code used for lemmatizing the words in the transcription. The file can contain invocations of the functions `generate-entry`, `add-variants`, and/or `harmless`.

**Macro, delimited:** A macro within braces, like `{\%}`, `{\rm ...}`, or `{\dh}`

**Macro, undelimited:** A macro with no enclosing braces, like `\%`, `\rm` or `\indexentry{nouns}{x}%{verbs}{}`.

**Output:** The typeset result of running  $\TeX$  on the *input file* or *files*. Technically, the result of running  $\TeX$  is a `dvi` file, however I usually mean either the paper printout or a display on the computer terminal in a program like `xdvi` or `Ghostview`.

**Text lines:** Lines in the *input file* which are processed by  $\TeX$ . They may be *transcription lines* or *commentaries*.

**Transcription lines:** Lines in the *input file* containing the transcription of the manuscript. They should correspond, for the most part, to the actual lines in the manuscript; however, they may also contain *commentaries*, which may affect the length and hence the line breaking in the output.

**Word element:** A character which `conctex.lsp` considers to be part of a word. Includes all letters (characters whose `\catcode` is 11), some characters of type “other” (`\catcode` 12), and special character macros.

**Word separator:** Characters that cannot be part of a word. Currently these are blanks, punctuation, and the character which is represented as  $\text{Y}$  on my terminal (decimal 165, octal 245, hexadecimal A5).

## References

- The Bible. Containing the Old and New Testaments. Revised Standard Version.* American Bible Society. New York: 1952.
- Die Bibel. Mit Apokryphen.* Nach der Übersetzung Martin Luthers neubearbeitet. Deutsche Bibelgesellschaft. Stuttgart: 1985.
- KNUTH, Donald E. *The  $\TeX$ book.* Addison-Wesley Publishing Company. Reading, Mass.: 1986.
- STEELE, Guy L., *Common Lisp. The Language.* 2<sup>nd</sup> ed. Digital Press. 1990.
- CLEASBY, Richard and Gudbrand Vigfusson. *An Icelandic-English Dictionary.* 2<sup>nd</sup> ed. The Clarendon Press. Oxford: 1957.

WILLIAMSON, Hugh. *Methods of Book Design. The Practice of an Industrial Craft.* 3<sup>rd</sup> ed. Yale University Press. New Haven: 1983.

WINSTON, Patrick Henry and Berthold Klaus Paul Horn. *LISP.* 3<sup>rd</sup> ed. Addison-Wesley Publishing Company. Reading, Mass.: 1989.

◇ Laurence Finston  
Skandinavisches Seminar  
Georg-August-Universität  
Humboldtallee 13  
D-37073 Göttingen, Germany  
lfinsto1@gwdg.de

## Language Support

### Cyrillic encodings for $\text{\LaTeX} 2_{\epsilon}$ multi-language documents

A. Berdnikov, O. Lapko, M. Kolodin,  
A. Janishevsky, A. Burykin

#### Abstract

This paper describes the X2 and the T2A, T2B, T2C encodings designed to support Cyrillic writing systems for the multi-language mode of  $\text{\LaTeX} 2_{\epsilon}$ . The encoding X2 is the “Cyrillic glyph container” which can be used to insert into  $\text{\LaTeX} 2_{\epsilon}$  documents text fragments from all modern Cyrillic writings, but it does not strictly obey all the rules of  $\text{\LaTeX} 2_{\epsilon}$ . The encodings T2A, T2B and T2C are the “true”  $\text{\LaTeX} 2_{\epsilon}$  encodings which satisfies all the requirements of the  $\text{\LaTeX} 2_{\epsilon}$  kernel, but as a result three encodings are necessary to support the whole variety of languages based on the Cyrillic alphabet.

These restrictions of the  $\text{\LaTeX} 2_{\epsilon}$  kernel, the specific features of Cyrillic writing systems and the basic principles used to create the encodings X2 and T2A, T2B, T2C are considered. This project supports all the Cyrillic writing systems known to us, although the majority of the accented letters need to be constructed using internal  $\text{\TeX}$  tools. The X2 encoding was approved at *CyrTUG-97*—the annual conference of Russian-speaking  $\text{\TeX}$  users and was previously presented at the *EuroTeX-98* Conference. The encodings X2, T2A, T2B and T2C were intensively discussed on the *cyrtext-2* mailing list.

## 1 Introduction

The project (originally named T2) to produce encodings necessary to support modern Cyrillic languages in  $\text{\LaTeX} 2_{\epsilon}$  multi-language mode was initiated at the *TUG-96* Conference in Dubna. This paper presents the results of that project. Although some minor corrections could still appear as new information about minor Cyrillic writings appear, the kernel of the project seems to be stable. The encoding support includes:

- L $\text{H}$ font font collection version 3.2—the Computer Modern Cyrillic fonts and European Computer Modern Cyrillic fonts created by O. Lapko,
- T2enc macro package created by V. Volovich and W. Lemberg—the input and output encoding and font definitions necessary for the  $\text{\LaTeX} 2_{\epsilon}$  packages `fontenc` and `inputenc`,
- the hyphenation patterns in encoding-independent style: `ashyphen` by A. Slepuhin, `ruhyphen` by D. Vulis, `lvhyphen` by M. Vorontsova and S. Lvovski, `znhypen` by S. Znamenskii,
- `rusbabel` package created by V. Volovich and W. Lemberg to support Cyrillics (based on new encodings) in `BABEL`.

The  $\beta$ -versions of these packages are on the *T $\text{E}$ XLive 3* CD-ROM distribution. The final versions are available on CTAN.<sup>1</sup>

Support for Cyrillics includes the following encodings:

- X2—the Cyrillic glyph container which contains all the glyphs necessary to support modern Cyrillic writing. It does not obey all the specifications that  $\text{\LaTeX} 2_{\epsilon}$  requires for an encoding with the prefix T, but as a result it is enough to have just one encoding to insert in  $\text{\LaTeX} 2_{\epsilon}$  documents the characters, words, names, bibliography references, short sentences, citations, etc., specific for all modern Cyrillic writings without too large an increase in the number of fonts used for this purpose (i.e., this encoding is a tool which enables Latin-writing people to occasionally use Cyrillics in their documents). The price is that some Cyrillic letters do not exist as a separate glyphs but must be composed from pieces (accents and modifiers) contained in X2, and the user must obey some rules of safety (described in section 6) since X2 does not satisfy all the requirements obligatory for T $\langle n \rangle$  encodings;

- T2A, T2B, T2C—the encodings which *strictly* satisfy the requirements necessary for  $\text{\LaTeX} 2_{\epsilon}$  multi-language mode. With these encodings it is safe to mix different languages inside your documents and to use large pieces of text without any problems. The price is that it is necessary to have three encodings (and an enormous number of fonts each in agreement with the encoding conventions of the European Computer Modern fonts) to support the whole variety of Cyrillic alphabets. Some Cyrillic languages are supported by one or two encodings from the T2\* encoding family, some are supported by all three encodings. The encodings are in agreement with the  $\text{\LaTeX} 2_{\epsilon}$  multi-language mode and encoding paradigm. Although there are no obstacles to the use of these encodings for original Cyrillic texts, native users may have different preferences;
- LR0 and LR1—the encodings which combine the OT1 encoding for 0–127 and Cyrillic letters and symbols from the leading languages of the Former Soviet Union for 128–255. (The encoding LR0 contains the Russian letters only, the encoding LR1 contains in addition the most frequently used national letters.) These encodings are as close as possible to X2 and T2A/T2B/T2C and are designed mainly for non- $\text{\LaTeX}$  formats based on the CM font family and the original *Plain*  $\text{\TeX}$ . There is some hope (not confirmed at this moment) that LR0 and LR1 may become the inter-platform and inter-format standard for representing Russian letters inside  $\text{\TeX}$  (as ASCII is the standard for English);
- LR $\langle n \rangle$ —local encodings necessary to support individual Cyrillic languages. The T2\* encodings are intended for the multi-language mode of  $\text{\LaTeX} 2_{\epsilon}$  and for this reason may be not well suited to bilingual documents or to the preferences of the native users. It is definitely not the task of the T2 working group, but that of the national  $\text{\TeX}$  User Groups, to organize local encodings that are useful for their language;
- TS2—the encoding containing accents, non-letter symbols, etc., necessary for Cyrillic writings which are outside X2 and T2\* encodings. This encoding is under consideration, and with great probability all necessary additional glyphs could be added to TS1 encoding. (The latter case has the advantage that it prevents the increase of the number of fonts needed to support multi-language mode in  $\text{\LaTeX} 2_{\epsilon}$ );

<sup>1</sup> /fonts/cyrillic, /languages/hyphenation/ruhyphen, and /macros/latex/contrib/supported/t2.

- LWN—the encoding which generalizes the WNCyr font family by adding new Cyrillic letters and new substitution pairs (ligatures) based on ASCII Latin input. It is suitable for Latin-writing users who use Cyrillics only occasionally. (This encoding is still under development and is not described here.);
- T5 encoding(s) to support Old Slavonic, Glagolitic, Church Slavonic, etc., writings. The project to develop these encodings has just started and its discussion is outside the scope of this publication.

## 2 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> system of encodings

The following types of encodings are recognised by the L<sup>A</sup>T<sub>E</sub>X Project:<sup>2</sup>

OT $\langle n \rangle$ —essentially 7 bit ‘old’ encodings. Typically these will be small modifications of the original T<sub>E</sub>X encoding, OT1 (for example, OT4, a variant for Polish).

T $\langle n \rangle$ —8 bit Text Encodings. T $\langle n \rangle$  encodings are the main text encodings that L<sup>A</sup>T<sub>E</sub>X uses. They have some essential technical restrictions to enable multilingual documents with standard T<sub>E</sub>X: (a) they should have the basic Latin alphabet, the digits and punctuation symbols in the ASCII positions, (b) they should be constructed so that they are compatible with the lowercase code used by T1. Further discussion of the technical requirements for T $\langle n \rangle$  encodings is given in section 3.

X $\langle n \rangle$ —other 8 bit Text Encodings (eXtended, or eXtra, or X=Non Latin). Sometimes it may be necessary, or convenient, to produce an encoding that does not meet the restrictions placed on the T $\langle n \rangle$  encodings. Essentially arbitrary text encodings may be registered as X $\langle n \rangle$ , but it is the responsibility of the maintainers of the encoding to clearly document any restrictions on the use of the encoding.

TS $\langle n \rangle$ —Text Symbol Encodings. Encodings of symbols that are designed to match a corresponding text encoding (for example, paragraph signs, alternative digit forms, etc.). The font style of fonts in TS $\langle n \rangle$  encoding will ordinarily be changed in parallel with that of the fonts in T $\langle n \rangle$  encoding using NFSS mechanisms. As a result, at any moment the TS $\langle n \rangle$  font style is compatible with the T $\langle n \rangle$  font and the glyphs from TS $\langle n \rangle$  font (accents, punctu-

ation symbols, etc.) can be mixed with the glyphs from the corresponding T $\langle n \rangle$  font.

S $\langle n \rangle$ —Symbol encodings. The style of fonts in S $\langle n \rangle$  encoding need not be synchronized with that of T $\langle n \rangle$  fonts. These encodings are used for arbitrary symbols, ‘dingbats’, ornaments, frame elements, etc.

A $\langle n \rangle$ —Encodings for special Applications (not currently used).

E\*—Experimental encodings but those intended for wide distribution (currently used for the ET5 proposal for Vietnamese).

L\*—Local, unregistered encodings (for example, the LR0, LR1 and LR $\langle n \rangle$  encodings mentioned above).

OM\*—7 bit Mathematics encodings.

M\*—8 bit Mathematics encodings.

U—Unknown (or unclassified) encoding.

## 3 Specifications for T $\langle n \rangle$ and X $\langle n \rangle$ encodings

There are two main restrictions to be fulfilled before an encoding may be considered as an encoding with the prefix ‘T’ satisfying the requirements of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> kernel:

- the `\lccode`–`\uccode` pairs should be the same as they are in the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> kernel (i.e., as they are in the T1 encoding);
- the Latin characters and symbols: !, ', (, ), \*, +, ,, -, ., /, :, ;, =, ?, [, ], ‘, |, @ (questionable), 0–9, A–Z, a–z should be at the positions corresponding to ASCII, and the symbols produced by the ligatures --, ---, ‘‘, ’’ (at arbitrary positions).

If the encoding requires the redefinition of the values `\lccode`–`\uccode`, or if it does not contain the necessary Latin characters in the ASCII positions, it will produce undesirable effects in some situations inside L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> and will make use of the encoding incompatible with the general multi-language mode. The reasons for such restrictions are explained in detail in [1].

Although the L<sup>A</sup>T<sub>E</sub>X Team’s technical specifications for X $\langle n \rangle$  encodings are less restrictive than those for ‘ordinary’ text encodings, there are corresponding restrictions on their use, and some *desirable* properties for them to have. In particular:

- If the encoding does not have Latin letters in ASCII slots then the users must take care not to enter such text, otherwise ‘random’ incorrect output will be produced, with no warning from

<sup>2</sup> The following text is slightly adapted from a post by David Carlisle to the mailing list `cyrtex-t2`.



the  $\LaTeX$  system. Also, care must be taken with ‘moving’ text that is generated internally within  $\LaTeX$  (such as cross references), which may fail if the encodings change;

- To reduce the problems with cross reference information, the  $\LaTeX$  maintainers strongly recommend that at least the digits and ‘common’ punctuation characters are placed in their ASCII slots;
- If the encoding uses a lowercase table that is incompatible with the lowercase table of T1, then it is not possible to mix this encoding and a T $\langle n \rangle$  encoding within a single paragraph, and obtain correct hyphenation with standard  $\TeX$ .

If the X $\langle n \rangle$  encoding does not use a lowercase table that is compatible with that of T1, the package supporting this encoding should ensure that encoding switches only happen between paragraphs (or that hyphenation is suppressed when temporarily switching to the new encoding). It should be noted that this restriction on the lowercase table applies *only* to systems using standard  $\TeX$  (version 3 and later). Using  $\varepsilon$ - $\TeX$  version 2 will remove the need for this restriction as the hyphenation system has been improved — it will use a suitable lowercase table for each language (the table will be stored along with each language hyphenation table), and surely it deals not at all with the *Omega* system.

#### 4 “Cyrillic glyph container” — the X2 encoding

The encoding X2 should include all the glyphs necessary to represent in  $\LaTeX 2_\varepsilon$  documents containing texts from stable Cyrillic languages. The basis of X2 is the Russian alphabet (since it is the main language used for publication in Cyrillic). Taking account of the variety of old Cyrillic texts, only those modern alphabets which are still in use are included in X2. The exceptions are the characters Ъ/ъ, Ѡ/ѡ, V/v which were used in Russian and Bulgarian texts at the beginning of the 20<sup>th</sup> century.

The X2 encoding is designed so that by combining "00–"7f from OT1 and "80–"ff from X2 one can construct an encoding which is adequate to support the most common Cyrillic languages. This permits use of X2 as the base Cyrillic encoding for a variety of  $\TeX$  formats (*Plain*, *AMS- $\TeX$* , *BLUET $\TeX$* ,  $\LaTeX 2.09$ , etc.) as well as  $\LaTeX 2_\varepsilon$ . (This local encoding is called LR1 below. The design aim for LR1 was to select glyphs required by the most widely-used languages and to put them into the 128–255 section of X2.)

Unfortunately the full set of glyphs including accented letters is too big to fit into 256 slots,

especially taking into account the  $\backslash\lccode$ – $\backslashuccode$  restrictions. So it is necessary to accept some principles of selection which enable us to decrease the number of Cyrillic glyphs included in X2:

1. The X2 encoding follows the  $\LaTeX 2_\varepsilon$  agreements about  $\backslash\lccode$ – $\backslashuccode$  not to produce garbage for the headings, table of contents, hyphenations inside paragraphs, arguments of  $\backslashuppercase$  and  $\backslashlowercase$ ;
2. All glyphs used in publishing for some language are included in X2 if they cannot be constructed as accented letters or letters with additional modifiers using  $\TeX$  commands. Variant glyphs for Cyrillic alphabets are also included in X2 if there is some free space and if different languages use different variants;
3. The X2 encoding includes all punctuation symbols, digits, mathematical symbols, accents, hyphens, dashes, etc., needed to form the full set of symbols necessary for Cyrillic typography;
4. The additional Cyrillic letters which are used in the PC 866 and MS Windows 1251 code pages are included in X2 even if they are accented forms;
5. Glyphs which are not used now but which were used at some stage in the 20<sup>th</sup> century may be included if there are good reasons to do so (as, for example, with the old Russian and Bulgarian letters);
6. Glyphs which were used in old Cyrillic texts before 1900 (Old Slavonic, Church Slavonic, Glagolitic, old phonetic symbols, etc.) should be moved to a separate glyph container. There could also be an additional glyph container to collect the exotic glyphs used in some contemporary Cyrillic texts;
7. When jettisoning accented letters it is necessary to take into account that they may be necessary for hyphenation patterns for some languages (if such patterns have been created or if there is a chance that they will be created sometime). For example, accented letters for Russian, Ukrainian and Belorussian, Kazakh, Tatar, and Bashkir are included in X2;
8. When deciding whether to jettison an accented letter that is used in a language supported by LR1, one must keep in mind that only the CM accents are available in that encoding;
9. The following priorities are used when the accented letters or letters with simple modifiers are thrown away: (0) letters which are easily constructed by the internal command  $\backslashaccent$  (so that the letters using accents available in

CM fonts have lower significance); (1) letters which contain a centered diacritic below the letter (cedilla, ogonek, dot, macron) and are easily constructed using a command similar to `\c` in *Plain T<sub>E</sub>X*; (2) letters which contain a horizontal stroke positioned symmetrically; (3) letters which require special alignment of accents and modifiers;

10. Accents and modifiers used in Cyrillic are included in X2 even if all accented forms are included in X2 for some other reasons (an example is *Cyrillic breve* used for **Ў** and **Ѳ**);
11. Latin letters or glyphs which are similar to some Latin letter (used in Macedonian, Kurdish, etc.) are placed at the same positions as the Latin letters are in ASCII. Among other things, this increases the number of languages supported by the LR1 encoding;
12. Whenever it is possible, glyphs (ASCII, accents, special symbols, etc.) are placed at the same positions as they are placed in T1 encoding.

The X2 encoding is shown in Fig. 1. The Russian letters **А–Я**, **а–я** (except **Ё** and **ё**) are placed in the only region in the encoding table where 32 consecutive letter positions are available — i.e., positions "c0–"df and "e0–"ff. The Russian letters **Ё** and **ё** are placed at the end of the block "80–"9c and "a0–"bc which simplifies the ordering of non-Russian letters. Latin letters and letters similar to Russian letters are placed as in ASCII. Letters used in other Cyrillic alphabets are grouped into the parts "80–"ff and "00–"7f of the encoding table according to the “popularity” of the corresponding languages (to satisfy the requirements of the LR1 encoding). They are placed in free positions reserved by L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> for letters in some quasi-alphabetic order. The old Russian and Bulgarian letters are placed at the end of the block of letters in "00–"7f.

Accents and modifiers are placed in X2 at "00–"1f; those also used by T1 are placed at the same positions as in T1. The same is true for additional symbols produced by the ligatures --, ’’, etc. The punctuation symbols, digits, mathematical symbols, etc., are placed as they are positioned in ASCII. A special case is made of the symbols № □ § „ « » which are essential for Russian typography. These symbols are placed in "80–"ff at the positions reserved for symbols, to guarantee the correctness of the LR1 encoding.

Some accents (macron, dot) can be used as lower accents as well for transliteration systems. In some specific cases the upper comma ("1b) and

lower comma are also used as accents. The lower accents will be constructed using T<sub>E</sub>X commands from the upper accents available in X2.

The accents `^` ("12) and `~` ("13) are used as stresses in Serbian; there is no letter in any Cyrillic language where these symbols are used as “normal” accents.

The quasi-letters `'` (apostrophe, "27), `”` (double apostrophe, "22) and **I** (palochka, "0d) are used like letters in some languages but do not have uppercase and lowercase forms (i.e., for these letters the uppercase form is just the same as the lowercase form).

Single quotes are not used in Cyrillic writing, and for this reason there is no need to keep single French quotes. Instead, in their place, the angle brackets `<` ("0e) and `>` ("0f) are provided. Angle brackets *are* used in Cyrillic typography, and it is good if their style is changed in parallel with the style of other symbols.

The Cyrillic breve `“` ("14) is a very famous glyph (it is even included in the Adobe and Word-Perfect Cyrillic fonts). Although all letters with this accent (**Ў/ў**, **Ѳ/ѳ**) are included in X2, it is included as a special glyph as well.

Cedilla `,”` ("0b) and ogonek `“,”` ("0c) are used by some letters already included in X2 (**З**, **Ѓ**, **Є**). These letters have variant forms where *cedilla* could be oriented to the left or to the right depending on the user’s taste. Also, some applications use *ogonek* instead of *descender* for **K**, **X**, **Г**, **Т**, etc. The availability of *cedilla* and *ogonek* in X2 makes it possible to satisfy these needs.

Percentage zero `“o”` ("18) is included as a useful idea borrowed from the T1 encoding and EC fonts: this symbol is used to convert `‘%’` into `‘%o’` and `‘%oo’`.

Punctuation ligatures, i.e., the symbols produced by the abbreviations -- (endash, "15), --- (emdash, "16), ‘ ‘ (opening English quotes, "10), ’ ’ (closing English quotes, "11) are used in the same manner and are placed at the same position as in T1, as is - (the hyphen used for hanging Hyphenation, "7f). It is worth noting that the ligature --- (emdash, "16) corresponds to *Cyrillic emdash* which (following the traditions of Russian typography) is much shorter than that glyph in Latin-encoded CM and EC fonts.

There are the special cross-modifiers: horizontal `“,”` at "17, grave-diagonal `“,”` at "19 and acute-diagonal `“,”` at "1a which are used to construct from pieces the letters **Ғ/ғ**, **Х/х**, **С/с** and **Р/р** used in some minor Cyrillic languages. (These letters are included as separate glyphs in T2A/T2B/T2C, but

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
0x	˘	˙	ˆ	˜	¨	˝	˚	ˇ	0x
	˘	˙	ˆ	˜	¨	˝	˚	ˇ	
1x	“	”	ˆ	˜	¨	˝	˚	ˇ	1x
	“	”	ˆ	˜	¨	˝	˚	ˇ	
2x	˘	˙	ˆ	˜	¨	˝	˚	ˇ	2x
	˘	˙	ˆ	˜	¨	˝	˚	ˇ	
3x	0	1	2	3	4	5	6	7	3x
	8	9	:	;	<	=	>	?	
4x	@	Æ	Ђ	Ѓ	€	€	К	К	4x
	Д	І	Ј	Љ	М	Њ	О	Ѓ	
5x	Р	Q	Т	Ѕ	Ц	Ц	Ч	W	5x
	Ђ	Ж	V	[	\	]	ˆ	˚	
6x	‘	æ	ђ	ѓ	€	€	к	к	6x
	д	i	j	љ	м	њ	о	џ	
7x	р	q	т	ѕ	ц	ц	ч	w	7x
	ђ	ж	v	{		}	˜	-	
8x	Г	Г	Г	Ђ	h	Ж	З	З	8x
	Ї	К	К	Ѓ	Ј	Ѓ	Н	Ѓ	
9x	Ө	Ç	ÿ	У	У	Х	Х	Ч	9x
	Ч	€	ə	ε	ë	„	«	»	
Ax	г	г	г	ђ	h	ж	з	з	Ax
	ї	к	к	ѓ	ј	ќ	н	ќ	
Bx	ө	ç	ÿ	у	у	х	х	ч	Bx
	ч	€	ə	ε	ë	„	«	»	
Cx	А	Б	В	Г	Д	Е	Ж	З	Cx
	И	Й	К	Л	М	Н	О	П	
Dx	Р	С	Т	У	Ф	Х	Ц	Ч	Dx
	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	
Ex	а	б	в	г	д	е	ж	з	Ex
	и	й	к	л	м	н	о	п	
Fx	р	с	т	у	ф	х	ц	ч	Fx
	ш	щ	ъ	ы	ь	э	ю	я	
	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	

Figure 1: The “Cyrillic glyph container” X2

unfortunately the limit of 256 characters prevents including them in X2.)

The positions "1c"/"1d and "1e"/"1f are used for the exotic letters  $\delta/\delta$  and  $\mathbb{H}/\mathbb{H}$  used by some minor Cyrillic alphabets. Although following  $\text{\LaTeX} 2_{\epsilon}$  rules these positions should be used for *symbols*, not for *letters*, we have made them exceptions from the severe  $\text{\LaTeX} 2_{\epsilon}$  requirements. Formally speaking, the  $\text{\LaTeX} 2_{\epsilon}$  requirements are not violated since the `\lccode`–`\uccode` data for these positions conserve the original values. Instead of the explicit use of the `\lccode`–`\uccode` mechanism, the lowercase–uppercase conversion is performed by the  $\text{\LaTeX} 2_{\epsilon}$  `\MakeUppercase` and `\MakeLowercase` transformations using the list `\@uclclist` of identifiers. As a result these “letters” cannot be used in hyphenation patterns and they break the automatic hyphenation whenever they appear in a word. The gain is that even exotic Cyrillic texts could be created (if necessary) using X2 only and without additional encodings and fonts.

## 5 “True” $\text{\LaTeX} 2_{\epsilon}$ encodings T2A, T2B, T2C

The base features of the T2\* encodings are determined by the  $\text{\LaTeX} 2_{\epsilon}$  specifications for T(*n*) encodings and by the already created X2 encoding. Some more requirements are added due to the necessity to keep the fonts in the T1, X2 and T2\* encodings compatible. For example, it is necessary to keep similar glyphs at the same positions in all fonts whenever it is possible. As a result the following basic principles appear.

1. The set of T2\* encodings supports the full set of modern Cyrillic languages, each Cyrillic language is supported at least by one encoding T2\* so that there is no necessity to mix encodings for some languages;
2. Cyrillic letters occupy the positions reserved in  $\text{\LaTeX} 2_{\epsilon}$  for letters, Cyrillic symbols—positions reserved for symbols, Cyrillic accents—positions reserved for accents;
3. Letters included in T2\* follow the  $\text{\LaTeX} 2_{\epsilon}$  convention about uppercase and lowercase letters (i.e., have the same `\uccode`–`\lccode` assignments as in T1);
4. ASCII glyphs (Latin letters, digits, mathematical and punctuation symbols, etc.) are placed at "20–"7f;
5. The symbols produced by the ligatures --, ---, ‘, ’ are included at the same positions as they are in T1 and X2 (it is worth noting that, as in X2, the *emdash* glyph is the *Cyrillic*

*emdash* which is shorter than that in OT1 and T1);

6. The ff-ligatures (ff, fi, fl, ffi, ffl) are included at "1b–"1f as in T1 to keep the full set of Latin glyphs necessary for typography;
7. The standard accents and symbols, and Cyrillic-specific accents and symbols used in "00–"1a of X2, are reproduced in T2\* at the same positions except the cross-modifiers which are not necessary (the letters with cross-modifiers are included as separate glyphs);
8. The Russian alphabet is reproduced similar to X2;
9. The symbols specific to Cyrillic typography ( $\mathbb{N}$   $\propto$  § „ « ») are reproduced in "9d, "9e, "9f, "bd, "be and "bf, similar to X2;
10. Positions "80–"9b and "a0–"bb are used for national Cyrillic letters (these are the only positions that differ from the T2\* encodings since all other codes are already fixed as described above);
11. To prevent an increase in the number of encodings up to infinity, the accented letters for Cyrillic languages which do not have the hyphenation tables in  $\text{\TeX}$  format (and rarely will have in future) are not included;
12. Equivalent letters occupy just the same positions in all the T2\* encodings whenever possible (i.e., some glyphs may be absent in some encodings, but if the glyph is included in an encoding, it occupies the same position as in the other encodings).

The encodings can therefore be decomposed into the following regions:

- the accent, non-ASCII punctuation and ligature symbols ("00–"1f),
- the ASCII-encoded Latin letters, digits, punctuation and mathematical symbols, etc. ("20–"7f),
- non-letter symbols at "9d–"9f and "bd–"bf ( $\mathbb{N}$   $\propto$  § „ « »),
- specific (uppercase and lowercase) Cyrillic letters ("80–"9b, "a0–"bb),
- uppercase and lowercase Russian letters ("c0–"df, "e0–"ff, "9c, "bc).

The accent part (see Fig. 2) is copied from X2 with minor changes:

- a) the exotic letters ("1c–"1f) and the upper comma accent ("1b) are substituted by the ff-ligatures “ff”, “fi”, “fl”, “ffi”, “ffl”,

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
0x	`	´	^	~	¨	˝	°	˘	0x
	˘	-	·	ˆ	˙	I	<	>	
1x	“	”	ˆ	˝	˘	-	—		1x
	o	l	J	ff	fi	fl	ffi	fff	

a) Accents, ligatures, special symbols, etc.

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
2x	˘	!	"	#	\$	%	&	'	2x
	(	)	*	+	,	-	.	/	
3x	0	1	2	3	4	5	6	7	3x
	8	9	:	;	<	=	>	?	
4x	@	A	B	C	D	E	F	G	4x
	H	I	J	K	L	M	N	O	
5x	P	Q	R	S	T	U	V	W	5x
	X	Y	Z	[	\	]	^	_	
6x	‘	a	b	c	d	e	f	g	6x
	h	i	j	k	l	m	n	o	
7x	p	q	r	s	t	u	v	w	7x
	x	y	z	{		}	~	-	

b) ASCII glyphs

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
Cx	А	Б	В	Г	Д	Е	Ж	З	Cx
	И	Й	К	Л	М	Н	О	П	
Dx	Р	С	Т	У	Ф	Х	Ц	Ч	Dx
	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	
Ex	а	б	в	г	д	е	ж	з	Ex
	и	й	к	л	м	н	о	п	
Fx	р	с	т	у	ф	х	ц	ч	Fx
	ш	щ	ъ	ы	ь	э	ю	я	

c) Russian letters

**Figure 2:** Common parts in T2A/T2B/T2C

- b) the cross-modifiers are substituted by the *compound-word-mark* symbol<sup>3</sup> ("17), the *dotless-i* ("19) and the *dotless-j* ("1a) like it is in T1.

The ASCII-encoded Latin letters, digits, punctuation and mathematical symbols at "20–"7f are placed exactly as in the T1 encoding as shown in Fig. 2.

The Russian alphabet letters and non-letter symbols at "9d–"9f, "bd–"bf are just copied from the corresponding positions in the X2 encoding (see Fig. 2).

The component which is different in T2A/T2B/T2C/ are the national letters at "80–"9b and "a0–"bb, which are shown in Fig. 3 and Table 1. Although we tried to fulfill the requirement that the equivalent letters should occupy the same positions in all encodings, it was impossible to completely fulfill this requirement. Fortunately there are only two exceptions: the letters Ъ/ъ and Ѓ/ѓ are placed at "87/"a7 and "9b/"bb in T2A, but at "88/"a8 and "99/"b9 in T2B. All other letters and symbols in T2A, T2B and T2C (and in the accent part "00–"1a, symbol/digit part "20–"3f, "5b–"5f, "7b–"7f and lower part "80–"ff of X2) have fixed positions.

A summary of the languages covered by T2A/T2B/T2C is shown below. The encoding T2A contains the leading languages sorted by using statistical data on populations. The encoding T2B contains the majority of the remaining languages. Finally, the encoding T2C contains several languages with exotic letters which do not fit into T2A or T2B: Abkhazian, Orok (Uilta), Saam (Lappish), Old-Bulgarian, Old-Russian. Due to the intersections between Cyrillic alphabets some languages are supported by two or three encodings simultaneously:

**T2A:** Abaza, Avar, Agul, Adyghei, Azerbaidzan, Altai, Balkar, Bashkir, Belorussian, Bulgarian, Buryat, Gagauz, Dargin, Dungan, Ingush, Kabardino-Cherkess, Kazah, Kalmyk, Karakalpak, Karachaevsii, Karelian, Kirgiz, Komi-Zyrian, Komi-Permyak, Kumyk, Lak, Lezgin, Macedonian, Mari-Mountain, Mari-Valley, Moldavian, Mongolian, Mordvin-Moksha, Mordvin-Erzya, Nogai, Oroch, Osetin, Russian, Rutul, Serbian, Tabasaran, Tadjik, Tatar, Tati, Teleut, Tofalar, Tuva, Turkmen, Udmurt,

Uzbek, Ukrainian, Hanty-Obksii, Hanty-Surgut, Gipsi, Chechen, Chuvash, Crimean-Tatar;

**T2B:** Abaza, Avar, Agul, Adyghei, Aleut, Altai, Balkar, Belorussian, Bulgarian, Buryat, Gagauz, Dargin, Dolgan, Dungan, Ingush, Itelmen, Kabardino-Cherkess, Kalmyk, Karakalpak, Karachaevsii, Karelian, Ketskii, Kirgiz, Komi-Zyrian, Komi-Permyak, Koryak, Kumyk, Kurdian, Lak, Lezgin, Mansi, Mari-Valley, Moldavian, Mongolian, Mordvin-Moksha, Mordvin-Erzya, Nanai, Nganasan, Negidal, Nenets, Nivh, Nogai, Oroch, Russian, Rutul, Selkup, Tabasaran, Tadjik, Tatar, Tati, Teleut, Tofalar, Tuva, Turkmen, Udyghei, Uigur, Ulch, Khakass, Hanty-Vahovskii, Hanty-Kazymskii, Hanty-Obksii, Hanty-Surgut, Hanty-Shurysharskii, Gipsi, Chechen, Chukcha, Shor, Evenk, Even, Enets, Eskimo, Yukagir, Crimean Tatar, Yakut;

**T2C:** Abkhazian, Bulgarian, Gagauz, Karelian, Komi-Zyrian, Komi-Permyak, Kumyk, Mansi, Moldavian, Mordvin-Moksha, Mordvin-Erzya, Nanai, Orok (Uilta), Negidal, Nogai, Oroch, Russian, Saam, Old-Bulgarian, Old-Russian, Tati, Teleut, Hanty-Obksii, Hanty-Surgut, Evenk, Crimean Tatar.

## 6 The Cyrillic glyph container X2 versus Cyrillic encodings T2A, T2B, T2C

As was already specified, there are two main requirements essential to the reliable working of  $\text{\LaTeX} 2_{\epsilon}$  in multi-language mode:

- we must keep the `\lccode–\uccode` table as in T1,
- we must keep the ASCII encoding for positions 32–127.

Both requirements are fulfilled by the T2A/T2B/T2C encodings and hence they can be safely mixed with the Latin encodings OT1 and T1 inside a document. The encoding X2 conserves the `\lccode–\uccode` values but does not contain these ASCII glyphs. As a result it *may* cause problems and unexpected effects inside  $\text{\LaTeX} 2_{\epsilon}$  documents if the user is not careful enough. So, why do we need X2 when we have T2A/T2B/T2C?

The reason is that the requirement to keep all the ASCII glyphs is very restrictive—it leaves only 61 positions for non-ASCII letters.<sup>4</sup> To fit all Cyrillic letters into T<n> encodings requires *three* tables

<sup>3</sup> The empty character with the zero thickness and the height equal to *1ex* used in EC fonts and T1 encoding for special applications—such as hyphenating compound words, breaking down ligatures, creating accents to be placed over the invisible space between two letters.

<sup>4</sup> For Cyrillic encodings it is even more restrictive: it is necessary to keep 32 base Russian letters in each encoding as well since they are encountered in almost all Cyrillic alphabets.

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
8x	Г	Ғ	Ђ	Ң	Һ	Ж	З	Љ	8x
	Ї	Қ	К	К	Æ	Ң	Ң	Ѕ	
9x	Ө	Ҙ	Ү	Ү	Ү	Х	Ц	Ч	9x
	Ҙ	€	Ә	Ң	Ё	№	⊘	§	
Ax	г	ғ	ђ	ң	һ	ж	з	љ	Ax
	ї	қ	к	к	æ	ң	ң	ѕ	
Bx	ө	ҙ	ү	ү	ү	х	ц	ч	Bx
	Ҙ	€	ә	Ң	ё	„	«	»	

a) T2A encoding

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
8x	Ғ	Ғ	Г	Ђ	Һ	Ж	δ	З	8x
	Љ	Қ	Л	Ѓ	Ј	Ң	Ң	Ң	
9x	Ө	С	Ү	Ү	Х	Х	Х	Ч	9x
	Ҙ	Ң	Ә	€	Ё	№	⊘	§	
Ax	ғ	ғ	г	ђ	һ	ж	δ	з	Ax
	љ	қ	л	ѓ	ј	ң	ң	ң	
Bx	ө	с	ү	ү	х	х	х	ч	Bx
	Ҙ	Ң	ә	€	ё	„	«	»	

b) T2B encoding

	x0/x8	x1/x9	x2/xA	x3/xB	x4/xC	x5/xD	x6/xE	x7/xF	
8x	Ң	Ц	Т	Ђ	Һ	Р	Р	З	8x
	М	Қ	Л	К	Ј	Ң	М	Ң	
9x	Ө	€	€	Ђ	Ү	Х	Ц	⊘	9x
	Ҙ	Ң	Ә	Ж	Ё	№	⊘	§	
Ax	Ң	ц	т	ђ	һ	р	р	з	Ax
	м	қ	л	к	ј	ң	м	ң	
Bx	ө	€	€	ђ	ү	х	ц	⊘	Bx
	Ҙ	Ң	ә	ж	ё	„	«	»	

c) T2C encoding

**Figure 3:** The national Cyrillic letters in T2A/T2B/T2C

Code	T2A	T2B	T2C
"80/"a0	ghe-upturned	ghe-with-descender hcrossed	pe-with-tail
"81/"a1	ghe-hcrossed	ghe-hcrossed	te+tse
"82/"a2	dje (T+h with tail)	ghe-with-descender	te-with-descender
"83/"a3	tshe (T+h)	ghe-with-tail	ghe-with-tail
"84/"a4	h-special	h-special	h-special
"85/"a5	zhe-with-descender	zhe-with-descender	er-with-descender
"86/"a6	ze-with-descender	delta-phonetical	er-gravecrossed
"87/"a7	lje	zet	zet
"88/"a8	i-with-umlaut	lje	em-with-descender
"89/"a9	ka-with-descender	ka-with-descender	ka-with-descender
"8a/"aa	ka-with-left-poker	el-with-descender	el-with-descender
"8b/"ab	ka-vcrossed	ka-with-tail	ka-hcrossed
"8c/"ac	a+e	el-with-tail	el-with-tail
"8d/"ad	en-with-descender	en-with-descender	en-with-descender
"8e/"ae	en+ghe	en+ghe	em-with-tail
"8f/"af	S	en-with-tail	en-with-tail
"90/"b0	o-barred	o-barred	o-barred (fita)
"91/"b1	es-with-descender	es-acutecrosseds	abkhazian-ch
"92/"b2	u-with-cyrbreve	u-with-cyrbreve	abkhazian-ch with descender
"93/"b3	Y-special	Y-special	yat (semisoft sign)
"94/"b4	Y-special-hcrossed	ha-hcrossed	izhitsa
"95/"b5	ha-with-descender	ha-with-descender	ha-with-descender
"96/"b6	tse-macedonian	ha-with-tail	tse-macedonian
"97/"b7	che-vcrossed	che-with-left-descender	abkhazian-ha
"98/"b8	che-with-descender	che-with-descender	che-with-descender
"99/"b9	e-ukrainian	nje	en-with-right-tail
"9a/"ba	shwa	shwa	shwa
"9b/"bb	nje	epsilon	big yus

Table 1: The national Cyrillic letters in encodings T2A/T2B/T2C

to achieve the coverage of X2: the problem is that most characters in the  $T\langle n \rangle$  tables are the same. And to support each such  $T\langle n \rangle$  encoding it is necessary to have a separate font class like the EC fonts.

To keep such enormous numbers of fonts is too large a price for people who use Cyrillics only occasionally. On the other hand, if all the Cyrillic glyphs are put into just one table without the Latin letters in 32–127, but in a way that satisfies the  $\backslash\code\code$  requirements, one table and one font class is enough provided the user obeys some elementary rules of safety. This “economy mode” is implemented by X2.

There is a similar situation for Old Slavonic characters and some other encodings which are only occasionally used by normal users. To resolve this problem, “glyph containers” like X2 could again be helpful. The “glyph container” encodings  $X\langle n \rangle$  should be an intermediate case between  $T\langle n \rangle$  and the “free style”  $X\langle n \rangle$ : such encodings do not have

ASCII in 32–127, but they do have the standard, compatible  $\backslash\code\code$  values.

Currently the L<sup>A</sup>T<sub>E</sub>X Team only supports the  $T\langle n \rangle$  encodings and TS encodings, while the support of an  $X\langle n \rangle$  encoding is entirely the responsibility of the designer of the encoding.<sup>5</sup> For the “glyph container” X2 such support should include listing some simple rules which should be followed by Users so as to avoid strange and undesirable effects. The X2 encoding does not contain Latin ASCII letters but only digits, punctuation and mathematical symbols, etc., therefore the rules should guarantee that text containing Latin ASCII letters is never used with the X2 encoding:

<sup>5</sup> It seems that there *should be* support for such “glyph container” encodings by the L<sup>A</sup>T<sub>E</sub>X Team as well (such support should include the registration procedure for glyph containers and maintenance of the official list of exceptions where the glyph container encodings produce undesirable results).



1. If you use ASCII Latin letters in the text part of your document, some Latin encoding (i.e., OT1 or T1) must be active for this piece of your text;
2. The following commands should be used only outside the range where the X2 encoding is active (or should be preceded by the explicit specification of some “Latin” encoding) because they may implicitly include the Latin text in your document:

```
\part, \chapter, \section,
\subsection, \subsubsection,
\paragraph, \subparagraph, \caption,
\tableofcontents, \listoftables,
\listoffigures, \ref, \pageref, \cite,
\item, \labelenumxx, \labelitemxx,
\makelabel, \numberline, \thechapter,
\thesection, \thesubsection, \date,
\today.
```

Exactly the same requirement is needed for all user-defined macros (or those loaded from external packages) that have ASCII Latin text in their body but without explicit specification of the “Latin” encodings OT1 or T1 for this text;

3. When you deal with floating objects (or moving arguments), you should not rely on the assumption that the Cyrillic letters are used by L<sup>A</sup>T<sub>E</sub>X when the floating material is inserted into the document. For example, if Cyrillic letters are used inside some command defining the floating object, the encoding X2 should be activated explicitly in front of Cyrillic letters even if X2 is active at the point where the command is issued. Among such commands are:

- (a) the floating environments:

```
\begin{table}–\end{table},
\begin{figure}–\end{figure},
\begin{table*}–\end{table*},
\begin{figure*}–\end{figure*},
```

- (b) the commands that define floating text explicitly:

```
\author, \title, \date, \address,
\name, \signature, \telephone,
\footnote, \footnotetext, \thanks,
\marginpar, \markboth, \markright,
\bibitem, \topfigrule, \botfigrule,
\dblfigrule, \footnoterule,
```

- (c) the commands that define headers, footers, margin remarks, etc., implicitly:

```
\part, \chapter, \section,
\subsection, \subsubsection,
\paragraph, \subparagraph, \caption,
```

- (d) the commands that write, explicitly or implicitly, text to external files, which may be loaded outside the X2 encoding:

```
\addtocontents, \addcontentsline,
\glossary, \index, \part,
\chapter, \section, \subsection,
\subsubsection, \paragraph,
\subparagraph, \caption.
```

Similarly, if such a floating command includes Latin letters and the resulting object may appear inside the range where X2 is active, some Latin encoding (i.e., OT1 or T1) should be activated explicitly before the Latin-encoded text.

4. Just the same requirement holds for *all* commands which can occasionally insert Cyrillic or Latin text where the Latin encodings OT1/T1 or the “Cyrillic glyph container” encoding X2 are active. For example, you should be careful with the definition and re-definition of the following commands:

- (a) commands which create automatically generated text used by other commands:

```
\labelenumxx, \labelitemxx,
\makelabel, \numberline, \thechapter,
\thesection, \thesubsection, \today,
```

- (b) commands which are used in international L<sup>A</sup>T<sub>E</sub>X to define language-specific names:

```
\abstractname, \appendixname,
\alsoname, \ccname, \chaptername,
\contentsname, \enclname,
\headtoname, \figurename, \indexname,
\listfigurename, \listtablename,
\notesname, \pagename, \partname,
\prefacename, \seename, \tablename,
```

- (c) commands which implicitly define headers, footers, margin remarks, etc., and/or implicitly write something into external files:

```
\part, \chapter, \section,
\subsection, \subsubsection,
\paragraph, \subparagraph, \caption,
\addtocontents, \addcontentsline,
\glossary, \index,
```

- (d) commands which create floating text and floating environments:

```
\author, \title, \date, \address,
\name, \signature, \telephone,
\footnote, \footnotetext, \thanks,
\marginpar, \markboth, \markright,
\bibitem, \topfigrule, \botfigrule,
\dblfigrule, \footnoterule,
\begin{table}–\end{table},
\begin{figure}–\end{figure},
```

- (e) macros and user-defined commands which may be expanded unintentionally inside or outside X2:

```
\begin{table*}-\end{table*},
\begin{figure*}-\end{figure*},
\def, \newcommand, \newcommand*,
\renewcommand, \renewcommand*,
\providecommand, \providecommand*,
\newenvironment, \newenvironment*,
\renewenvironment,
\renewenvironment*,
\newtheorem, \ProvideTextCommand,
\ProvideTextCommandDefault,
\AtBeginDocument, \AtEndDocument,
\AtEndClass, \AtEndPackage,
\DeclareRobustCommand,
\DeclareTextCommand,
\DeclareTextCommandDefault.
```

## 7 The weak points of X2 and T2\*

The X2 and T2\* encodings do not contain accented letters, and (for some languages) this throws the user back on the `\accent` primitive which prevents construction of correct hyphenation tables and destroys kerning pairs. The encodings (especially X2) are also overloaded (to some extent) with rare glyphs, which arise from the attempt to collect *all* Cyrillic glyphs in one table.

There are the cross-modifiers (horizontal stroke “.”, vertical stroke “|”, diagonal strokes “/” and “\”) which are included in X2 but are absent in T2A/T2B/T2C. Although there is a great chance that these glyphs will be included in TS2 (see section 8), their status at this stage of the project is undefined. Similarly, there are the title forms<sup>6</sup> for the letters Ъ/ъ and Ѓ/ѓ which were included in previous (intermediate) versions of X2 but are now excluded for some reason.

Another disadvantage of minor importance is that there are two glyphs (Ѣ/ѣ and Ѧ/ѧ) which correspond to logically different letters: Ѣ/ѣ stands for Saam *semisoft sign* and for old Russian *yat*, and Ѧ/ѧ stands for *o-barred* and old Russian *fita*. Although graphically these symbols are similar, they are different logically.

This situation can be accepted taking into account the status of X2 as a glyph table rather than

<sup>6</sup> The title form is a combination of the uppercase “J” or “H” and the bowl from the *lowercase* “b”. These glyphs are used for first letters in titles, etc., where the first letter is capital and other letters are in lowercase mode. For example, there is the title form “Ij” for the ligature “IJ”/“ij” used in Dutch. (Note, that the title form “Ij” is absent in T1 and TS1 encodings.)

a table for direct text coding, and the status of T2\* as the *modern Cyrillic* encodings. In structured markup, the ambiguity would be addressed by assigning *two* symbolic names for each glyph (say, `\yat/\semisft` and `\fita/\obarred`) and only using the semantically correct one to code texts.

Some preliminary information about exotic glyphs and pure phonetic symbols has been provided by linguists studying some minor writing systems. These letters and symbols are not currently included in X2 and T2\* at all. The reason for not including the glyphs at this stage is that the writing systems are very unstable and are subject to change from publication to publication. There is no justification for including such symbols in the version of X2 and T2\* proposed as a *standard* until the situation becomes stable.

It seems that all the specific Cyrillic glyphs used in modern Cyrillic alphabets are included in X2 and in one of the T2\*, but there is also a chance that some minor writing system is omitted. There is also a chance that some linguists suggest a new alphabet for some minor language using their own glyphs not available in X2. Until this happens we can consider X2 and T2A/T2B/T2C as comprehensive glyph collections for modern Cyrillic texts (although not very comfortable and not specifically adjusted for intensive Cyrillic writing).

## 8 Some remarks about the TS2 encoding

The TS2 is expected to be the collection of accents and special symbols which are necessary for Cyrillic typography, but which are not included into the encodings X2 and T2A/T2B/T2C for some reasons (i.e., TS2 is the encoding supplementary to X2 and T2<n> as TS1 is supplementary to T1).

For typographical reasons, ‘wide’ versions of some accents—macron, tilde, breve, etc.—are desirable. These versions would be used for extra wide letters: as compared with the Latin alphabet, Cyrillic has a far higher proportion of wide letters. Such wide versions of the accents are good candidates for a TS2 encoding. Similarly, the lowercase/uppercase variants of cedilla, ogonek and the accents absent in T1 and TS1 may make useful additions to TS2.

The letters Ъ/ъ and Ѓ/ѓ used in some Cyrillic languages are actually ligatures “J+B” and “H+B”. As well as the uppercase and lowercase forms there is also a *title* form for these letters: the combination of the uppercase form for “J” or “H” and the bowl for the lowercase “b”. This form is used for titles where the first letter is capital while the other letters are ordinary (a similar effect occurs for ‘IJ’ used in Dutch). Such title letters

should be placed in TS2 and shared by the X2 and T2\* encodings.

To construct some exotic letters from pieces, special modifiers are necessary: horizontal stroke “.”, vertical stroke “|”, diagonal strokes “\” and “/”. The diagonal strokes are used only for letters  $\mathcal{C}/\mathcal{c}$  (Enetz) and  $\mathcal{F}/\mathcal{f}$  (Saam, or Lappish). Vertical strokes are used only for letters  $\mathcal{K}/\mathcal{k}$  and  $\mathcal{V}/\mathcal{v}$  which have become obsolete since modern Azerbaijan writing is based on the Latin alphabet. Horizontal strokes are used in several Cyrillic letters ( $\mathcal{F}/\mathcal{f}$ ,  $\mathcal{K}/\mathcal{k}$ ,  $\mathcal{Y}/\mathcal{y}$ , etc.). There are serious reasons for keeping these modifiers in TS2: there are still minor languages for which alphabets based on Cyrillic could be proposed. The availability of these modifiers in TS2 would support such developments without the necessity to include more glyphs in the X2 and T2\* encodings.

It is still a question how, and whether, the TS2 encoding should be realized. Taking into account that there are only a few glyphs really necessary for it and that there are several positions in TS1 reserved for future extension of this encoding, it may be a good decision just to combine these two encodings.

## 9 Acknowledgments

There are many people who have contributed to this project, and it is difficult to list all of them in this section. Among the people who contributed the essential components are Mikhail Grinchuk, Vladimir Volovich, Werner Lemberg, Frank Mittelbach, Jörg Knappen, Michel Goossens, Andrew Slepudin, but the list is not restricted to these names only.

We are especially thankful to Vladimir Volovich and Werner Lemberg for their work on macro support for the X2 and T2\* encodings and to the members of the mailing list *CyrTEX-T2* who discussed enthusiastically the X2 and T2\* problems. (To subscribe to this list send email to [Majordomo@vsv.vsu.ru](mailto:Majordomo@vsv.vsu.ru) with the command: `subscribe cyrtex-t2 your-e-mail-address.`)

We are grateful to Robin Fairbairns for his time spent polishing the text of our papers submitted to the *EuroTEX-98* conference where the preliminary results of this project (namely, the X2 encoding) was presented for the first time, and to Michel Goossens for his efforts to organize the support for Russian participants at *EuroTEX-98*.

Finally, although most work on this project was done on a voluntary basis (as it is traditional for T<sub>E</sub>X community), it is worth mentioning that part of the research was supported by a grant from the

Dutch Organization for Scientific Research (NWO grant No 07–30–007).

## References

- [1] A. Berdnikov, O. Lapko, M. Kolodin, A. Janishevsky, A. Burykin. The encoding paradigm in  $\LaTeX 2_{\epsilon}$  and the projected X2 encoding for Cyrillic texts. Proceedings of *EuroTEX-98*, Saint-Malo, 1998.
- [2] A. Berdnikov, O. Lapko, M. Kolodin, A. Janishevsky, A. Burykin. Alphabets necessary for various Cyrillic writing systems (towards X2 and T2 encodings). Proceedings of *EuroTEX-98*, Saint-Malo, 1998.
- [3] A. Berdnikov, O. Grineva. Some problems with accents in T<sub>E</sub>X: letters with multiple accents and accents varying for uppercase/lowercase letters. Proceedings of *EuroTEX-98*, Saint-Malo, 1998.
- [4] K. Píška, *Cyrillic Alphabets*, in: Proceedings of TUG'96, eds. M. Burbank and C. Thiele, pp. 1–7, JINR, Dubna; *TUGboat* 17(2), pp. 92–98.
- [5] O. Lapko, *Full Cyrillic: How many languages*, in: Proceedings of TUG '96, eds. M. Burbank and C. Thiele, pp. 164–170, JINR, Dubna; *TUGboat* 17(2), pp. 174–180.
- [6] K. Kenneth. The languages of the world. London, Henley.
- [7] M. Ruhlen. A Guide to the languages of the world. Stanford University, 1975.
- [8] C. F. Voegelin, F. M. Voegelin. Classification and Index of the World Languages. Academic Press, 1977.
- [9] WWW page by Karel Píška: <http://www-hep.fzu.cz/~piska/>
- [10] Ethnologue Database: <ftp://ftp.std.com/obi/Ethnologue/eth.Z>

◇ A. Berdnikov, O. Lapko,  
M. Kolodin, A. Janishevsky,  
A. Burykin  
Institute of Analytical  
Instrumentation  
Rizskii pr. 26, 198103  
St. Petersburg, Russia  
[berd@ianin.spb.su](mailto:berd@ianin.spb.su)

---

**Romanized Indic and L<sup>A</sup>T<sub>E</sub>X**

Anshuman Pandey

**1 Introduction**

In 1990 at the 8th World Sanskrit Conference in Vienna, a panel of Indologists devised two encoding schemes which would enable them to exchange electronic data across a variety of platforms. These schemes are the “Classical Sanskrit” and “Classical Sanskrit eXtended” encodings, widely known in Indological circles as CS and CSX, respectively, or simply, CS/CSX.

**2 CS and CSX**

The CS and CSX encodings are currently the closest thing to an accepted standardization of the 8-bit transliteration of Indic scripts. CS/CSX is based on IBM Code Page 437, whose characters of the range 129–255 have been reassigned with characters traditionally used for the romanization of Sanskrit.

The accented French and German characters in the cp437 range 129–223 were not altered, in order to facilitate the input of these languages as well as English and Sanskrit. The accented characters required for Sanskrit were located as far as possible in the positions used by cp437 for graphic or mathematical symbols.

The re-encoding of cp437 was discussed in a document by Dominik Wujastyk titled *Standardization of Sanskrit for Electronic Data and Screen Representation* [1].

CS is a basic inventory of diacritic letters comprising the following characters traditionally used for the transliteration of Classical Sanskrit:

ī	ṁ	ā	Ā	ī	Ī	ū	Ū	ṛ
Ṛ	ṝ	Ṝ	ḷ	Ḷ	ḹ	ṅ	Ṅ	
ṭ	Ṭ	ḍ	Ḍ	ṇ	Ṇ	ś	Ś	ṣ
ṡ	Ṣ	ḥ	Ḥ					

CSX is an extension of the above which provides the following additional characters used in Vedic Sanskrit and in Prakrit:

ṝ	ṝ̄	ī̄	ū̄	ṅ̄	ā̄	ā̄	í	ì
ú̄	ù̄	ṝ̄	ṝ̄̄	ṝ̄̄	ā̄̄	ī̄̄	ū̄̄	ē̄
ō̄	ě̄	ȫ	ḹ					

Contrary to what the name indicates CS/CSX is not limited to the transliteration of Sanskrit, and may be used to transliterate many other Indic scripts effectively.

**3 ISO 15919 and CSX+**

ISO/TC46/SC2/WG12, the International Standards Organization Working Group for the Transliteration of Indic, has been busy with the draft ISO 15919 standard [2]. This draft standard provides tables which enable the romanization of Indic scripts which are specified in Rows 09–0D and 0F of UCS (ISO/IEC 10646 and Unicode).

This romanization is accomplished using plain ASCII 7-bit (ISO-646) characters, two or three roman characters often being required to represent a single Indic one. These tables provide for the Devanagari, Gujarati, Gurmukhi, Bengali (including Assamese), Oriya, Telugu, Kannada, Malayalam, Tamil, and Sinhala scripts. This draft is not yet a standard, although work is well advanced.

While ISO 15919 is still in draft stages, it appears that a consensus has been reached with regard to the form of transliteration. It was therefore decided that CS/CSX ought to be further extended to account for the new characters proposed in the draft standard. John Smith, Dominik Wujastyk, and I developed an extension to CS/CSX known as CSX+ (Classical Sanskrit eXtended+).

CSX+ aims to be downward compatible with CS/CSX, save for the relocation of two characters in positions used for non-breaking spaces in popular software packages. While seeking to implement the draft ISO 15919 standard, CSX+ also retains a useful set of European accented characters, dashes, and quotes.

Most of the new characters are those required for the draft ISO 15919 standard, which specifies the following characters which are unsupported by CS/CSX:

æ	Æ	ǔ	ṝ	Ṝ	ṝ̄	ṝ̄̄	ṝ̄̄̄	ṝ̄̄̄̄
ḹ	Ḹ	ē̄	Ē̄	ē̄̄	ō̄	Ō̄	ō̄̄	ý̄
ř̄	ṡ̄	ň̄	ṡ̄̄	ṡ̄̄̄	ḵ̄	Ḹ̄	Ḹ̄̄	ḡ̄
Ġ̄	č̄	Č̄	ḥ̄	ḥ̄̄				

In addition to the above, the following further characters have been added as being centrally useful in any text encoding:

“ ” – —

There is a single “European” accented character —  $\ddot{y}$  — that CSX inherited from the original Code Page 437, but that is unlikely to be required for any Indian or European language. It has been eliminated to save one character slot. Other characters removed from the code page are the currency symbols sterling, yen, and cent, and the guillemets.

128	Ç	147	ô	165	Ñ	184	ì	203	Æ	222	—	241	‡
129	ü	148	ö	166	Ï	185	ē	204	kh	223	“	242	Ṭ
130	é	149	ò	167	ṁ	186	ō	205	ḡ	224	ā	243	ḍ
131	â	150	û	168	ā	187	Ṛ	206	ĉ	225	ß	244	Ḍ
132	ä	151	ù	169	ĩ	188	ṙ	207	ř	226	Ā	245	ṇ
133	à	152	æ	170	ũ	189	ú	208	ā	227	ī	246	Ṇ
134	å	153	Ö	171	ã	190	ù	209	ĩ	228	Ī	247	ś
135	ç	154	Ü	172	ĩ	191	ř	210	ũ	229	ū	248	Ś
136	ê	155	ÿ	173	ṅ	192	õ	211	ẽ	230	Ū	249	ṣ
137	ë	156	ē	174	ṛ	193	ṁ	212	õ	231	ṛ	250	Ṣ
138	è	157	ṛ	175	ḷ	194	ṭ	213	ě	232	Ṛ	251	”
139	ï	158	á	176	ḹ	195	Ē	214	ö	233	ṛ	252	ṡ
140	î	159	ṛ	177	ṙ	196	Ō	215	ḷ	234	Ṛ	253	Ṣ
141	ì	160	space	178	ṛ	197	ñ	216	ũ	235	ḷ	254	ḥ
142	Ä	161	í	179	ṙ	198	ṛ	217	Ġ	236	ḷ	255	Ḥ
143	Å	162	ó	180	ṁ	199	ṛ	218	Ĉ	237	ḷ		
144	É	163	ú	181	á	200	Kh	219	h	238	ḷ		
145	æ	164	ñ	182	à	201	k	220	h	239	ñ		
146	Æ			183	í	202	space	221	—	240	Ñ		

Table 1: CSX+ Character Encoding Table

The remaining assignments have been made on the basis that the best use for the small number of spare slots available is to use them for capitalised versions of those new characters with the most need for capital forms—i.e., characters capable of beginning a word.

#### 4 Input encoding

As the use of  $\text{\LaTeX}$  amongst Indologists has significantly increased, I felt that the CSX+ encoding scheme ought to be adapted for use with  $\text{\LaTeX}$  through the `inputenc` package. To serve this end an input encoding definition file called `cp437csx.def` has been developed and placed on CTAN in the directory `fonts/csx/styles/`. Such an input encoding definition will make the typesetting of romanized Indic much easier, and might lead to the development of hyphenation patterns for romanized Sanskrit and other Indic languages.

The file `cp437csx.def` enables text encoded in CSX+ to be read and accurately typeset by  $\text{\LaTeX}$  without the need for converting the CSX+ text into  $\text{\LaTeX}$  accent codes. Table 1 provides a map of the CSX+ encoding character set. A screen font and driver for displaying CSX+ text on MS-DOS and OS/2 systems is also available in the directory `fonts/csx/`.

The stabilization of the CSX+ encoding, in tandem with the emerging ISO standard, will encourage

further necessary work, such as hyphenation tables for romanized Indic.

#### References

- [1] Wujastyk, Dominik. *Standardization of Romanized Sanskrit for Electronic Data Transfer and Screen Representation* [results of a session held at the 8th World Sanskrit Conference, Vienna, 1990], in *Sesame Bulletin* 4(1), 1991, pp. 27-29. Also available as a PostScript document from `CTAN/fonts/csx/csx-doc.ps`.
- [2] Stone, Anthony [ed]. *ISO Committee Draft 15919: Transliteration of Devanagari and Related Scripts into Latin Characters*. Available at [http://ourworld.compuserve.com/homepages/stone\\_catend/trdcd1c.htm](http://ourworld.compuserve.com/homepages/stone_catend/trdcd1c.htm).

◊ Anshuman Pandey  
University of Washington  
Department of Asian Languages  
and Literature  
225 Gowen Hall, Box 353521  
Seattle, WA 98195  
[apandey@u.washington.edu](mailto:apandey@u.washington.edu)  
<http://weber.u.washington.edu/~apandey/>

---

## New Greek fonts and the greek option of the babel package

Claudio Beccari and Apostolos Syropoulos

### Abstract

A new complete set of Greek fonts and their use in connection with the `babel greek` extension is described. Some suggestions are proposed so as to enhance some  $\TeX$  related utilities and some  $\LaTeX 2\epsilon$  font description macros.

### 1 Introduction

*TUGboat* already reported several papers on the possibility of typesetting Greek with  $(\LaTeX)$  $\TeX$ . Perhaps the first paper was the one by Silvio Levy [1] who, so to speak, set forth a standard of fonts and macros in order to set texts both in English (or any other “Latin” written language) and Greek.

His fonts, prepared to be generated with METAFONT v.1.x, were very good replicas of the “standard” Didot Greek fonts; besides digits and punctuation marks, they contained the 24 upper case letters and the 25 lower case ones with all kinds of accents and breaths and implied such ligatures so as to insert the diacritical signs by inputting the corresponding keystrokes before (or after for the iota subscript) the letter to be marked. The correspondence between the keys of a “Latin” keyboard (namely the US keyboard) and the Greek letters and the diacritical signs was established in such a way that everybody, except perhaps Macintosh users who have access to utilities for configuring their keyboards for any “alphabet”, got acquainted with Levy’s convention in such a way as even people, like Beccari, who are not of Greek mother language, can read Greek compuscript text set on the screen with latin characters just as easily as real Greek text set with true Greek fonts.

Levy’s fonts exploited the METAFONT ability to describe fonts with 256 glyphs, but at his time drivers could generally handle only 128 glyph fonts. Haralambous [3] therefore developed a set of Greek fonts with 128 glyphs by reducing Levy’s set in such a way as to keep the substantial and most frequent accent vowel ligatures, and to do away with the automatic setting of the initial/medial sigma as opposed to the final sigma.

Mylonas and Witney [5] soon after proposed a set of fonts based on a main 256 glyph font and its adjunct font (a 128 glyph one) by which they could cover the extended necessities of the full set of Greek glyphs, including any sort of breath-accent-vowel-iota subscript combinations, including the *ou* ligature with its diacritics; in this way they imple-

mented each “alphabet” with more than 300 glyphs. Of course, due to  $\TeX$  limitations, they had to make some compromises for hyphenation of Greek text; as  $\TeX$ ies well know,  $\TeX$  can hyphenate words composed of glyphs taken from the same font, so that when the adjunct set was called for,  $\TeX$  was unable to hyphenate.

Haralambous [4] also described the hyphenation of ancient Greek, but for several years no further advances were done in the field of typesetting Greek, because (we suppose)  $\LaTeX$  was undergoing one of its major changes, the transformation into  $\LaTeX 2\epsilon$  with its standard use of the NFSS (New Font Selection Scheme) which has the ability to deal with several font encodings, and the `babel` package was getting richer and richer thanks to the introduction of the Cork “double” font encoding with 256 glyphs, that tremendously facilitated typesetting all those European and extra European languages that use lots of diacritical signs; moreover Haralambous had started the enormous task of developing  $\Omega$ , a descendant of  $\TeX$  that can handle 16 bit Unicode coded fonts.

Also Dryllerakis [6] generated with METAFONT a set of Greek fonts that included the regular, bold-face, slanted and italic typefaces; these fonts are to play an important role as we shall see soon.

For Greek it was necessary to wait for the constitution of the Greek Society of  $\TeX$  Users, in order to have the enthusiasm and the will to sit down and prepare a complete set of `babel` macros and environments capable of handling the necessary change of font encoding (with the corresponding changes into the catcodes of the various extended ASCII codes) and the switching in and out from Greek or Latin font typesetting.

Syropoulos took the initiative of collaborating with Johannes Braams, the author and curator of `babel` [10], for writing the `greek` option to the `babel` package. When Syropoulos first wrote his `greek.ld` language definition file, he made reference to the Dryllerakis fonts, that were the most complete set at that time.

Apparently everything was settled for the Greek authors and for the Ellenists around the world, because they now had all the necessary tools for typesetting Greek texts, both as main ones and as citations within “Latin” written ones.

Beccari, triggered by a teacher of classical Greek in Italian high schools, was induced to get strongly involved in producing a set of tools for his friend; when he had last examined the CTAN archives he had not noticed the `greek` extension to `babel` nor the Dryllerakis fonts; both had been there for a certain

time, but nevertheless he missed them. He started working on Levy's fonts, but he wanted to generate something that could be just as versatile and complete as J orge Knappen's `ec` "Latin" fonts are [7].

When he finally discovered the existence of both the Dryllerakis fonts and the `greek babel` extension by Syropoulos, he found out that his work, after all, was not a complete waste of time.

In facts Syropoulos could not avoid some short cuts for the lack of optically compatible sans-serif Greek fonts, to the point that the "new"  $\text{\LaTeX} 2_{\epsilon}$  font changing commands had to refer to other (not by Dryllerakis) Greek font families in order to avoid too many font substitutions.

## 2 The `cb` Greek fonts

When Beccari submitted his fonts to Syropoulos, the latter agreed that the `cb` set was more complete and supported the former with lots of helpful suggestions. Beccari ended up with a set of fonts [8], that he unmodestly called `cb` Greek fonts (as well as Kostis Dryllerakis' fonts are named after him `kd` Greek fonts), which is very rich in families, series, and shapes, so that all  $\text{\LaTeX} 2_{\epsilon}$  font changing commands refer to a specific font, and new commands may be defined in addition to the standard ones. According to Beccari's idea, all the Greek fonts are supposed to be optically compatible with one another *and* with the corresponding "Latin" ones. See the appendix for a sample of mixed text written with several families and shapes.

His work led him to conclude that the NFSS idea of encoding, family, series, and shape are possibly incomplete in order to describe a set of fonts, or at least Beccari's imagination was not wide enough to find a better description of the font characteristics.

Beccari decided that his fonts had to be based on Knappen's algorithm for interpolating the various font parameters, just as Knappen's `ec` fonts. So he "borrowed" Knappen's interpolating macros; such METAFONT macros work perfectly with the `ec` fonts; if they work well also with the `cb` fonts, it is just Knappen's merit, should they behave improperly it is just Beccari's demerit.

Beccari worked on the families, series and shapes listed in Table 1; the boldface series applies to all families except the monospaced ones. The outline family has only the medium and bold extended series. Fonts for slides comprise both proportional and monospaced, visible and invisible varieties, but lack the serified proportional shapes, as well as it happens for the "Latin" `ec` fonts.

As Table 1 clearly shows, the `cb` font set is even wider than the standard `ec` fonts directly accessible with the standard  $\text{\LaTeX} 2_{\epsilon}$  font changing commands. Actually the `ec` fonts include some shapes that require special commands to become usable in a document, or require some modifications to the standard font description files.

Syropoulos' `greek.ld` language definition file contains all the necessary commands to invoke any of those valid family, series and shape combinations, and the accompanying `.fd` font description files behave accordingly.

## 3 METAFONT considerations

It is necessary at this point to underline a drawback of the `ec` and `cb` METAFONT source files. METAFONT produces the `.tfm` and `.????gf`<sup>1</sup> files whose name derives from the `jobname` special METAFONT string variable; this `jobname` is assigned a value equal to the name of the first file input by the specific METAFONT run.

This approach does not produce any inconvenience with the standard `cm` fonts; with the `ec` and `cb` fonts, the name of which includes a numerical part equal to (one hundred times) the design size of the specific font, it is somewhat redundant to have hundreds of small files containing just two or three lines of METAFONT code; in facts they simply specify the design size again and then input a "generic" driver file whose task is exactly that of interpolating the font parameters. Take for example the main file for the roman medium normal `ec` (Latin) font:

```
% This is ecrm1000.mf in text format ...
if unknown exbase: input exbase fi;
gensize:=10;
generate ecrm
```

After inputting, if necessary, the base file for the `ec` fonts, the design size is set with a value that actually is already part of the file name (a part a factor of 100), and finally inputs the "generic" file for that family, series and shape.

The `cb` main files are even simpler; for example the main file for the regular medium normal `cb` (Greek) font is:

```
input cbgreek;
```

(and the same line is contained in any other `cb` main file); the trick lies simply in the fact that the file `cbgreek` generates the design size directly from the `jobname` and from the same `jobname` it extracts the "generic" driver file name specific for that family, series and shape.

<sup>1</sup> The `????` part of the file extension may be void, or it is formed by the product of the resolution times the magnification of the used METAFONT mode.

Table 1: The `cb` Greek font families, series and shapes

Family	Series	Shape
regular	medium	normal
outline	bold extended	oblique
sans serif	monospaced	italic
typewriter	invisible	upright italic
sans serif for slides	bold extended invisible	caps-and-small-caps
typewriter for slides	monospace invisible	

Where is the drawback, then? It consists in the hundreds of small specific main files necessary for generating the `jobname` correctly, instead of working the other way around. This implies that the file system gets overloaded with hundreds of small files, individually smaller than the smallest addressable disk memory unit, that nevertheless clog the disk with unnecessary information.

The excellent trick devised by Knappen of including the design size directly in the file name, so that the `jobname` is assigned the correct value, is sort of hijacked by the rigidity of METAFONT that does not allow to assign a value to `jobname` with an explicit assignment. If one could run METAFONT with a command such as:

```
mf \mode:=ljfive; mag:=1; gensize:=10;
  input ecrm
```

with `ecrm.mf` starting with

```
jobname:=jobname & decimal(100gensize);
```

no specific main files would be necessary allowing for an enormous saving of disk space. In computers with file systems that are not too smart, each main file, although smaller than a block or sector (512 bytes), may reserve up to 16Kb or 32Kb of disk space, and the hundreds of main files necessary to generate the `ec` fonts may take up several megabytes of disk space.

With the `cb` fonts Beccari got used to a simple batch file<sup>2</sup> that generates the specific main file, runs METAFONT, and then deletes the now unnecessary main file<sup>3</sup>:

```
@echo off
if "%1"==" " goto message
if "%2"==" " goto dpi600
if "%2"=="600" goto dpi600
if "%2"=="300" goto dpi300
echo "Density %2 non allowed"
```

<sup>2</sup> *Batch file* refers to DOS and related operating systems; other operating systems may use the terminology of *script* or *command* file.

<sup>3</sup> For typesetting reasons some lines are wrapped to the following line; in other words an indented line should be imagined as a continuation of the preceding one.

```
echo "Nothing done "
echo "======"
goto endbatch
:dpi300
set dpi=300
set mfmode=cx
goto dpiset
:dpi600
set dpi=600
set mfmode=ljfive
:dpiset
if exist %1.mf del %1.mf
echo input cbgreek; > %1.mf
mf \mode:=%mfmode%; mag:=1; input %1
if errorlevel 1 goto endbatch
gftopk %1.%dpi%gf
c:\texmf\fonts\pk\%mfmode%\beccari
  \cbgreek\dpi%dpi%\%1.pk
move %1.tfm
c:\texmf\fonts\tfm\beccari\cbgreek
rem
del %1.%dpi%gf
del %1.mf
del %1.log
set dpi=
set mfmode=
goto endbatch
:message
echo Font name missing
:endbatch
```

With this strategy Beccari has the advantage that unnecessary files are always immediately deleted and only the `.tfm` and `.pk` files of the fonts effectively employed are kept on disk.

The disadvantage is that the above batch file can not be handled by those utilities that automatically run the generation of `.tfm` and/or `.pk` files with those  $\TeX$  systems that may call on the fly such programs as `MakeTeXtfm` and/or `MakeTeXpk`<sup>4</sup>. At the same time both `MakeTeXtfm` and `MakeTeXpk` are smart enough to perform more elaborate tasks

<sup>4</sup> Such utilities with the Windows95 or NT based `MikTeX` version 1.10 or later become `maketfm` and `makepk` respectively.



than simply running METAFONT. They could be modified so as to handle both the `ec` and the `cb` fonts in a way similar to the above simple batch file.

Since `MakeTeXtfm` and `MakeTeXpk` are able to recognize the font group from the name first letter(s), it would be very simple to add the following file to the `ec` file bundle:

```
% File ecfonts.mf
% General driver file for ec fonts
if unknown exbase: input exbase fi;
string f_name, f_size;
f_name:=substring(0,4) of jobname;
f_size:=substring(4,8) of jobname;
scantokens("gensize:=\"%f_size&\"/100");
scantokens("generate \"%f_name");
```

so that substituting `ecfont`s to `cbgreek`, the previous batch file and/or the enhanced  $\TeX$  utilities could directly generate the `.tfm` and `.pk` files without the need of overloading the file system with useless files; notice also that since the main driver files are generated on the fly, if you specify in your source  $\LaTeX 2_{\epsilon}$  file something like:

```
\font\myfont=ecrm2600
```

i.e. an `ec` font with a design size that is *not* included in the standard font description file, on running  $\LaTeX$  the required `ecrm2600.tfm` is not found, so that the system is forced to shell out in order to generate it; this task may be performed without error messages by the proposed enhanced `MakeTeXtfm` utility.

This point sets forward another one: the font description files `.fd` for the `ec` fonts (and for the moment also for the `cb` ones) make use of a “name generating” function `genb` that most  $\LaTeX 2_{\epsilon}$  users are unaware of, because, although it frequently gets to play its role, it always operates behind the scenes. This function generates the external font name when a particular non-preloaded `ec` font is called for; but because of the way it is used in the font description files, it can generate names only *for the specified sizes*, not for any size, although the `ec` fonts (as well as the `cb` fonts) can be generated for any size from 5pt to 99.99pt.

Although the `ec` and the `cb` fonts are not vector fonts (well, unless their source files are compiled with Mike Vulis’ `Vmf METAFONT` interpreter that comes with  $\VTeX$ , or unless they are treated with Syropoulos’ perl script `mf2pt3` [9] for the generation of Type 3 fonts, that scale pretty well), under certain points of view they are not too different from the other vector fonts, in the sense that they may be properly scaled (to be precise, designed to the proper size) to almost any size.

Actually the `genb` “file name generating function” that is used in the font description files for the `ec` (and the `cb`) fonts is powerful enough to accept any font size, not only those that are specified in such files. If the font description file for the `T1-cmr` family, for example, contained simple lines such as:

```
\DeclareFontShape
  {T1}{cmr}{m}{n}{<-> genb * ecmr}{}
```

a user could specify in his source `.tex` file:

```
\DeclareFixedFont
  {\myfont}{T1}{cmr}{m}{n}{26}
```

causing the NFSS macros to ask  $\LaTeX 2_{\epsilon}$  to look for the external file `ecrm2600.tfm`, possibly shelling out in order to run `MakeTeXtfm` should that file still be missing. Analogous definitions are usable with the `cb` fonts.

In a similar way the sizes for `text`, `text math`, `display math`, `script` and `sub-subscript`, could be declared differently from the standard sequence with geometrical ratio 1.2; why not  $\sqrt[4]{2}$  or the square root of the golden section?

The enhancement of the utilities `MakeTeXtfm` and `MakeTeXpk`, and of the font description files, as suggested in the preceding paragraphs, would be very handy in the sense that Kappen’s extended fonts<sup>5</sup> (besides the new `cb` Greek fonts) could be treated almost as vector fonts, at least in the range of 5pt–99.99pt.

#### 4 babel greek extension

Greek typesetters as well as Ellenists do not have to bother about the way  $\LaTeX 2_{\epsilon}$  handles the encoding of the Greek fonts compared with that of the Latin ones. Font encoding and character catcodes are dealt with by the internal macros invoked by `babel`’s `greek` extension behind the scenes<sup>6</sup>.

Two Greek languages are actually defined: `greek` and `polutonikogreek`; they share the same hyphenation pattern set, but they typeset internal Greek words according to the modern “monotonic” (default) accent system, as opposed to the classical “polytonic” one. Actually there is no other difference.

Accents are introduced by means of some ordinary ASCII characters, not by means of control characters as it is the case with Latin alphabets: specifically ‘ and " are the only ones dealt with by the monotonic system, while with the polytonic

<sup>5</sup> That is, not only the `ec` fonts, but also the Text Companion fonts that are identified with similar names and generated with the same interpolating macros.

<sup>6</sup> The functionality described here relates to what can be achieved with release 3.7 of the `babel` package.

one there are also ‘, ~, <, and >, plus the iota subscript  $\iota$ . Since these characters are to be treated in a special way, they are catcoded as letters; upercasing changes them (except the diaeresis) to a dummy invisible character so that they disappear. Prefixed accents, breaths, and diaeresis and post-fixed iota subscript interact with the font specifications so as to produce ligatures; those signs that may be prefixed can be specified in any order, that is >’a and >’a produce the same result; with monotonic spelling ’’i and ’’i produce again the same result.

The same ligature mechanism controls the use of final as opposed to initial/medial sigma; if the typesetter is used to type the letter ‘c’ for the final sigma and the letter ‘s’ for the initial/medial one, he can keep doing so, but if he typed always ‘s’ the font characteristics recognize the end of the word and use the proper sigma in the final position; τ’ονος and τ’ονοc in the input file produce the same result in the output file: τόνος.

One simply declares the language selection with the traditional `babel` command

```
\selectlanguage{greek}
```

and for short citations in “Latin” characters within Greek text it is possible to use `\textlatin{...}` which behaves exactly as any other font changing command; on the opposite a Greek short citation within another language may be inserted by means of `\textgreek{...}`. The corresponding declarations are `\latintext` and `\greektext`.

While in Greek mode, the usual font changing commands, such as `\emph` or `\textbf` or `\sffamily`, perform as they are supposed to do except they operate on the Greek fonts. In addition to the other font changing commands, the new command `\textol{...}` allows typesetting with the outline Greek fonts. Remember though that all these commands obey the grouping rules, so that when a group is closed, the font parameters revert to the values they had before entering that group. If you change size, for example, within the Greek environment, when you close the Greek citation you automatically get to `\normalsize` or whatever size you had before.

The new commands `\greeknumeral{...}` and `\Greeknnumeral{...}` allow typesetting a counter value or an explicit number in Greek lowercase or uppercase numerals. With Syropoulos’ `athnum.sty` additional package it is possible to set numbers with Athenian numerals whose glyphs are already contained in the `cb` fonts.

With reference to numbers, and therefore to mathematics, it may be worth noticing that Sy-

ropoulos wrote also the package `grmath.sty` that allows to “ellenize” all the log-like operator names such as `log`, `sin`, `cos`,... This is intended especially for Greek authors who write school books with a special attention to young people who are not used to the corresponding (standard) Latin names.

Of course `\today`, while in Greek mode, typesets the date with the Greek names for the months, but keeps Arabic numerals for the day and the year. Another command, `\Grtoday`, typesets the Greek date using the Greek numerals for the day and the year.

In the  $\text{\LaTeX}2_{\epsilon}$  `enumerate` environment the numbering is redefined in such a way as to use Greek numerals for the numbered items. More specifically, the  $\text{\LaTeX}2_{\epsilon}$  internal commands that translate a number into a lower or upper case letter are redefined in such a way that the number is converted to a Greek numeral expressed by a suitable combination of the 29 Greek numeral symbols; therefore even page numbering, if requested in alphabetic form, turns to Greek numbering while in Greek mode.

Of course before using the `greek` (or the `polutonikogreek`) `babel` extension, it is necessary to rebuild the format with the inclusion of the Greek hyphenation patterns. The Greek bundle includes also the suitable hyphenation file, but it does not become effective (as for any other language) until the format file is rebuilt. Although the `babel` package documentation is explicit on this point, most users fail to notice it, or may be they assume that format rebuilding is automatic.

The necessary steps are quite simple; after verifying that one has the `babel` bundle on the hard disk, it is necessary to locate the file `language.dat`, and edit it so as to append the following lines<sup>7</sup>

```
greek grhyph.tex
=polutonikogreek
```

then run the  $\text{\TeX}$  initializer according to the instructions that come with the  $\text{\TeX}$  system. With `MikTeX` it is very simple, since it suffices to give the line command

```
makefmt latex
```

from a DOS window; with other systems the procedure is very similar although it might be necessary to give more instructions and/or move files from one directory/folder to another.

<sup>7</sup> This might be the right moment for controlling that the loaded hyphenation files correspond exactly to the languages one wants to use; it is convenient to control also that the file names correspond exactly to those one has on the hard disk  $\text{\TeX}$  search path; in case it is possible to fetch the proper files from CTAN.

In order to complete the collection of useful files that complement the basic `greek` option to `babel`, it is worth noticing that Syropoulos wrote also a number of other files that can be very useful for the Greek typesetters as well as the other Ellenists.

A first file `iso8859-7.def` [11] extends the collection of distributed encoding files so as to map directly the keystrokes of a standard Greek keyboard to the internal `cb` font codes; this allows people to enter their  $\LaTeX$  Greek text using actual Greek characters, which makes the `greek` option more user friendly.

A second set of files [11] generates either a single `cb` font driver file (a perl script `gendrv`) or the complete sets of both the text and slide fonts driver files (`cbstdedt.tex` and `cblstded.tex` respectively). At the moment these files are precious; should `makeTextfm` and friends be enhanced as suggested in this paper, their utility would be confined to those systems that do not allow to shell out.

A third file is `hellas.bst` [12], a bibliography style to be used with  $\text{Bi}\TeX$  for creating mixed bibliographies (Greek and non-Greek) in a “consistent” way.

## 5 Conclusion

The appendix shows the appearance of several Greek fonts in line with the corresponding Latin ones; of course the different shape of the single glyphs makes it very evident the change between Latin and Greek, but the use of the same font parameters both for the overall alphabet and for the single characteristics of the strokes (fine, crisp, . . . lines; vertical and horizontal upper and lower case strokes, upper case serifs, etc.) guarantee that there is some optical compatibility between the corresponding Latin and Greek alphabets. Nevertheless the sans serif family turned out more difficult than expected, so that few font parameters had to be modified.

The italic shape for all families and series was completely redesigned trying to get some inspiration from the elegant italic shape named Olga produced by the Greek Font Society [13]; since Beccari is neither an artist nor a good programmer, the result can not be even compared to the original Olga font, but if the constraints imposed by the “metaness” are taken into account, the results may be considered acceptable.

Criticism and suggestions, of course, are welcome.

## References

- [1] Levy S.: “Typesetting Greek”; the file `greekhistory.tex` is available from the CTAN

archives.

- [2] Levy S.: “Using Greek Fonts with  $\TeX$ ”, *TUGboat*, 9(1):20–24.
- [3] Haralambous Y. and Thull K., “Typesetting Modern Greek with 128 Character Codes”, *TUGboat*, 10(3):354–359.
- [4] Haralambous Y., “Hyphenation patterns for ancient Greek and Latin”, *TUGboat*, 13(4):457–469.
- [5] Mylonas C. and Whitney R., “Complete Greek with adjunct fonts”, *TUGboat*, 13(1):39–50.
- [6] Dryllerakis K.: “The `kd` Greek fonts”, available from the CTAN archives.
- [7] Knappen J., The `ec` fonts released on 1997/01/17 together with the sixth upgrade of  $\LaTeX 2_{\epsilon}$ ; the previous (almost definitive) temporary release, called the `dc` fonts, was described in “Release 1.2 of the `dc`-fonts: Improvements to the European letters and first release of text companion symbols”, *TUGboat*, 16(4):381–387.
- [8] Beccari C.: “The METAFONT source files for the `cb` fonts”, available from the CTAN archives in the directory `tex-archive/language/greek/cb/mf`.
- [9] Syropoulos A.: `mf2pt3` perl script, available at <http://obelix.ee.duth.gr/~apostolo>.
- [10] Beccari C., Braams J., Syropoulos A.: “The `babel` bundle for the Greek language”, will be available from the CTAN archives together with the new release 3.7 of `babel`. A preliminary version may be found in `ftp//obelix.ee.duth.gr/pub/TeX`.
- [11] Syropoulos A.: `iso8859-7.def`, `gendrv`, `cbstdedt.tex`, and `cblstded.tex` are available in the CTAN archives in the directory `tex-archive/language/greek/cb/misc`.
- [12] Syropoulos A.: `hellas.bst` is available in the CTAN archives in the directory `tex-archive/language/greek/cb/TeX`.
- [13] Matthiopoulos G.D.: “Oblique or Italics? A Greek Typographical Dilemma”, in *Greek letters – From Tablets to Pixels*, Makrakis M.S. ed., Oak Knoll Press, New Castle, Delaware, 1996.

◇ Claudio Beccari  
Politecnico di Torino  
Turin, Italy  
[beccari@polito.it](mailto:beccari@polito.it)

◇ Apostolos Syropoulos  
Xanthi, Greece  
[apostolo@platon.ee.duth.gr](mailto:apostolo@platon.ee.duth.gr)

## Appendix

This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»

*This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.*

*Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»*

This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»

This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»

This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.

Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»

*This is the beginning of J 1,1-8: «'Εν ἀρχῇ ἦν ὁ Λόγος, καὶ ὁ Λόγος ἦν πρὸς τὸν Θεόν, καὶ Θεὸς ἦν ὁ Λόγος. οὗτος ἦν ἐν ἀρχῇ πρὸς Θεόν. πάντα δι' αὐτοῦ ἐγένετο, καὶ χωρὶς αὐτοῦ ἐγένετο οὐδὲ ἓν ὃ γέγονεν. ἐν αὐτῷ ζωὴ ἦν, καὶ ἡ ζωὴ ἦν τὸ φῶς τῶν ἀνθρώπων. καὶ τὸ φῶς ἐν τῇ σκοτίᾳ φαίνει, καὶ ἡ σκοτία αὐτὸ οὐ κατέλαβεν.*

*Ἐγένετο ἄνθρωπος ἀπεσταλμένος παρὰ Θεοῦ, ὄνομα αὐτῷ Ἰωάννης· οὗτος ἦλθεν εἰς μαρτυρίαν, ἵνα μαρτυρήσῃ περὶ τοῦ φωτός, ἵνα πάντες πιστεύσωσιν δι' αὐτοῦ. οὐκ ἦν ἐκεῖνος τὸ φῶς, ἀλλ' ἵνα μαρτυρήσῃ περὶ τοῦ φωτός.»*

Date: 12 Ἰουνίου 1998

Greek date: ιβ' Ἰουνίου ,αλιη'

Athenian numerals: 1998 = ΧΗΗΗΗΗΗΔΔΔΔΠΠΠ

## Hints and Tricks

‘Hey — it works!’

Jeremy Gibbons

Welcome to ‘*Hey — it works!*’, a column devoted to  $\LaTeX$  tips, tricks and techniques. In this issue we have three articles: one from Jeroen Nijhof showing how to get multi-letter ‘initials’ in  $\BIBTeX$ ; one from me, showing how to use a text symbol (in this case, a hyphen) as an operator in maths; and one from Christina Thiele on generating ornamental rules. My backlog of articles is running low, so please send them in!

◇ Jeremy Gibbons  
 CMS, Oxford Brookes University  
 Oxford OX3 0BP, UK  
 jgibbons@brookes.ac.uk  
<http://www.brookes.ac.uk/~p0071749/>

### 1 Controlling abbreviations in $\BibTeX$

In the  $\TeX$  newsgroup, `comp.text.tex`, someone asked how to get  $\BIBTeX$  to abbreviate Yuri to ‘Yu.’, in styles in which the author’s first names are abbreviated — after all, ‘Yu’ is one letter in Russian. Take for example

```
@Article{govorukhin,
  author = {Yuri Govorukhin},
  title = {What we sow,
           that we reap the fruits of},
  ... }
```

How to get the author abbreviated to ‘Yu. Govorukhin’? Simply putting braces around the ‘Yu’ does not work. Somewhat surprisingly,  $\BIBTeX$  will not abbreviate `{\Yu}ri Govorukhin` at all.

But if one defines a macro `\Yu` in the the preamble that expands to ‘Yu’,

```
@Preamble{"\newcommand\Yu{\Yu}"}
```

then `{\Yu}ri Govorukhin` will be abbreviated to ‘Yu. Govorukhin’.

But as Oren Patashnik, the author of  $\BIBTeX$ , remarked, this solution has one problem. Namely, the ‘Yu’ will be neglected for sorting purposes, so this way the author would be sorted as if he were ‘ri Govorukhin’. To get both a proper abbreviation and proper sorting, one needs to use a feature of  $\BIBTeX$  (which is discussed in the  $\BIBTeX$  documentation `btndoc.dvi`):  $\BIBTeX$  considers a special character, which is everything from a left brace at the top level directly followed by a backslash up to the matching right brace, as one character, and all

characters in it will be taken into account for sorting purposes. So one can define a one-parameter macro `\oneletter`:

```
@Preamble{"\newcommand\oneletter[1]{#1}"}
```

and `{\oneletter{\Yu}ri Govorukhin}` will give the desired effect. There are more tricks in the preamble of `xampl.bib` in the  $\BIBTeX$  distribution.

In our case, though, we can get by without a preamble. The macro in the special character does not have to do anything, it only has to provide a backslash after the opening brace. So we can simply write

```
author = {{\relax \Yu}ri Govorukhin}
```

and here is some text.

◇ Jeroen H. B. Nijhof  
 Aston University  
 Photonics Research Group  
 Birmingham, UK  
[J.H.B.Nijhof@aston.ac.uk](mailto:J.H.B.Nijhof@aston.ac.uk)

### 2 A small minus sign

In a recent paper (*Deriving Tidy Drawings of Trees*, Journal of Functional Programming **6**(3) p. 535–562, May 1996) I had to make a visible distinction between unary minus (for example, the number ‘minus three’) and binary minus (for example, the subtraction ‘four minus three’). I decided to keep the ordinary minus symbol for subtraction, and to use a hyphen for negation: ‘2 – -3’.

The quick-and-dirty way to achieve this is to use an `\hbox`:

```
\def\minus{\hbox{-}}
```

This looks fine most of the time (‘2 – -3’). Unfortunately, the hyphen does not change size in superscripts and subscripts; compare ‘ $x_{y_2-3}$ ’ with how it should look (‘ $x_{y_2-3}$ ’).

A better way is to use  $\TeX$ ’s `\mathchardef`:

```
\mathchardef\minus="002D
```

This defines `\minus` to be an ordinary symbol (the first 0), taken from maths family zero (the second 0), using the character in position 45 (the 2D) of that font, which in the case of Computer Modern Roman happens to be the hyphen character. (There are no guarantees in other fonts, of course!) The  $\LaTeX 2\epsilon$  way of writing this is

```
\DeclareMathSymbol{\minus}
{\mathord}{operators}{"2D}
```

but this has to go in the preamble of the document.

◇ Jeremy Gibbons  
 Oxford Brookes University  
[jgibbons@brookes.ac.uk](mailto:jgibbons@brookes.ac.uk)





## The Treasure Chest

### A package tour from CTAN — `soul.sty`

When the Treasure Chest is CTAN, there's so much to choose from. But even worse ... there are so many packages that keep being added! And how to even find out about them, if you don't keep up with notices posted to the newsgroups? This column is one way to try to bring some of these treasures to *TUGboat* readers, with a quick introduction to the package and some examples of what it can do.

This is the first such column; let me know what aspects are most useful and which ones less so, what additional facets should be examined, what other packages cover some of the same issues; which packages do *you* prefer.

#### 1 Quick tour

##### Package: `soul.sty`

This is version 1.2, dated 11 Jan. 1999. Upon processing, the file `changes.tex` is generated, and describes the differences (the file is also inside the `.dtx` file<sup>1</sup>).

Explanation of the name: “[it] is only a combination of the two macro names `\so` (*space out*) and `\ul` (*underline*) — nothing poetic at all...”

**Keywords:** spacing out, letterspacing, underlining, striking out

##### Purpose:

`soul.sty` provides hyphenate-able letterspacing, underlining, and some variations on each. All features are based upon a common mechanism for typesetting text syllable-by-syllable, using  $\TeX$ 's excellent hyphenation algorithm to find the proper hyphenation points. As well, two examples are presented to show how to use the interface provided to address such issues as ‘an-a-lyz-ing syl-la-bles’. Although the package is optimized for  $\LaTeX 2_{\epsilon}$ , it works under plain  $\TeX$  and  $\LaTeX 2.09$ , and is compatible with other packages, too.

**Author:** Melchior Franz  
a8603365@unet.univie.ac.at

<sup>1</sup> Documented source (`.dtx`) files are a combination of macros and documentation, an evolution of Frank Mittelbach's original DocStrip utility. There are usually two steps: run  $\LaTeX$  over the `.dtx` file to get the documentation, and run  $\LaTeX$  over its matching `.ins` file to generate the style files, which are extracted from the `.dtx` file (sometimes the `.ins` file is itself generated by the first step, which means you only have to pick up the one `.dtx` file — even more compact packaging).

**Compatible with:** plain,  $\LaTeX$  (old and new).

Note: the documentation describes some restrictions when the `soul` package is not used with  $\LaTeX 2_{\epsilon}$ .

##### Location on CTAN:

`/macros/latex/contrib/supported/soul`

**Files to fetch:** `soul.dtx` and `example.cfg`.<sup>2</sup>

**How to install:** Put files with your other class and style files on your system. Read the top portion of `soul.dtx` (or the file `soul.txt`) for instructions on processing the files (you will need  $\LaTeX 2_{\epsilon}$ ). Notice that the `soul.sty` package is not actually on CTAN; it uses the `.dtx` method of documentation, a wonderful feature in  $\LaTeX 2_{\epsilon}$ . If you're unfamiliar with how this works, see footnote 1 for a general overview.

**Files generated:** `soul.ins`, `soul.dvi` (documentation), `soul.toc`, `soul.sty`, `changes.tex`, (as well as the usual `soul.aux` and `.log` files).

#### 2 Documentation

The documentation is so extensive (26 pages long), with explanations, examples of basic use and variations, that little needs to be said here!

The opening pages are a pleasant introduction to the general notions of emphasis, however it is achieved, and the various opinions which exist on the suitability of their use. There is a pragmatism expressed here, offering the user the choice of options, leaving the reasons for such choices to the user.

The user portion of the documentation provides extensive examples and explanations for creating the various effects (underlining, overstriking, letterspacing).

Chapter 7 (pp. 14–25) provides a detailed explanation of the macros themselves, along with some additional points and tips, so do glance through it.

One nice addition from the author (in collaboration with Stefan Ulrich) is a sample configuration file, `example.cfg`, which shows how to select specific spacing values for different fonts automatically, and store them for local use. As well, the local file (call it `soul.cfg` and hooks exist to read it in automatically via `soul.sty`) can be used to store other changes to the package default settings, thus avoiding making changes in either the style file or inserting the customizations into individual source files.

#### 2.1 Table of Contents

##### 1. Introduction

<sup>2</sup> Note: CTAN also has the file `soul.txt` (description of package + processing instructions), and `soul.ins`, which can either be fetched, or generated by processing the `.dtx` file.

2. Typesetting rules
  - (a) Theory ...
  - (b) ... and Practice
3. Modes and options
  - (a) L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> mode
  - (b) Plain T<sub>E</sub>X mode
  - (c) Command summary
4. Letterspacing
  - (a) The macros
  - (b) Some examples
  - (c) Typesetting Fraktur
  - (d) Dirty tricks
5. Underlining
  - (a) Settings
  - (b) Some examples
  - (c) The dvips problem
6. How the package works
  - (a) The kernel
  - (b) The interface
  - (c) Doing it yourself
  - (d) Common restrictions
  - (e) Known features (aka bugs)
7. The macros
  - (a) The preamble
  - (b) Common definitions
  - (c) The letterspacing interface
  - (d) The underlining interface
  - (e) The ~~striking out~~ interface
  - (f) The postamble
  - (g) Additional hacks

### 3 Examples

The following examples, taken directly from the documentation (with a few modifications for TUGboat's narrow columns), provide ample demonstration of the many useful features available in `soul.dtx`.

---

<ul style="list-style-type: none"> <li>■ <code>\so{electrical<sub> </sub>industry}</code></li> <li>■ electrical industry</li> </ul>	<ul style="list-style-type: none"> <li>■ elec- tri- cal in- dus- try</li> </ul>
---	---

*Ordinary text can be typed in as usual.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{man\-\u-\script}</code></li> <li>■ manuscript</li> </ul>	<ul style="list-style-type: none"> <li>■ man- u- script</li> </ul>
---	--

*\- works as usual.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{le<sub> </sub>th{\`e}{\`a}tre}</code></li> <li>■ le théâtre</li> </ul>	<ul style="list-style-type: none"> <li>■ le théâtre</li> </ul>
---	--

*Tokens that belong together have to be grouped; text inside groups is not spaced out. Grouped text must not contain hyphenation points.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{just<sub> </sub>an<sub> </sub>{\hbox{example}}}</code></li> <li>■ just an example</li> </ul>	<ul style="list-style-type: none"> <li>■ just an example</li> </ul>
---	---

*To prevent material with hyphenation points from being spaced out, you have to put it into an `\hbox` (`\mbox`) with two pairs of braces around it. However, it's better to end spacing out before words not to be broken and restart it afterwards.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{inside.}\&amp;\so{outside.}</code></li> <li>■ inside. &amp; outside.</li> </ul>	<ul style="list-style-type: none"> <li>■ in- side. &amp; out- side.</li> </ul>
--	--

*Punctuation marks are spaced out if they are put into the group.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{{'\&lt;Pennsylvania\&lt;'}}</code></li> <li>■ "Pennsylvania"</li> </ul>	<ul style="list-style-type: none"> <li>■ "Penn- syl- va- nia"</li> </ul>
--	--

*Spaceout skips may be removed by typing `\<`. However, it is better to put the quotation marks outside of the argument.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{input\slash<sub> </sub>output}</code></li> <li>■ input/output</li> </ul>	<ul style="list-style-type: none"> <li>■ in- put/ out- put</li> </ul>
---	---

*`\slash`, `\hyphen`, `\endash`, and `\emdash` allow hyphenation before and after the break point.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{unbreakable\~<sub> </sub>space}</code></li> <li>■ unbreakable space</li> </ul>	<ul style="list-style-type: none"> <li>■ un- break- able space</li> </ul>
---	---

*The `\~`-command inhibits line breaks. A space  is mandatory here to mark the word boundaries.*

---

<ul style="list-style-type: none"> <li>■ <code>\so{1\&lt;3<sub> </sub>December<sub> </sub>{1995}}</code></li> <li>■ 13 December 1995</li> </ul>	<ul style="list-style-type: none"> <li>■ 13 De- cem- ber 1995</li> </ul>
---	--

*Numbers should never be spaced out.*



---

■ <code>\so{broken}\line}</code>	■ <code>bro-</code>
■ <code>broken</code>	<code>ken</code>
<code>line</code>	<code>line</code>

*\ works as usual. Additional arguments (e.g., \* or vertical space) are not accepted. Mind the space.*

---

■ <code>\so{\dots\and\hbox{-}}jet}</code>	■ <code>... and</code>
■ <code>... and -jet</code>	<code>-jet</code>

*\hyphen must not be used for leading hyphens.*

---

■ <code>\so{pretty\awful\break}test}</code>	■ <code>pretty</code>
■ <code>pretty</code>	<code>awful</code>
<code>test</code>	<code>aw-</code>
	<code>ful</code>
	<code>test</code>

*The braces keep T<sub>E</sub>X from discarding the space.*

---

#### 4 Applications and comments

For my own purposes, I will most likely find uses for the package in all three T<sub>E</sub>Xs: plain T<sub>E</sub>X (for critical editions), and both old and new L<sup>A</sup>T<sub>E</sub>Xs (for most everything else). In the past, I've had to cobble together very unpresentable macros to deal with overstriking and underlining: both were needed in an article and then a book, to reproduce the creative writing process in Philip Larkin's notebooks. That is, an application which had nothing to do with typographic emphasis and everything to do with trying to reproduce hand-written notes via typesetting. This package would certainly have made the job easier!

What is perhaps not immediately obvious is that this package provides not just various forms of emphasis but emphasis **while retaining hyphenation**. Using `\underline` only works for one line of text, and it blocks the last word from being hyphenated. Devising simple strike-out macros (my case) similarly removes the word(s) inside the macro's argument from being considered for hyphenation.

This package gets over that hurdle, yielding essentially a two-for-one set of tools which many typesetters find they need at the last minute — as they turn the manuscript page, open the next file, and stare at several lines of underlined or over-struck text.

#### 5 Follow-up

I'd like to invite users to fetch this package and see how it works out for them, and then send word on their application and results.

If you've been using other packages (perhaps because they're old and comfortable friends), give this one a try and then tell us how they compare. In particular, make a note of which T<sub>E</sub>X you're using; I myself am hanging on to a number of 'old' things because they work — or is it the other way around . . . that I'm not using the new L<sup>A</sup>T<sub>E</sub>X in all instances because I don't have suitable replacements or substitutes for the old and trusted friends?!

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

#### Helpful hint for finding files on the CTAN archives via ftp

Don't know where to find the package you want? The following shows how to use the `quote site index` command, to quickly locate packages. Our example is for `soul`, the package just presented.

```
ftp> quote site index soul
200-index soul
200-NOTE. This index shows at most 20 lines. for a full list of files,
200-retrieve /tex-archive/FILES.byname
200-1998/12/08 |      3484 | macros/latex/contrib/supported/soul/example.cfg
200-1999/01/12 |     70239 | macros/latex/contrib/supported/soul/soul.dtx
200-1999/01/11 |       279 | macros/latex/contrib/supported/soul/soul.ins
200-1999/01/11 |     1437 | macros/latex/contrib/supported/soul/soul.txt
200 (end of 'index soul')
```

## Abstracts

### *Les Cahiers GUTenberg*

#### Contents of Issue 30

October 1998

#### Hommage à Gérard Blanchard

This issue of the *Cahiers* was produced in a very short time in order to be ready for MultiTypo 98, the ATypI meeting held in Lyon, France in October, to honor Gérard Blanchard, who had just died in August. It includes his last manuscript — his text as invited speaker to ATypI.

JACQUES ANDRÉ and JEAN-FRANÇOIS PORCHEZ,  
Éditorial : ATypI & Blanchard; pp. 3–5

The editors present a brief overview of the ATypI (Association de Typographique Internationale), from its inception in France in 1957, through the great technological changes from lead to computer, till its return to France for the October 1998 annual meeting in Lyon, a city long associated with print.

In addition to the conference publication, *Lettres françaises*, all attendees (courtesy of Louis-Jean Printers) received copies of this 30th issue of the *Cahiers*, which was originally to be a thematic issue on electronic typography but was quickly re-worked to serve as a tribute to Gérard Blanchard, whose interest in typography extended beyond the subject itself, to the many participants involved in different aspects of typography.

In addition to the last Blanchard manuscript, the issue includes tributes from such old friends as John Dreyfus, Fernand Baudin, and Massin. As well, there are two pieces supplementing Blanchard's *Aide au choix de la typo-graphie*: a brief discussion of the methodology behind the work, and an index for it.

JOHN DREYFUS, A Tribute to Gérard Blanchard;  
pp. 6–9

First paragraph from the English text:

The death of Gérard Blanchard in August robbed us of our chance to hear him speak at this Congress. Though he had taken part in several of our earlier congresses, many of you who are not French may never have met him, and may know very little about him. As I had the happy experience of meeting him in the mid-1950s for several consecutive years at the international typographical meetings held at Lurs-en-Provence, I readily agreed to pay this short tribute to his wide ranging contributions to the graphic and the typographic arts.

The text includes several references to Blanchard's works, which might interest the TUGboat reader:

1. an early long essay: "The Typography of the French Book 1800–1914", in *Book Typography 1815–1965*, ed. Kenneth Day. Chicago: U of Chicago Press, 1966, pp. 37–80. [The next chapter is by M. Vox, "The Half Century 1914–1964".]
2. a book: *La Lettre. La lettre et ses usages sociaux*. Éditions du Gymnase typographique, 1975.
3. a large paperback: *Pour une sémiologie de la typographie*. Andenne: Rémy Magermans, 1979.
4. his last book: *Aide au choix de la typo-graphie, cours supérieur*. Reillanne: Atelier Perrousseaux, Éditeur, 1998. [This book includes an extensive bibliography of Blanchard's work.]

FERNAND BAUDIN, Rencontres & confluence  
[Meeting and merging]; pp. 10–11

A personal account of feelings and memories which came to mind upon hearing of Blanchard's death, and how it affects the author's role at the upcoming ATypI meeting (the note was written 6 October, before ATypI).

MASSIN, Des caves d'Hollenstein à la Sorbonne  
[From the Hollenstein cellars to the Sorbonne];  
pp. 12–13

A brief reminiscence of discordant beginnings to what became a lasting friendship with and respect for Blanchard, with a particular memory of Blanchard's intensity in after-hours sessions held in the cellars of Hollenstein Printers in Paris.

GÉRARD BLANCHARD,  
Connotation typographique : Pour une poétique  
de la typo-graphie [Typographic meaning: A case  
for a poetry of typography]; pp. 14–39

Foreword to the article, edited by Jean-François  
Porchez and Jacques André:

This paper contains the text Gérard Blanchard was supposed to deliver as Guest Speaker at the ATypI conference, Lyon, October 1998. He died at the end of August and we edit here his draft, as it was found.

The editors have chosen to simply transcribe Blanchard's manuscript notes, unchanged and unedited, supplemented with scanned images of select pages, to show his creative process at work. Their subsequent article in this issue discusses this process in more detail.

JACQUES ANDRÉ, Noms propres cités dans *Aide aux choix de la typo-graphie* de Gérard Blanchard [Names cited in Gérard Blanchard's *Aide aux choix de la typo-graphie*]; pp. 40–56

Author's abstract:

The following pages include an index of all the names (people, authors, works, founderies) quoted in the following book: Gérard Blanchard, *Aide aux choix de la typo-graphie*, Atelier Perrousseaux éditeur, Reillanne, 1998. ISBN 2-911220-02-1.

N.B. Works names are in italic; fonts are already indexed in the book itself; cover pages are referenced as 0. This index is published here with Gérard Blanchard's and Yves Perrousseaux's authorizations.

JACQUES ANDRÉ and JEAN-FRANÇOIS PORCHEZ, Classeurs et chemin de fer [File folders and thumbnail layout]; pp. 57–62

Authors' abstract:

To prepare his *Aide aux choix de la typo-graphie*, Gérard Blanchard used small loose-leaf binders together with a thumbnail layout (sketches of how each set of facing pages would look). These techniques allowed him to organize his book's layout as a hypertext.

The abstract merely hints at the strategies Blanchard used not simply to organize book layout but also — and I would hazard, primarily — to organize its contents. To capture those thoughts and views which usually come to mind in a very non-linear fashion as an idea begins to take shape and then seemingly races off in a dozen different directions, on at least half a dozen levels: references, examples, suitable quotes, illustrations, cross-references, and so on. In short, an analysis of the creative process at work, whose topic just happens to be typography. Rather like us, using T<sub>E</sub>X to talk about T<sub>E</sub>X!

[Compiled by Christina Thiele]

Articles from *Cahiers* issues can be found in the form of PostScript files at the following site:

<http://www.univ-rennes1.fr/pub/GUTenberg/publications>

## Late-Breaking News

### Production Notes

Mimi Burbank

Here is your last issue of 1998—better late than never. Herein are the promised articles held over from the TUG'98 conference held in Torun, Poland. The article by Miroslava Misáková on page 355 presented quite a challenge. Mirka (her nickname) used DC fonts with Czech encoding and after moving all of the fonts to SCRI, I was still unable to get the font encoding correct (everyone else on the team was either sick or traveling), so in order to finish the issue, I distilled the .ps file used for the preprints and created .eps files of the examples rather than typesetting them.

Several authors and I corresponded frequently about layout and fonts—I would put up pdf documents in my WWW directory for them to peruse using their browser, and they would then email me comments and we would begin again. I find this a satisfactory way of working out problems—of course, depending upon any problems concerning bandwidth across the ocean. I'm have a vision of some shark cruising around the ocean floor, chewing on the “cables”.

**Output** The final camera copy was prepared at SCRI on IBM rs6000 workstations running AIX v4.2, using the *TEX Live* setup (Version 3), which is based on the *Web2c* *TEX* implementation version 7.2 by Karl Berry and Olaf Weber. PostScript output, using outline fonts, was produced using Radical Eye Software's *dvips(k)* 5.78, and printed on an HP LaserJet 4000 TN printer at 1200dpi.

**Coming In Future Issues** I think the news everyone is waiting for is, “When will *TEX Live*4 be available?” The projected completion date of the CD is “Spring of 1999” and it will reportedly be “bigger and better”. The completion date will affect our decision whether or not to delay shipment of the first issue of *TUGboat* until we receive delivery of the CD (otherwise, it will be June 1999 before we can ship).

We will also be posting a list of additional english hyphenation exceptions uncovered since 1995; the full article will be posted on the TUG Web site.

◇ Mimi Burbank  
mimi@scri.fsu.edu

## Calendar

### 1999

- |   |  |
|---|--|
| <p>Jan 29 – Apr 24 <i>Exhibition: Primitive types: The sans serif alphabet from John Soane to Eric Gill, St. Bride Printing Library, London, UK. For information, visit <a href="http://www.stbride.org/soanep.r.htm">http://www.stbride.org/soanep.r.htm</a>.</i></p> <p>Feb 15 <i>TUGboat 20</i> (1), deadline for technical submissions.</p> <p>Feb 24 – 26 DANTE'99, 20<sup>th</sup> meeting, “10 years of DANTE e.V.”, Universität Dortmund, Germany. For information, visit <a href="http://dante99.cs.uni-dortmund.de/">http://dante99.cs.uni-dortmund.de/</a>.</p> <p>Mar 1 <i>TUGboat 20</i> (1), deadline for reports.</p> <p>Mar 1 – 5 Seybold Seminars Boston/Publishing, Boston, Massachusetts. For information, visit <a href="http://www.seyboldseminars.com/Events/bo99/">http://www.seyboldseminars.com/Events/bo99/</a>.</p> <p>Mar 13 – Apr 17 ABeCeDarium: A traveling juried exhibition of contemporary artists' alphabet books by members of the Guild of Book Workers. This show will join historical examples from the Newberry collections. Newberry Library, Chicago, Illinois. Sites and dates are listed at <a href="http://palimpsest.stanford.edu/byorg/gbw">http://palimpsest.stanford.edu/byorg/gbw</a>.</p> <p>Apr 26 – 30 XML Europe '99, Granada, Spain. For information, visit <a href="http://www.gca.org/conf/euro99/">http://www.gca.org/conf/euro99/</a>.</p> <p>May 1 – 3 BachoT<sub>E</sub>X '99, 7<sup>th</sup> annual meeting of the Polish T<sub>E</sub>X Users' Group (GUST), “Practical Aspects of Electronic Publishing”, Bachotek, Brodnica Lake District, Poland. For information, visit <a href="http://www.gust.org.pl/BachoTeX/">http://www.gust.org.pl/BachoTeX/</a>.</p> <p>May 11 – 14 8<sup>th</sup> International World Wide Web Conference, Toronto, Canada. For information, visit <a href="http://www8.org">http://www8.org</a>.</p> <p>May 10 <i>TUGboat 20</i> (2), deadline for technical submissions.</p> <p>May 18 – 20 GUTenberg '99, “L<sup>A</sup>T<sub>E</sub>X, a route to the Web”, l'Institut de physique nucléaire de Lyon, France. For information, visit <a href="http://www.ens.fr/gut/manif/">http://www.ens.fr/gut/manif/</a>.</p> | <p>May 24 <i>TUGboat 20</i> (2), deadline for reports.</p> <p>Jun 1 – 11 Society for Scholarly Publishing, 21<sup>st</sup> annual meeting, Boston, Massachusetts. For information, visit <a href="http://www.sspnet.org">http://www.sspnet.org</a>.</p> <p>Jun 9 – 13 Joint International Conference of the ACH/ALLC in 1999 (Association for Computers and the Humanities, and Association for Literary and Linguistic Computing), University of Virginia, Charlottesville, Virginia, USA. For information, visit <a href="http://www.iath.virginia.edu/ach-allc.99">http://www.iath.virginia.edu/ach-allc.99</a>.</p> <p>Jul 5 – 7 CIDE'99: Second Colloque International sur le Document Électronique, Damascus, Syria. For information, visit <a href="http://infodoc.unicaen.fr/cide/">http://infodoc.unicaen.fr/cide/</a>.</p> <p>Aug 8 – 13 SIGGRAPH, Los Angeles, California. For information, visit <a href="http://www.siggraph.org/s99/">http://www.siggraph.org/s99/</a>.</p> <p>Aug 15 – 19 <b>TUG'99</b> — The 20<sup>th</sup> annual meeting of the T<sub>E</sub>X Users Group, “T<sub>E</sub>X Online — Untangling the Web and T<sub>E</sub>X”, University of British Columbia, Vancouver, Canada. Information will be posted to <a href="http://www.tug.org/tug99/">http://www.tug.org/tug99/</a> as plans develop.</p> <p>Aug 23 <i>TUGboat 20</i> (3), deadline for reports.</p> <p>Sep 20 – 23 EuroT<sub>E</sub>X '99, the XIth European T<sub>E</sub>X Conference, “Document Publishing”, Heidelberg, Germany. Tutorials will precede and follow the main conference. For information, visit <a href="http://www.dante.de/eurotex99">http://www.dante.de/eurotex99</a>.</p> <p>Nov 8 <i>TUGboat 20</i> (4), deadline for technical submissions.</p> <p>Nov 22 <i>TUGboat 20</i> (4), deadline for reports.</p> <p>Dec 6 – 9 XML 99, Philadelphia, Pennsylvania. For information, visit <a href="http://www.gca.org/conf/conf1996.htm">http://www.gca.org/conf/conf1996.htm</a>.</p> |
|---|--|

---

### 2000

- Jul-Aug **TUG 2000** — The 21<sup>st</sup> annual meeting of the T<sub>E</sub>X Users Group, in the UK.

*Status as of 20 December 1998*

For additional information on TUG-sponsored events listed above, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: [office@tug.org](mailto:office@tug.org)). For events sponsored by other organizations, please use the contact address provided.

Additional type-related events and news items are listed in the Sans Serif Web pages, at <http://www.quixote.com/serif/sans>.

## Institutional Members

Academic Press,  
*San Diego, CA*

American Mathematical Society,  
*Providence, Rhode Island*

CERN, *Geneva, Switzerland*

College of William & Mary,  
Department of Computer Science,  
*Williamsburg, Virginia*

CSTUG, *Praha, Czech Republic*

Elsevier Science Publishers B.V.,  
*Amsterdam, The Netherlands*

Florida State University,  
Supercomputer Computations  
Research, *Tallahassee, Florida*

Hong Kong University of  
Science and Technology,  
Department of Computer Science,  
*Hong Kong, China*

Institute for Advanced Study,  
*Princeton, New Jersey*

Institute for Defense Analyses,  
Center for Communications  
Research, *Princeton, New Jersey*

Iowa State University,  
Computation Center,  
*Ames, Iowa*

Kluwer Academic Publishers,  
*The Netherlands*

Los Alamos National Laboratory,  
University of California,  
*Los Alamos, New Mexico*

Marquette University,  
Department of Mathematics,  
Statistics and Computer Science,  
*Milwaukee, Wisconsin*

Masaryk University,  
Faculty of Informatics,  
*Brno, Czechoslovakia*

Mathematical Reviews,  
American Mathematical Society,  
*Ann Arbor, Michigan*

Max Planck Institut  
für Mathematik,  
*Bonn, Germany*

New York University,  
Academic Computing Facility,  
*New York, New York*

Princeton University,  
Department of Mathematics,  
*Princeton, New Jersey*

Space Telescope Science Institute,  
*Baltimore, Maryland*

Springer-Verlag,  
*Heidelberg, Germany*

Stanford University,  
Computer Science Department,  
*Stanford, California*

Stockholm University,  
Department of Mathematics,  
*Stockholm, Sweden*

University of California, Irvine,  
Information & Computer Science,  
*Irvine, California*

University of Canterbury,  
Computer Services Centre,  
*Christchurch, New Zealand*

University College,  
Computer Centre,  
*Cork, Ireland*

University of Delaware,  
Computing and Network Services,  
*Newark, Delaware*

Universität Koblenz–Landau,  
Fachbereich Informatik,  
*Koblenz, Germany*

University of Oslo,  
Institute of Informatics,  
*Blindern, Oslo, Norway*

University of Texas at Austin,  
*Austin, Texas*

Uppsala University,  
Computing Science Department,  
*Uppsala, Sweden*



**Promoting the use of  
TeX throughout the  
world**

*mailing address:*  
P.O. Box 2311  
Portland, OR 97208-2311 USA

*shipping address:*  
1466 NW Naito Parkway,  
Suite 3141  
Portland, OR 97209-2820 USA

Phone: +1 503 223-9994  
Fax: +1 503 223-3960  
Email: [office@tug.org](mailto:office@tug.org)  
WWW: [www.tug.org](http://www.tug.org)

President: Mimi Jett  
Vice-President: Kristoffer Høgsbro Rose  
Treasurer: Donald W. DeLand  
Secretary: Arthur Ogawa

**1999 TUG Order Form**

Rates for TUG membership, subscription, and products are listed below. Please check the appropriate boxes and mail payment (in US dollars, drawn on a United States bank) along with a copy of this form. If paying by credit card, you may fax the completed form to the number at left.

- 1999 TUGboat includes Volume 20, nos. 1-4.
- 1999 CD-ROMs include TeX Live 4 (1 disk) and Dante's CTAN (3 disk set).
- *Multi-year orders:* You may use this year's rate to pay for more than one year of membership or of the CD-ROM product.
- Orders received after March 1, 1999: please add \$10 to cover the additional expense of shipping back numbers of TUGboat and CD-ROMs.

	Rate	Amount
<b>Annual membership</b> for 1999 (TUGboat and CD-ROMs) <input type="checkbox"/>	\$65	_____
<b>Student membership</b> for 1999 (TUGboat and CD-ROMs) (please attach photocopy of 1999 student ID) <input type="checkbox"/>	\$35	_____
<b>CD-ROM product</b> for 1999 (CD-ROMs and TUGboat) <input type="checkbox"/>	\$65	_____
<b>Subscription</b> for 1999 (TUGboat and CD-ROMs) (Non-voting) <input type="checkbox"/>	\$75	_____
<b>Shipping charge</b> if after March 1, 1999. <input type="checkbox"/>	\$10	_____
<b>Voluntary donations</b>		
General TUG contribution <input type="checkbox"/>		_____
Contribution to Bursary Fund* <input type="checkbox"/>		_____
		<b>Total \$</b> _____

**Payment** (check one)  Payment enclosed  Charge Visa/Mastercard

Account Number: \_\_\_\_\_

Exp. date: \_\_\_\_\_ Signature: \_\_\_\_\_

\*The Bursary Fund provides financial assistance to members who otherwise would be unable to attend the TUG Annual Meeting.

Please complete all applicable items; we need this information to mail you products, publications, notices, and (for voting members) official ballots.

*Note:* TUG neither sells its membership list nor provides it to anyone outside of its own membership.

Name: \_\_\_\_\_

Department: \_\_\_\_\_

Institution: \_\_\_\_\_

Address: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

Phone: \_\_\_\_\_ Fax: \_\_\_\_\_

Email address: \_\_\_\_\_

Position: \_\_\_\_\_ Affiliation: \_\_\_\_\_

## T<sub>E</sub>X Consulting & Production Services

Information about these services can be obtained from:

**T<sub>E</sub>X Users Group**  
1466 NW Naito Parkway, Suite 3141  
Portland, OR 97209-2820, U.S.A.

**Phone:** +1 503 223-9994  
**Fax:** +1 503 223-3960  
**Email:** [office@tug.org](mailto:office@tug.org)  
**URL:** <http://www.tug.org/consultants.html>

### North America

#### Hargreaves, Kathryn

135 Center Hill Road,  
Plymouth, MA 02360-1364;  
(508) 224-2367; [letters@cs.umb.edu](mailto:letters@cs.umb.edu)

I write in T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, METAFONT, MetaPost, PostScript, HTML, Perl, Awk, C, C++, Visual C++, Java, JavaScript, and do CGI scripting. I take special care with mathematics. I also copyedit, proofread, write documentation, do spiral binding, scan images, program, hack fonts, and design letterforms, ads, newsletters, journals, proceedings and books. I'm a journeyman typographer and began typesetting and designing in 1979. I coauthored *T<sub>E</sub>X for the Impatient* (Addison-Wesley, 1990) and some psychophysics research papers. I have an MFA in Painting/Sculpture/Graphic Arts and an MSc in Computer Science. Among numerous other things, I'm currently doing some digital type and human vision research, and am a webmaster at the Department of Engineering and Applied Sciences, Harvard University. For more information, see: <http://www.cs.umb.edu/kathryn>.

#### Loew, Elizabeth

President, T<sub>E</sub>Xniques, Inc.,  
362 Commonwealth Avenue, Suite 5E,  
Boston, MA 02115;  
(617) 670-1916; Fax: (617) 670-1916  
Email: [elizabeth@texniques.com](mailto:elizabeth@texniques.com)

Long-term experience with major publisher in preparing camera-ready copy or electronic disk for printer. Complete book and journal production in the areas of mathematics, physics, engineering, and biology. Services include copyediting, layout, art sizing, preparation of electronic figures; we keyboard from raw manuscript or tweak T<sub>E</sub>X files.

#### Ogawa, Arthur

40453 Cherokee Oaks Drive,  
Three Rivers, CA 93271-9743;  
(209) 561-4585  
Email: [Ogawa@teleport.com](mailto:Ogawa@teleport.com)

Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> document classes and packages. Instruction, support, and consultation for workgroups and authors. Application development in L<sup>A</sup>T<sub>E</sub>X, T<sub>E</sub>X, SGML, PostScript, Java, and βC++. Database and corporate publishing. Extensive references.

### Outside North America

#### DocuT<sub>E</sub>Xing: T<sub>E</sub>X Typesetting Facility

43 Ibn Kotaiba Street,  
Nasr City, Cairo 11471, Egypt  
+20 2 4034178; Fax: +20 2 4020316  
Email: [main-office@DocuTeXing.com](mailto:main-office@DocuTeXing.com)

DocuT<sub>E</sub>Xing provides high-quality T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X typesetting services to authors, editors, and publishers. Our services extend from simple typesetting and technical illustrations to full production of electronic journals. For more information, samples, and references, please visit our web site: <http://www.DocuTeXing.com> or contact us by e-mail.