TUGboat

Volume 31, Number 1 / 2010

TEX Users Group

TUGboat (ISSN 0896-3207) is published by the TEX Users Group.

Memberships and Subscriptions

2010 dues for individual members are as follows:

- Ordinary members: $95.
- Students/Seniors: $55.

The discounted rate of $55 is also available to citizens of countries with modest economies, as detailed on our web site.

Membership in the TEX Users Group is for the calendar year, and includes all issues of *TUGboat* for the year in which membership begins or is renewed, as well as software distributions and other benefits. Individual membership is open only to named individuals, and carries with it such rights and responsibilities as voting in TUG elections. For membership information, visit the TUG web site.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is $100 per year, including air mail delivery.

Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TEX and the TEX Users Group, as well as providing a discounted group rate and other benefits. For further information, see `http://tug.org/instmem.html` or contact the TUG office.

TEX is a trademark of the American Mathematical Society.

Addresses

TEX Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Telephone

+1 503 223-9994

Fax

+1 206 203-3960

Web

`http://tug.org/`
`http://tug.org/TUGboat/`

Electronic Mail
(Internet)

General correspondence, membership, subscriptions:
`office@tug.org`

Submissions to *TUGboat*, letters to the Editor:
`TUGboat@tug.org`

Technical support for TEX users:
`support@tug.org`

Contact the Board of Directors:
`board@tug.org`

Have a suggestion? Problems not resolved?
The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: May 2010]

Printed in U.S.A.

Consistency in spelling is difficult when two alphabets are involved. . .

Ali Ahmad Jalali and Lester W. Grau
The Other Side of the Mountain: Mujahideen Tactics in the Soviet-Afghan War (1995)

TUGBOAT

COMMUNICATIONS OF THE TEX USERS GROUP
EDITOR BARBARA BEETON

VOLUME 31, NUMBER 1 • 2010
PORTLAND • OREGON • U.S.A.

TUGboat

This regular issue (Vol. 31, No. 1) is the first issue of the 2010 volume year. No. 2 will contain the TUG 2010 (San Francisco) proceedings and No. 3 will contain papers from the EuroTEX 2010 conference in Pisa, Italy.

TUGboat is distributed as a benefit of membership to all current TUG members. It is also available to non-members in printed form through the TUG store (`http://tug.org/store`), and online at the *TUGboat* web site, `http://tug.org/TUGboat`. Online publication to non-members is delayed up to one year after an issue's print publication, to give members the benefit of early access.

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 deadline for receipt of final papers for the upcoming TUG 2010 proceedings issue is July 19, and for regular material for the third issue is September 1.

As always, suggestions and proposals for *TUGboat* articles are gratefully accepted and processed as received. Please submit contributions by electronic mail to `TUGboat@tug.org`.

The *TUGboat* style files, for use with `plain` TEX and LATEX, are available from CTAN and the *TUGboat* web site. We also accept submissions using ConTEXt. More details and tips for authors are at `http://tug.org/TUGboat/location.html`.

Effective with the 2005 volume year, submission of a new manuscript implies permission to publish the article, if accepted, on the *TUGboat* web site, as well as in print. Thus, the physical address you provide in the manuscript will also be available online. If you have any reservations about posting online, please notify the editors at the time of submission and we will be happy to make special arrangements.

Other TUG publications

TUG is interested in considering additional manuscripts for publication, such as manuals, instructional materials, documentation, or works on any other topic that might be useful to the TEX community in general.

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 at `tug-pub@tug.org`.

TUGboat advertising

For information about advertising rates and options, including consultant listings, write or call the TUG office, or see our web pages:
`http://tug.org/TUGboat/advertising.html`
`http://tug.org/consultants.html`

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 commonly appear in *TUGboat* should not be considered complete.

METAFONT is a trademark of Addison-Wesley Inc.
PostScript is a trademark of Adobe Systems, Inc.
TEX and $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX are trademarks of the American Mathematical Society.

From the President

Karl Berry

Conferences

TUG 2010 (http://tug.org/tug2010) takes place at the end of June. We are especially excited about this year's conference, as the 2^5 anniversary of TEX78. Donald Knuth and most of his Stanford colleagues from the early days of TEX development have indicated they will attend, and we have planned several events in their honor. Its proceedings will be the second *TUGboat* issue for 2010.

EuroTEX 2010 (http://www.guit.sssup.it/eurotex2010) will take place in Pisa, Italy; that proceedings will account for (at least) most of the third issue.

Another upcoming conference is the fourth ConTEXt user meeting, this year in Brejlov (Prague), Czech Republic, from September 13–18, 2010 (http://meeting.contextgarden.net/2010).

Looking ahead to 2011, the TUG meeting will be held in Cairo, Egypt, from November 28 through December 1. Hossam Fahmy, long-time TUG supporter and *TUGboat* contributor, is the chief organizer. http://tug.org/tug2011 will be updated as planning proceeds.

http://tug.org/meetings has information on most TEX meetings, past, present, and future.

Interviews

In keeping with the anniversary celebration, Dave Walden, coordinator of the Interview Corner (http://tug.org/interviews), has interviewed several of the early Stanford participants: Luis Trabb Pardo, Howard Trickey, John Hobby, and Michael Plass, as well as Tom Rokicki and David Fuchs earlier, with others in progress. Other interviewees since my last column are Ulrike Fischer, Idris Hamid, and Joseph Wright.

TEX Collection 2010

We have hopes that the software release for 2010 will be finished rather earlier than the past few years. By the time this issue reaches mailboxes, online updates of TEX Live 2009 will have been frozen so that we can finalize the changes for 2010.

Although nothing is certain, we expect some of the most notable changes in TEX Live to be (not an exhaustive list):

- Automatic conversion of EPS graphics to PDF when running under pdfTEX.
- pdfTEX will output PDF version 1.5 by default, enabling better-compressed PDF output.
- Inclusion of the Japanese TEX suite, pTEX.
- Oren Patashnik has released the first new version of BibTEX in many years, with one small bug fix to avoid breaking .bbl output lines anywhere except at whitespace.

We expect the 2010 collection to contain the same major items as in the past few years: TEX Live, MacTEX, proTEXt, and a CTAN snapshot. See http://tug.org/texcollection for more.

We welcome anyone's participation, from testing the final candidate release to core development. And thanks to all the many, many, people involved already at every level.

TEX journals

Careful readers of *TUGboat* will have noticed that the list of active TEX journals is growing. *TUGboat* reprints abstracts, in English translation where necessary, from all TEX journals. Some notable events in the journal world:

- The UK-TUG journal *Baskerville* was revived in 2009, with new editor Jonathan Webley, and is available both online and in print.
- The longstanding GUTenberg journal *Cahiers*, with new editor Thierry Bouche, is once again very active, having caught up on all back issues.
- The Korean TEX Society started publication of *The Asian Journal of TEX* in 2007, with articles in several languages.
- As this issue of *TUGboat* went to press, *The PracTEX Journal* published a new issue, numbered 2008-4, with a collection of selected articles since its founding in 2005.

Some journals make all articles available online immediately, such as *Eutypon* (Greek TEX Friends), *Baskerville*, *The Asian Journal of TEX*, and *The PracTEX Journal*. Other journals, such as *Cahiers*, *MAPS* (NTG), *Die TEXnische Komödie* (DANTE e.V.), *Zpravodaj* (CSTUG), and *TUGboat* itself, all have approximately the same policy, of making articles available to user group members and subscribers for a year before posting them publicly.

For links and other information about all extant TEX journals, see http://tug.org/pubs.html. And let us know if we missed one.

◇ Karl Berry
president (at) tug dot org
http://tug.org/TUGboat/Pres/

Editorial comments

Barbara Beeton

TEX at 2^5

I don't know about anybody else, but I'm looking forward eagerly to this year's TUG meeting in San Francisco. While it won't be on the Stanford campus, where it all started, and where quite a few of the early meetings took place, it's close enough, and an important enough anniversary, that Don Knuth has been persuaded to share some of his precious time with us. A substantial number of Don's students who worked on the TEX project will also be there, if only in anticipation of a good party, since all of them have gone off in other directions. Since 2^6 is 32 years away, it's unlikely we'll all be around to celebrate that anniversary together, though it might be interesting to place wagers on whether TEX itself will still be around.

Speaking of Don, he has received yet another prize — one of the Katayanagi Prizes in Computer Science, awarded annually by Carnegie Mellon University in cooperation with the Tokyo University of Technology. The announcement can be found here: `http://cacm.acm.org/magazines/2010/3/76269-katayanagi-prizes-and-other-cs-awards` (thanks to Nelson Beebe for spotting this).

Pi Day

March 14 marked Pi Day; more precisely, it should have beem marked at 1:59 (local time, of course). Stephen Hicks (whose acquaintance I've not had the pleasure of making) sent the following offering to `texhax`, to introduce himself and celebrate the day. Try it for yourself. It requires plain TEX.

```
\let~\catcode
~‘z0~‘‘1~‘,2~‘q13~‘z#
14~‘46zdefq41’~‘4113zgdef,q
QQ41’~‘4113zlet,qBB415425’41-P#
7427,QPPzexpandafterqAA414243’H#H
42434341542415,qCC41742743’41i-D42#
434343,qww’,zedefw’PPPCPBAyap!,qEE’,q
6641’~‘4113zcountdef,QNNzifnum6RR1R20E6
YY2QAAzadvanceY-RAY1QMMzmultiply6XX3EqS
S’Ezhskip0.5em,EQJJzjobnameEQmmwE6TT5qj
j4142x’41zgdefm’42,,EQGGzglobalE6CC7Eqr
r41’T41MT41ACT,qHH’C0rXrYC-CrR,qOO’zifx
mEPjwxEzelsePjmxzfi,qcc’NX<RHNC<0Szelse
OEzfiAX1PczelseEGAY2zfi,EzedefzJ’J,qv
v’@,zifxzJvQOO*zfiEqll’NY<RX-Rzhbox
’c,Plzfi,zttlzbye3.14159265358979
323846264338327950288419716939937
37510582097494459230781640628
2862089986280348253421170679821...
```

The @ sign as a design icon

In the March 21 issue of the *New York Times* is an announcement that the Museum of Modern Art (MoMA) in New York has admitted the @ sign into its permanent architecture and design collection. (`www.nytimes.com/2010/03/22/arts/design/22iht-design22.html`)

The rationale is that the use of this sign in e-mail addresses excels "in terms of form and function[.] Does it embody the values of clarity, honesty and simplicity that MoMA considers essential to good design? Has it made an impact on our lives? Is it innovative?" The senior curator of architecture and design at MoMA, Paola Antonelli, asks "If this object had never been designed or manufactured, would the world miss out?" Absolutely!

The use of the @ to indicate a destination in an e-mail address was instigated by Ray Tomlinson, then at Bolt, Beranek & Newman.

I was able to ask Ray some questions about his choice.

bb: Were there any other candidates?
RT: Not really. I wanted to avoid characters that appeared in the login names on various operating systems. This eliminated letters, numerals, comma, period, square brackets and parentheses. Others such as curly braces, tilde, back quote were in the lower case set of characters and didn't appear on all terminals. Of the remainder, the at-sign leaped out as the obvious choice. It satisfied the criteria and just made sense. Later, I discovered that the at-sign was the line erase character on the 2741 terminal used by Multics. Dave probably remembers the grief that Multics users suffered every time they typed the at-sign. In order to type in an at-sign, you had to type it twice in a row. After the first one, the terminal controller locked the keyboard so you had to wait a second or two before the keyboard would unlock so you could type the second at-sign.

bb: I don't remember when computer keyboards finally recognized the difference between upper- and lowercase letters, but they did have punctuation and a few symbols (though not necessarily the same set as available on standard typewriters).
RT: There were computer terminals around that had both upper and lower case characters. The aforementioned 2741 was one such terminal and more expensive models of Teletype terminals also had lower case characters. But there were many that were restricted to upper case.

bb: How much of the "full" current format of Internet addresses was part of the original design? Was it sufficient to have just `<user name> @ <host>`, or

was there a need to provide for something more elaborate?

RT: Email addresses have always been

<user>@<host>,

what has changed is <user> and <host>. At first, user names varied. For example, some were numeric. As time went by user names shifted towards being word-like if not actually words. Host names had little structure and certainly did not have .com, .org, or .edu at the end. The namespace was flat and typically started with the name or abbreviation of a company or university followed by something to distinguish one computer from another at that site. So, BBN-TENEXA was our main computer system. BBN-TENEXB was the development and test machine. Later the Domain Name System (DNS) came along to make computer name assignment more manageable.

bb: I'm delighted to have this opportunity, and want to extend a big "thank you" for helping to create this beast that we sometimes love, and sometimes hate, but on which we have certainly grown totally dependent.

RT: You are welcome.

Amusements on the Web

Do you need something typographical to decorate your wall? The Periodic Table of Typefaces might be just the thing. Check out www.squidspot.com/Periodic_Table_of_Typefaces/Periodic_Table_of_Typefaces_large.jpg.

Other "periodic tables" can be found with a bit of help from Google. This one, illustrating visualization methods, might come in handy when preparing a beamer presentation: www.visual-literacy.org/periodic_table/periodic_table.html.

Another site about visual communication is www.citrinitas.com/history_of_viscom/. Sadly, I found the text rather difficult to read, as it's white sans serif on a black background, but the illustrations are splendid.

For a really spectacular introduction to a book, the New Zealand Book Council has destroyed the book they're inviting you to read, to make a stop-motion animation, but with results that are hauntingly beautiful. This book is on my reading list, and I've bookmarked the video: www.creativepro.com/article/stop-motion-animation-book-lovers.

Videos of typography talks on the Web

For several years, Kaveh Bazargan has been recording talks at various conferences on TEX, typography, publishing, and other topics. The River Valley TV site (river-valley.tv) is home to this collection of videos. The full schedules of TUG 2007–2009 are included. This is an invaluable resource, and we are most grateful to Kaveh for his generous contribution of time and resources.

One non-TEX video well worth searching out is a talk by John Hudson of Tiro Typeworks on scholarly types, presented at the Type[&]Design 2009 conference organized by DTL in the Hague. One of the topics covered by the talk is the Cambria Math font, developed by Tiro Typeworks for Microsoft. (This font is at least partially supported by XETEX.) The video is at river-valley.tv/scholarly-types/ and the accompanying slides are at www.fonttools.org/downloads/TD_2009/Scholarly_Types.pdf.

Other talks from Type[&]Design 2009 may also be of interest. On the main page at River Valley for this conference is a link to the conference site. At the top of the conference home page is a "Stop Press!" announcement — the Dr. Peter Karow Award for Thomas Milo. The citation for the award recognizes "the development of the ACE layout engine (the heart of the Tasmeem plugin for InDesign ME) for Arabic text setting." Attendees at EuroTEX 2003 may remember Thomas Milo's talk "ALI-BABA and the 4.0 Unicode characters — Towards the ideal Arabic working environment" (*TUGboat* 24:3, pp. 502-511).

Alphabet soup

Another web site ... This one describes a project that "attempts to determine a number of things about the shapes of letters in several different writing systems." The building blocks discovered by examining various alphabets are operated on by a set of rules, "a grammar or syntax", to combine into recognizable letters that are nevertheless not of the "standard" form.

The program is written in Python, is downloadable from the site, and experimentation is encouraged. Find it at www.theory.org/artprojects/alphabetsoup/main.html.

◇ Barbara Beeton
American Mathematical Society
201 Charles Street
Providence, RI 02904 USA
tugboat (at) tug dot org

An argument for learning LaTeX: The benefits of typesetting and beyond

Evan Wessler

Abstract

I discovered LaTeX more or less by accident, and I could not have estimated the benefits learning the typesetting system would confer. Here, I argue for the merits of LaTeX from perspectives apart from/stemming out of typesetting.

Introduction

As an undergraduate biology major, I had little reason and no impetus to leave the world of the WYSIWYG word processor for an advanced typesetting system. After all, the documents I was producing featured almost exclusively text, with an occasional chemical formula (e.g. $CaCl_2$) or simple mathematical equation (e.g. for linear regression analyses) for use in laboratory reports. It was by chance that I was introduced to LaTeX during my sophomore year of college, when a friend who had used it to typeset a bioengineering paper happened to send me his source and output. I became interested in the typesetting system immediately, because I had recognized the appearance of the output (i.e. the Computer Modern font and the well-formatted mathematics) as similar to that which I had seen on my calculus exams and homework sets. (Admittedly, I was always impressed with the aesthetics of these documents, and had [in retrospect, rather embarrassingly] tried to replicate their style in Microsoft Word, to no avail.) I quickly learned the ins and outs of the typesetting system, and ever since have been a regular user, enthusiast, and unabashed proponent of LaTeX versus conventional word processing and presentation software.

Over the past three years, I have used LaTeX to produce an array of documents: analytical chemistry laboratory reports, formal business letters, physics equation sheets, charts, a symposium presentation... the list goes on. All of them were higher in quality — in terms of aesthetics — than comparable documents produced by my peers (most of whom used Microsoft Office or OpenOffice) for the same purposes. However, in looking back on my experience with typesetting, I realize that there are many benefits of a *non-aesthetic* nature to learning and using LaTeX. In the remainder of this article, I will present and assess these advantages, and explain how learning the typesetting system has developed my skills, both in typesetting and beyond.

Problem solving

All users of the LaTeX typesetting system — experienced or not — are intimately familiar with LaTeX error messages. These notifications appear when errors in the source are encountered during typesetting. An experienced user knows they can be due to a number of things, among which are incorrectly-spelled commands, missing or extraneous brackets, failure to close environments, and other errantly typed and/or conceived text in the source. However, a new user — that is, one who is new to LaTeX and has no experience in dealing with code-based, debuggable source entry (e.g. in computer programming) — will be unfamiliar with the presentation and interpretation of errors, as well as with the proper action(s) that must be taken to correct them. This process is often non-trivial, because it is potentially not as straightforward as the new user might assume. For example, a message may indicate that there is something wrong at a certain line, whereas the incorrect element may actually be present several lines before the stated location. In addition, the very syntax of error messages may be puzzling (bewilderment with errors during creation of complex tables comes to mind). It is also the case that some commands and environments cannot be used in tandem (e.g. the `\verb` command, I found, cannot be used as-is inside `\section{}` commands). Incompatible environments and non-global commands may be unknown to the author; the consequent errors are often the source of frustrating problems that require an advanced solution.

Thus, to be able to produce a correctly typeset document using LaTeX, one must become proficient at troubleshooting. The process may be as simple as searching a few lines for errant symbols, or trying different commands. However, a particularly perplexing error message containing ambiguous and unhelpful language may demand more creative solutions. An especially useful approach that I discovered at some point early on is what I will call the "incremental comment-out" strategy. This involves systematically commenting-out (i.e. marking lines with the "%" symbol, so that they are ignored by the typesetting engine) different short segments of the source in succession, and attempting to typeset each time. For an error message that reveals little to no information about the true location of the error, this tactic is invaluable; one sees that the document fails to typeset every time except for when the region in which the error is contained is commented-out. In this way, the error is pinpointed and can be corrected. This method may be dismissed as inefficient by those

users who produce very large documents (intermittent typesetting is always advisable) and those who are more experienced and highly-versed in the nuances of LaTeX warnings; however, to the novice, this is a good way to learn about error syntax, typical problems encountered, and the methods behind locating and fixing mistakes. Of course, once an error is found, solving the problem is usually a matter of referencing any decent LaTeX manual; but in order to get to this point, the significant work of finding and understanding the error must be performed by the user. Any exercise of this nature is bound to increase one's capacity and ability to problem solve.

Taking command of the command line

Before I started using LaTeX, the command-line interface was largely unknown to me; its seemingly obscure commands and cold, intimidating appearance (its operations, after all, are executed using *just* text, which was difficult for someone like myself — who started using computers when graphical user interfaces had already become the norm — to accept) made it esoteric at best. I had previously dabbled in the Windows "Command Prompt" program, but had no real idea of what I was doing. (Fortunately, this did not lead to any catastrophes.) Upon my first introduction to LaTeX, I edited and typeset my documents in the way that most new users probably do; I wrote the source in a user-friendly GUI front-end program (in my case, Richard Koch's "TeXShop") and hit the "Typeset" button. It was only when I started reading up on LaTeX that I discovered that editing, typesetting, previewing and printing could be accomplished within a terminal (via the use of editors such as Emacs and Vim, and by issuing commands such as `latex` and `xdvi`). This discovery led me to utilize the terminal more often, and in turn to start experimenting with its other uses.

Since then, I have become fairly proficient at the command line. But this is not just for the sake of using it as a neat or exhibitionist alternative to the GUI (I have become convinced that anyone unfamiliar with the terminal who sees me using it thinks I am either up to no good, or that I am performing operations too sophisticated to be relevant to the "normal" computer user); I regularly use it to perform necessary functions (e.g. secure shell, efficient exploration, creation, copying and moving of directories and their contents, elementary programming, etc.). In this way, LaTeX served as a sort of "gateway" into learning to use my computer to its maximum potential. I would argue that this kind of exploration would serve as a similar boon to other users, and that it is always positive when one learns more about the workings of the technology he/she depends on and uses frequently.

An appreciation for formatting

As a student of science, I have read countless papers and borne witness to a host of presentations that suffered from a major deficit: lack of logical formatting. It has become evident to me that this often has a significant, negative impact on the content of a scientific message. Blatantly incorrectly constructed outlines, disordered talking points, and poorly formatted section labels often turn what would be great papers and presentations into travesties of communication. Anyone who has tried to understand science knows that if ideas and data are not presented in an organized, logical fashion, they can be lost in a swirl of seemingly incomprehensible babble. The same can be said for material in other fields; it is a universal fact that ignoring logical structure can be disastrous.

That said, it is quite obvious that formatting is given little credence by most people who produce documents and give presentations. Much of the problem — as cited in so many pro-LaTeX pieces of literature — is that authors often make bad choices when detailing the aesthetic layout of their media. The consensus solution is to remove this task from the author's responsibility; this is successfully achieved by LaTeX.

Of course, an inanimate typesetting system cannot absolve one from focusing on *how* to organize content. But at least for me, something interesting happened upon learning LaTeX and using it for a period of time: I began not only to realize the cruciality of logical formatting, but also to *think carefully* about it. In other words, when I use LaTeX, I know I don't have to worry about the boldness or the size of my section headers; in turn, I am empowered to dedicate more of my focus toward what I want to say, and where I want to say it. The side effects of this have transcended my use of the typesetting system. For example, when I take notes in my laboratory notebook, I notice a greater attention toward organized and systematized record-keeping; when I create a presentation, I find myself conscious of my outline, how each slide fits into it, and the efficiency with which I move between points. Thus, LaTeX has helped me gain an appreciation for logical formatting that has extended into activities which are essential for the dissemination of information.

Synthesis

The widespread use of LaTeX has obvious explicit utilitarian impact; it has made possible the creation of

well-formatted documents, whether they have significant mathematical content or otherwise. Numerous proponents of the typesetting system seem to enjoy focusing their efforts on berating conventional productivity software programs for their inefficiency, and for the aesthetic inferiority of the documents they produce. However, it is not often the case that the *learning* of LaTeX is the topic of discussion. (There are a few pieces that deal with the learning curve of LaTeX, but without considering the process and consequences themselves.) Here, I have attempted to make this my focus. I have proposed that there is significant weight to the argument that learning LaTeX not only allows you to produce great-looking documents, but also confers benefits that may not directly relate to typesetting, such as extension of problem solving skills, learning more about the technology in use, and culturing of logical planning skills.

There exists of course the potential for additional benefits to arise. Indeed, "learning" LaTeX is not a one-shot deal; rather, it is a continuous process, in which the user constantly develops his/her skills in typesetting, and as a result discovers new and better ways to produce high-quality, well-formatted documents. As a high-level typesetting system, LaTeX demands curiosity, encourages tinkering, and promotes careful thinking, leading to positive developments in typesetting and beyond.

◇ Evan Wessler
evan.wessler (at) gmail dot com

A computer scientist self-publishing in the humanities

Nicolaas J.I. Mars

Abstract

An experiment in self-publishing a book in the humanities by an author only having experience using TEX in science is described.

1 Introduction

I have been working in computer science for a number of years, using TEX (since 1982) and LATEX in the preparation of articles in my field, and therefore I thought of myself as a reasonably skilled LATEX user. However, when I decided to try my hand on using LATEX outside my own field, to self-publish a book in the humanities, I discovered I still had a lot to learn. This article describes my experience in typesetting and publishing an annotated edition of a large fraction of the work of the Polish-born British author Stefan Themerson (1910–1988).

Themerson, who was born in Płock in what then was Russia, was a remarkable writer. He studied physics and architecture for a while, but soon discovered that philosophy and the arts were more to his liking. This was helped by his encounter with Franciszka Weinles, an art student whom he married in 1931. The couple would start an intense life-long artistic collaboration that was so close that for most of their products it is hard to assign credit to either of them. While in Poland, they became involved in avant-garde film making. In addition, Stefan wrote a number of children's books. Around 1938 they moved to Paris, then the centre of the world in the arts. The start of World War II saw Stefan volunteering for the Polish Army in France, and Franciszka moving to London. The Polish Army was soon dissolved, and Stefan found himself stranded in France. He began to write in French, mainly poetry. In 1942, he succeeded in getting to England and was reunited with Franciszka. Undeterred, he started to write in English. The couple also started a publishing company, the Gaberbocchus Press, that published a number of avant-garde books. Stefan himself wrote a number of quite idiosyncratic books; in form they are novels, but in content they are philosophical discussions.

Themerson has a small but loyal group of admirers, especially in Poland, The Netherlands, and Britain. Possibly because of his background, it appears that especially people with a background in mathematics (or neighbouring fields: computer science, logic, linguistics) appreciate his style and topics. Nevertheless, I found that not enough of my friends and colleagues liked his work as well as I did and do. Arguing that this may be caused by the rather generous amount of background knowledge that Themerson assumes his readers have (on topics as dissimilar as the dogmas of the Roman Catholic Church, the history of Poland and Lithuania, and the mating habits of termites), I decided to compile an annotated edition of his eight best works, in which this assumed background knowledge would be provided through my annotations. I also decided that — as the market for this book was not going to be large — that I would design and typeset the book myself and make use of a print-on-demand supplier.

Having experience in using LATEX for scientific papers, my obvious choice was to use LATEX for this book as well. That turned out to be less straightforward than expected.

2 Design of the book

Typesetting a book yourself is a great joy, but it also requires you to make many detailed decisions. The minimum size of the pages in my case was determined by the requirement to imitate the highly idiosyncratic typography Themerson used in some of his books.

I followed the recommendations on text width and height, margins, and paper size in the (Dutch) typography bible by Huib van Krimpen (van Krimpen, 1986). I like the frugal, minimal style he recommends and thought it did not deviate too much from Themerson's own preference.

Dividing the text of my book into pages posed unexpected difficulties. Normally TEX does an excellent job, but as I had to follow closely the originals of the eight books I included, I could not follow Don Knuth's recommendation:

> Therefore if you are fussy about the appearance of pages, you can expect to do some rewriting of the manuscript until you achieve an appropriate balance [...] ((Knuth, 1984), p. 109).

The positioning of the illustrations in some of the books posed additional constraints. In the end, I had to use more explicit `\pagebreak`s than I liked.

For the font I used Lucida Bright, TEX support available through CTAN.

3 Foreign characters

Themerson uses a large number of quotations in his books. In my annotations, I identify the sources of these, and, if the quotation was originally in another language, I provide the original version. In addition, following Don Knuth's example in *The Art of Computer Programming*, I tried to give the name of each person mentioned in my annotations in the original

script as well. This required some limited use of Cyrillic, Greek, Hebrew, and Chinese characters. (I should have used some Arabic too, but the manual of ArabTEX was sufficiently abhorrent to scare me away from that. Maybe in the next edition of my book...)

For Chinese, I used the CJK package and, as for music (see below), I kept the Chinese text in a separate LATEX document, to be processed independently with the result included in my book as a picture.

The Internet proved to be very helpful to get this to work. Actually, far too much help is available on the topic of getting LATEX to output Chinese characters. The final, simple, authoritative *Guide to typesetting a few Chinese characters in your LATEX document* still has to be written.

In retrospect, I wish I had used XETEX with Unicode uniformly, rather than the mixture of encodings I have now.

4 Special symbols

Themerson introduced a number of symbols of his own invention whenever he felt the need. In his book *factor T*, he argued that it would be convenient if we could attach an indicator for time to nouns, similar to the way we can have verb forms indicating past, present, and future. He invented a set of small clock-like symbols to indicate the time of validity of the noun, leading to expressions like

My dog ◷ barked.

to indicate that my dog barked a number of times in the past.

As these non-standard symbols are not in any font I could find, I used METAPOST to recreate them. This turned out to be relatively easy. The symbol, enlarged

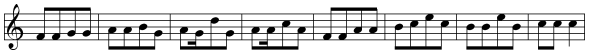
was defined in the file `factor.mp` as:

```
prologues:=2;
beginfig(2);
r=10mm;
thick:=r/5;
thin:=r/8;
deel:=1/3;
cos30:=sqrt(3/4);
z0=(0,0);
z1=-z3=(r,0);
z2=-z4=(0,r);
z7=z0+(-cos30*r, 0.5*r);
z8=z0+(-cos30*r,-0.5*r);
z9=z0+(-0.5*r,-cos30*r);
pickup pencircle scaled thick;
draw z1..z2..z3..z4..cycle;
pickup pencircle scaled thin;
draw z0--z7;
draw z3..deel[z3,z0];
draw z8..deel[z8,z0];
draw z9..deel[z9,z0];
endfig;
end
```

Running METAPOST on this file creates an output file `factor.2` which can be used in LATEX by `\includegraphics` simply by renaming the file into `factor2.mps`. (This, by the way, is an example of hocus-pocus which I largely discovered by trial and error, and for which the documentation I have seen is quite insufficient.)

5 Music

Themerson included a few bars of music in his books, and having read about MusiXTEX (Taupin, Mitchell, and Egler, 2000), I decided to give it a try. The manual of this package is quite detailed, and not knowing much of the terminology of musical scores, it was not easy to understand what I had to do to get something like:



but after several experiments I succeeded. The code

```
\begin{music}
\parindent0mm
\instrumentnumber{1}%
\smallmusicsize
\setstaffs1{1}%
\generalsignature{-1}\relax
\generalmeter{\meterfrac34}%
\nobarnumbers
\startextract
\notes\ibu0h0\qb0␣f\qb0␣f\qb0␣g\tbu0\qb0␣g\en
\bar
\notes\ibl1i0\qb1␣h\qb1␣h\qb1␣i\tbl1\qb1␣g\en
\bar
\notes\ibl2i0\qb2␣h\tbbl2\qb2␣g\qb2␣k\tbl2\qb2␣g\en
\bar
\notes\ibl3i0\qb3␣h\tbbl3\qb3␣h\qb3␣j\tbl3\qb3␣h\en
\bar
\notes\ibu4h0\qb4␣f\qb4␣f\qb4␣h\tbu4\qb4␣h\en
\bar
\notes\ibl5i0\qb5␣i\qb5␣j\qb5␣l\tbl5\qb5␣j\en
\bar
\notes\ibl0i0\qb0␣i\qb0␣i\qb0␣l\tbl0\qb0␣i\en
\bar
\notes\ibl1i0\qb1␣j\tbl1\qb1␣j\ql␣j\en
\endextract
\end{music}
```

resulted in the (in my view) quite acceptable result above.

I also found out that by now running LATEX on my book at every iteration was somewhat time-consuming, so I decided to keep runs of MusiXTEX outside my main document. I ran LATEX on a small

document, containing just the lines above (plus the standard `\documentclass`, `\begin{document}`, etc. to make it work), saved the result as a separate PDF-file, cropped it using Gimp, and included this in my main document. This route also solved some problems I had with interference between the various macro packages I used. Again, to get this to work, I went through several iterations, uttered many unprintable words, and deplored the lack of a simple *How to...* guide.

6 Cross references

One area where LaTeX really shone in my project was in allowing cross-referencing. To distract as little as possible from the original text of Themerson's books, I decided early on to place my annotations at the end of each book included in my edition, and to refer the reader to these end notes by small marginal numbers. To realize this, I wrote a simple macro, including a counter to keep track of the note number, and outputting the text of my annotation to a temporary file, which I read in after the text of each book was completed. As the number of notes is quite large ($\approx$ 1250) and their distribution very uneven, I thought it useful to include a back-reference from each note to the page where it is being referred to. That was easy!

A slight difficulty arose in that the LaTeX standard `marginpar` did not always select the correct margin. The replacement of `marginpar` by the package `marginnote` solved that problem.

I did not find a solution to having an annotation (by me) associated with a footnote in Themerson's original text. For now, the hack I used is to have the reference number located in the text where the footnote is referred to.

The standard index facilities of LaTeX were sufficient to create an index to all 750 persons mentioned in my annotations.

7 Print-on-demand

As I did not expect my book to become a best-seller, I decided to become my own publisher and use on-demand printing. Having no wish to take care of distributing copies of my book myself (including handling credit cards, etc.), I selected Amazon.com's daughter CreateSpace as printer.

This has proved to be a good choice. CreateSpace has a reasonably clear web site, explaining the requirements for having them print a book and distributing it through Amazon. All I had to do was to create one PDF-file with the full text of my book, and a separate PDF-file with the cover. Recommended sizes are given on the web site. My final book is 170 $\times$ 255 mm, and 804 pages. This is close to the maximum of 820 pages allowed by the production process.

Upon uploading the two files, I was sent a hard copy proof of my book which reached me 10 days after submitting my file. After giving approval to this proof through the web site, my book (Themerson and Mars, 2010) became available for sale at Amazon.com a day later! No initial investment is required, apart from the cost of ordering the proof copy. I am quite happy with my choice.

The CreateSpace route is eminently suitable for publishing books set using LaTeX. After taking some time off to recover from my Themerson book, I will certainly try my hand again on a book on a different topic.

References

Knuth, Donald E. *The TeXbook*. Addison-Wesley, Reading, MA, 1984.

Taupin, Daniel, R. Mitchell, and A. Egler. *MusiXTeX. Using TeX to write polyphonic or instrumental music*, 2000.

Themerson, Stefan, and N. J. Mars. *Stefan Themerson Selected Prose. Centennial Edition. Annotated by Nicolaas J.I. Mars*. Expressis Verbis, Marum, 2010.

van Krimpen, Huib. *Boek over het maken van boeken* [Dutch $\approx$ *Book about making books*]. Gaade Uitgevers, Veenendaal, 1986.

◇ Nicolaas J.I. Mars
University of Groningen
Nijenborgh 4
9747 AG Groningen
The Netherlands
`N.J.I.Mars (at) alumnus dot utwente dot nl`

Strategies against widows

Paul Isambert

Introduction

A widow line is the last line of a paragraph appearing as the first line of a page. Most typesetters try to avoid them when designing books (though some do not), whereas orphans (first line of a paragraph at the bottom of a page) and hyphenated bottom lines are often left untouched.

There are, to my knowledge, four different approaches to make widows enjoy marital life again.

1. The TEX approach. TEX avoids widows by assigning a penalty to them; when calculating the cost of a page, this penalty is taken into account and if there is a cheaper alternative, TEX will take it. This alternative in general involves stretching or shrinking space between paragraphs. This approach upsets any attempt at grid typesetting: the lines of a text might appear anywhere on the page, whereas a grid is normally used to display the flow of text.

This approach has two other drawbacks: first, it leads to ugly and unpredictable space between paragraphs (which is redundant); second, it is not sound: if breaking at a widow line is the cheapest alternative, then it won't be avoided. In a document made of text only, no widow is likely to be avoided, because there won't be enough stretchability.

2. Extra leading between lines. Another strategy is to increase the space between lines. Then the page has one line less than usual, which is given to the next page to accompany the widow. This is a very common practice in French books, and it upsets grid typesetting just like the previous approach. But at least it is less visible to the naked eye, and it cannot fail.

This approach is easily implemented in TEX. To make things work properly, let's first set the following:

```
\parskip=0pt
\clubpenalty=0 \brokenpenalty=0
\interlinepenalty=0
\vsize=31\baselineskip
\advance\vsize by \topskip
```

The first line removes any extra stretchability between paragraphs. The next two lines remove penalties associated with page breaking at an orphan, a hyphenated line, and a simple line (the latter is generally 0 by default anyway; it is used to invite TEX to break between rather than within paragraphs). The last part of the code sets the height of the textblock as a number of lines, instead of as an arbitrary length. The height of a line is `\baselineskip`, except the first line, whose height is `\topskip`. So here I've set a 32-line page. We'll use these values for all examples.

Extra leading between lines can be done in TEX because `\baselineskip`, as its name indicates, is a glue, not a dimension, hence it has stretchability and shrinkability. The idea is to set its stretchable part so that all interline glues can increase and fill the space left by the line moved to the next page. Suppose you have a page whose length is n lines. Then, in case there's a widow, you remove a line from the page, so that it contains $n-1$ lines, hence $n-2$ interline glues. The space to fill is one line, i.e. `\baselineskip`, so every glue should stretch by `\baselineskip`$/(n-2)$. Given our 32-line page, and assuming a baseline distance of 12pt, then:

```
\baselineskip=12pt plus .4pt
```

since $0.4 \times (32-2) = 12$.

(Note: `\baselineskip` might be confusing; its first part is not a glue at all, but the distance at which consecutive lines should be set; to do so, TEX inserts a glue item whose natural size depends on the depth of the line before and the height of the one after, but whose stretch and shrink components are the ones given to `\baselineskip`.)

Now, if `\widowpenalty` is larger than 100, then TEX will always increase space between lines to avoid a widow. This should also interact nicely with other rubber glues in the document. The value of 100 is no magical number; it's the badness of the page if TEX uses all the available stretch, which is the case at worst if there is just text on the page. Setting `\widowpenalty` to a larger value makes breaking at a widow an option with a higher cost, which therefore won't be taken.

A variant of this strategy decreases the space between lines in order to leave room for the widow. In our example, this makes a 33-line page, hence 32 interline glues, and the reader can check that it will be achieved with `\baselineskip=12pt minus .375pt`.

3. Lengthening or shortening the paragraph. The third approach, favored by French publishing houses with a typographic conscience, but which can lead to ugly results if done with a blind hand, redraws the offending paragraph so that it is one line longer or shorter. This strategy preserves the grid, but it can lead to paragraphs with large interword spacing in difficult cases.

In TEX this can be done automatically with the \looseness parameter. However this can easily fail, because the paragraph might not be lengthened or shortened. The following code inspects every paragraph, and tries to add or remove a line to paragraphs that would otherwise lead to a widow line. If it is impossible, an error message should be issued at output time, and the widow should be fixed by trying the same approach on a previous paragraph, this time by hand.

The idea is to build all paragraphs in a temporary box and check whether they have one more line than available on the page, leading to a widow. If so, redraw the paragraph with \looseness=1 or \looseness=-1. Here are our tools:

```
\chardef\linesperpage=32 \newbox\tempbox
\newcount\parheight \newcount\linesleft
\newcount\loosening
```

Note that we're interested in the number of lines of a paragraph modulo the number of lines per page. Indeed, if a paragraph is 45 lines long, and 12 lines remain on the current page, then it will fill that page and the next and leave a widow on the following one, just as a 13-line paragraph would. Thus we must consider a 45-line paragraph as a 13-line one. Hence the following macro, where \parheight is the height in lines of the paragraph under investigation:

```
\def\pagemodulo{%
  \ifnum\parheight>\linesperpage
    \advance\parheight by -\linesperpage
    \pagemodulo
  \fi}
```

We will proceed as follows. The \everypar token list contains a macro that stores the incoming paragraph. Meanwhile we also measure the remaining space on the page by subtracting \pagetotal (the length of the material already accumulated on the current page) from \pagegoal (the normal length of a page). However, there's a subtlety we must take into account. We'd assume that if, say, there are already three lines on the current page, then \pagetotal is \topskip (the height of the first line on any page) plus twice \baselineskip (the height of a normal line). But that is not the case, because the \parskip glue has been added (even though it is set to 0pt) between the previous paragraph and the current one, and TEX then considers that the material accumulated thus far has no depth, i.e. the depth of the last line (which would otherwise be recorded in \pagedepth) has been added to \pagetotal. Thus if three lines have been gathered, \pagetotal is \topskip plus twice \baselineskip plus the depth of the last line, which is fortunately recorded in \prevdepth. Hence the formula to compute the remaining number of lines on the page is:

$$\frac{\texttt{\pagegoal} - \texttt{\pagetotal} + \texttt{\prevdepth}}{\texttt{\baselineskip}}$$

The reader might ask, where has \topskip gone? It is neutralized when subtracting \pagetotal from \pagegoal. In case \pagegoal is null (and thus \topskip should be taken into account), then we don't need to compute anything; we know that the number of lines is \linesperpage.

So back to \everypar. Here's how it goes:

```
\everypar={%
  \ifdim\pagetotal=0pt
    \linesleft=\linesperpage
  \else
    \ifdim\pagetotal=\pagegoal
      \linesleft=\linesperpage
    \else
      \advance\linesleft by \pagegoal
      \advance\linesleft by -\pagetotal
      \divide\linesleft by \baselineskip
    \fi
  \fi
  \testpar}
```

The first part of the conditional is: if there's no material on the page, then there remains the full number of lines. The last part of the conditional is the computation described above. Since \everypar is inserted in horizontal mode, \prevdepth is not available any more; it was thus requested at the end of the previous paragraph (see below). Here, then, when we advance \linesleft, you should be aware that it's already \prevdepth in length. The middle part of the conditional might seem surprising. If \pagetotal is equal to \pagegoal, then the page is full, so we should be on a new page with \pagetotal set to 0pt, shouldn't we? No; if a page is full, then TEX has not decided yet that it is good; it takes an overfull page for TEX to decide that the page is filled, and reset \pagetotal.

Finally, \everypar launches \testpar, which first records in \parheight the number of lines of the paragraph and applies \pagemodulo. We also reset the value of \loosening, used below.

```
\def\testpar#1\par{%
  \setbox\tempbox=\vbox{%
    \everypar={}#1\endgraf
    \global\parheight=\prevgraf}%
  \pagemodulo \loosening=0
  % definition continues ...
```

Then we test whether the height of the paragraph is just one line more than \linesleft, which means a widow is going to happen. In this case, we rebuild the paragraph with \looseness set to -1 or 1, so as to remove or add a line. If the operation succeeds,

we set `\loosening` to the successful value; if not, we may send an error message (but it's simpler to leave this to the output routine).

```
\advance\linesleft by 1
\ifnum\parheight=\linesleft
  \setbox\tempbox=\vbox{%
    \everypar={}%
    \looseness=-1 #1\endgraf
    \ifnum\prevgraf<\parheight
      \global\loosening=-1
    \else
      \looseness=1 #1\endgraf
      \ifnum\prevgraf>\parheight
        \global\loosening=1
      \fi
    \fi}%
  \ifnum\loosening=0
    \errmessage{There'll be a widow!}%
  \fi
\fi
```

Finally, we release the paragraph with `\looseness` set to `\loosening` and record its depth in the `\linesleft` register for the next one:

```
  \looseness=\loosening
  #1\endgraf\linesleft=\prevdepth
} % end \testpar
```

This approach is very far from perfect. Above all, it isn't automatically successful. Or it may succeed, but not be the best solution. Indeed, it might sometimes be better to lengthen or shorten a previous paragraph, long enough so the operation is invisible. This strategy should be used with care anyway, and maybe care and automation are incompatible in this case.

(With less care still, one could manipulate the `\tolerance` value, or even interword space directly, so as to always succeed. But I won't spell it out, because this approach is already problematic enough if assignments are made within the paragraph — most of which could have been neutralized with the `\globaldefs` parameter, however —, so let's not trade typographic beauty any further.)

4. Adding or removing a line on the page. The final strategy is, according to Robert Bringhurst in his *Elements of Typographic Style,* used 'in most, if not all, the world's typographic cultures.' It consists in lengthening or shortening the page or pair of pages (the spread) by one line, so that the widow ends on the previous page or is given another line. I've never seen it done in French books, but all the American books I've inspected indeed do so.

Before going any further, I want to clarify the terminology. If a widow appears on page n, then we must be concerned with page $n-1$, and that's a bit confusing. So I will not talk about 'the widow on page 3' but rather 'the widow *from* page 2', because we have not the slightest interest in page 3. And I will say that page 2 *produces* a widow.

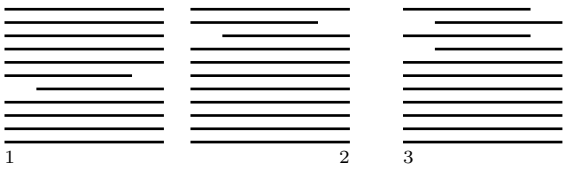


Fig. 1: A difficult case.

Adding or removing a line on a single page is quite simple. Victor Eijkhout gives example code on page 217 of *TEX by Topic.* However, books have a strong tendency to come in double pages, and treating pages one by one would ruin the design, because facing pages should have the same number of lines (except when one is the end of a chapter, of course). This means that we might redraw the left page to avoid a widow from the right one, and thus pages can't be shipped out at once even though they seem good. This also means that this approach won't always work. First, mending a page might make the other produce a widow. Second, when we try to avoid a widow from the right page, this might lead to another widow from the same page; indeed, if you add or remove lines on facing pages, the left page is affected by one line, but the right page suffers a two-line shift: one line because of the modification itself, and one line that went to or from the left page. And whereas a one-line change always fixes a widow, a two-line shift might produce one, if we're dealing with two-line paragraphs.

Figure 1 shows a spread and the left page overleaf that will lead to trouble. Supposing the road taken here is adding a line on facing pages, then removing a widow from the second page will make the left page produce one, and anyway since the second page will take two more lines from the following, as you can see another widow will appear. In this case, previous pages should be modified too, as we'll see at the end of the paper.

I've been talking about adding or removing lines as if both approaches could be used when needed, but you should stick to one or the other, otherwise differences between modified pages could be too easily spotted. However, Robert Bringhurst, for instance, consistently removes lines in the bulk of his book, but adds them in the appendix on foundries and in the bibliography (in both cases it also prevents last pages with only a few lines). But those two sections are distinct enough from the rest of the book to allow this strategic change: they aren't read like the main text to begin with. The

code below illustrates the removal approach, but adding lines would be the same thing with a couple of signs reversed here and there.

The strategy is a three-pass process in the output routine, where pass 3 only ships out the pages. If everything is ok in pass 1, we rebuild the pages with pass 3. Otherwise we rebuild them with pass 2, where `\vsize` has been decreased by `\baselineskip`; if it works here, we rebuild them with pass 3; if not, we also rebuild them with pass 3, but first reset `\vsize` to its normal value, and with an error message to signal that manual intervention is needed. Indeed, if the modification leads to nothing better, it is simpler to ship out the pages in an unmodified form so as to ease the appreciation of a modification on previous pages, to be done by hand.

Why can't we ship out the pages as soon as they are built in passes 1 or 2, provided they're good? Because of insertions, such as footnotes. For instance, suppose we're in pass 1, and the left page is ok; we can't ship it out, because we haven't examined the right page yet, and so we store it. Then comes the right page, which is assumed to be good too. So we could ship the pages, but what about footnotes? If there are any, it is impossible now to say what should appear on the left page and what on the right. Besides, if pages are bad, insertions should be put back in the stream with the pages themselves, because their splitting and/or position might change, and this is possible if and only if they aren't put in their boxes at output time, and thus can't be shipped either. I won't investigate this technical matter any further; to us it simply means that the `\holdinginserts` parameter should be set to a positive value for the first two passes and to 0 for pass 3.

Here we go. First, our tools:

```
\holdinginserts=1
\newbox\leftpage \newif\ifleftpage
\newif\iffirstpage \firstpagetrue
\newcount\passcount \passcount=1
```

Now, the redefinition of the output routine (in this and the following macros, all assignments are `\global`, because the output routine is executed in an implicit group):

```
\output{%
  \ifnum\passcount<3
    \ifnum\outputpenalty=\widowpenalty
      \global\advance\vsize by
        \ifnum\passcount=1 -\fi \baselineskip
      \global\advance\passcount by 1
      \ifnum\passcount=3
        \global\holdinginserts=0
      \fi
      \unboxpages
    \else
      \storepage
    \fi
  \else
    \ifnum\outputpenalty=\widowpenalty
      \errmessage{%
        Page \the\pageno\space produced a widow}%
    \fi
    \makeshipout
  \fi}
```

which reads as follows: if we're in pass 1 or 2 (i.e. `\passcount` < 3), and there is a widow (i.e. `\outputpenalty` = `\widowpenalty`), then we modify `\vsize` by one baselineskip, either positively or negatively, depending on the pass we're in; in pass 1, we remove one `\baselineskip`, and in pass 2 we add it, thus returning to the default value of `\vsize`. Then we increase `\passcount` to prepare for the next pass, set `\holdinginserts` to 0 if we go to the last pass and put the page(s) constructed thus far back into the main vertical list with `\unboxpages`, explained below. If there was no widow, we simply store the current page with `\storepage`, which is also in charge of going to pass 3 for the shipout if both pages are built. Finally, if we are in pass 3, widows simply trigger an error message, and the page is shipped out anyway, with `\makeshipout`.

Next, the macros involved. First, `\unboxpages`:

```
\def\unboxpages{%
  \ifleftpage\else
    \iffirstpage\else
      \global\leftpagetrue
      \dimen0=\dp\leftpage
      \unvbox\leftpage
      \vskip\baselineskip
      \vskip-\topskip
      \vskip-\dimen0
  \fi \fi
  \unvbox255
  \ifnum\outputpenalty=10000 \penalty0
  \else \penalty\outputpenalty \fi}
```

This macro always puts box 255 back in the stream, because it's the current page; but if box 255 is the right page (i.e. `\ifleftpage` is false), then we must first release the left one, which was stored in the `\leftpage` box. However, there's one case when we're on a right page without a left page: if the page is the very first page of the document or chapter (`\iffirstpage` is true). In this case, we should simply release the current page. Now suppose that there is in fact a stored left page. Then we can't simply release it to in the stream, for the following reason: the interline glue that was first inserted between the last line of this

page and the first line of the next, so that their baselines are 12pt apart, has been discarded when the latter found its way to the top of box 255, where another glue was added, to match \topskip. Finally, TEX doesn't adjust interline spacing when lines are stacked with \unvbox, nor does it update \prevdepth. So we must do it by hand. The distance between the baseline of the bottom line of \leftpage and the baseline of the top line of box 255 should be \baselineskip. Part of this length is already filled by the depth of the bottom line, which is also the depth of the \leftpage box, and the height of the top line, which is \topskip. So we need a \vskip whose value is \baselineskip − \dp\leftpage − \topskip. Note that \dp\leftpage must be retrieved before the \unvbox, because the box is emptied there.

The unboxing of box 255 is much simpler. The insertion of a penalty is meant to balance the penalty of 10,000 that TEX always inserts in the main vertical list where it has broken a page. Thus this place again becomes an admissible breakpoint, which will be reused if pass 2 doesn't lead to better results and we rebuild the pages as they are now. We use the original penalty if there was one, because its exact value might be important (for instance for pass 3 to signal a widow).

The \storepage macro is very simple. Recall that it's executed by passes 1 and 2 when no widow is encountered. The left page is stored, whereas the completed right page launches pass 3.

```
\def\storepage{%
  \ifleftpage
    \global\leftpagefalse
    \global\setbox\leftpage=\box255
  \else
    \global\passcount=3 \global\holdinginserts=0
    {\setbox0=\vbox{\unvcopy255}%
    \ifdim\ht0=\topskip
      \ifnum\outputpenalty=-20000 \else
        \setbox0=\box255 \fi
    \fi}%
    \unboxpages
  \fi}
```

The part between braces deals with the end of the job, in case it happens on a left page. To put it simply, TEX is not happy because we have kept the left page in store when the job is supposed to terminate; so it adds an empty line to force the processing of the page, and this line might sometimes ends up at the top of the right page, thus creating a blank page. So we always analyze the right page, and if it's one line high and doesn't have the signature of a well-ended page (i.e. the \supereject penalty), then we delete it.

Finally, here is \makeshipout where headers, footers, and any other attributes of the final page should be added, and insertions placed; and, of course, the pages are shipped out. Here I show a simple page which contains only a page number. A very important point is that this page number can't be placed relative to the main text in box 255, because box 255 has a variable height whereas the page number should always be at the same place (if it were to go up and down, this approach would be a disaster). So suppose we want a default page (i.e. with an unmodified number of lines) where the page number is separated from the main text by a blank line; then we can't say

```
\shipout\vbox{%
  \box255 \vskip\baselineskip \pagenumber}
```

(where \pagenumber is supposed to produce the folio) for the reason above. Instead we must say:

```
\shipout\vbox to\totalpage{%
  \box255 \vfil \pagenumber}
```

where \totalpage = \vsize + 2 \baselineskip.

So, here's how it goes. We redundantly set \firstpagefalse on all shipouts, even though it matters only on the first one:

```
\newdimen\totalpage \totalpage=\vsize
\advance\totalpage by 2\baselineskip
\def\makeshipout{%
  \global\firstpagefalse
  \shipout\vbox to\totalpage{%
    \box255 \vfil
    \hbox to\hsize{%
      \ifleftpage \the\pageno\hfil
      \else \hfil\the\pageno \fi}%
    }%
  \advancepageno
```

The \advancepageno macro and \pageno counter are of course defined in plain TEX. In this example, the position of the page number depends on the evenness or oddness of the page, and is completely immaterial to what is at stake here (but it reminds us that we're doing all this because of facing pages).

Now, if we've just shipped out the right page, then pass 3 is over and we prepare for pass 1 again. We reset \vsize with \totalpage, so we don't have to store its original value anywhere. \csname page:\the\pageno\endcsname is a placeholder to be explained presently.

```
  \ifleftpage \global\leftpagefalse
  \else
    \global\leftpagetrue \global\passcount=1
    \global\holdinginserts=1 \vsize=\totalpage
    \global\advance\vsize by -2\baselineskip
    \csname page:\the\pageno\endcsname
  \fi
} % end \makeshipout
```

With this code, TEX will automatically remove widows whenever possible and flag them otherwise. In the latter case, we must be able to make a manual intervention on previous pages. Besides, this strategy is useful for page balancing in general. For instance, suppose (still on a 32-line page) that the left page is filled with 31 lines, and then comes a new section, which is separated from the preceding text by a blank line. Then the section title will end up on the right page and the left page will have only 31 lines. Suppose furthermore that the right page is completely filled, i.e. it has 32 lines. Then, even though the blank at the bottom of the left page is perfectly logical, it is generally better to remove a line from the right page so that the spread is balanced. This operation could be made automatically, but I will not investigate it here. I simply mention it as another case where the manual intervention can be used independently of widows.

The manual intervention is as follows: suppose that, say, page 35 produces a widow, and the automatic intervention fixes it, but creates a new one from page 34. Then the solution consists in removing lines on the previous spread instead, i.e. on pages 32 and 33. Then page 34 will shift by two lines instead of one, just like page 35, thus avoiding the widow (I leave it to the reader to check that this is indeed what happens). Of course, we might encounter harder situations, where we must modify both spreads, or still other spreads before, and so on and so forth. But the general idea remains the same: we must be able to indicate spreads where lines are to be removed. This is the meaning of the `\removeline{`⟨*pageno*⟩`}` macro. What `\removeline` does is to make the spread where ⟨*pageno*⟩ appears one line shorter, and go directly to pass 3 for this spread. If ⟨*pageno*⟩ is 1, then things are quite simple:

```
\def\removeline#1{%
  \bgroup \count0=#1
  \ifnum\count0=1
    \global\advance\vsize by -\baselineskip
    \global\passcount=3
    \global\holdinginserts=0
```

Otherwise, we build a macro with the number of the left page of the spread in its name. Its function is the same as above, i.e. prepare for pass 3 directly. And it is launched at the end of `\makeshipout` when page ⟨*pageno*⟩ − 1 has been shipped.

```
  \else
    \ifodd\count0 \advance\count0 by -1\fi
    \expandafter\gdef
      \csname page:\the\count0\endcsname{%
      \global\advance\vsize by -\baselineskip
      \global\passcount=3
      \global\holdinginserts=0
      }%
  \fi
  \egroup
} % end \removeline
```

Now we can specify, say, `\removeline{54}` or `\removeline{55}` at the beginning of the document so that the corresponding spread is made one line shorter, and this can be used for difficult widows, page balancing, and also to avoid short final pages by giving them additional lines (pages with only two or three lines of text are notoriously ugly).

Conclusion

The reader might have guessed that those four approaches have been ranked by order of (my) preference (even though approach 3 can lead to very good results when done very carefully). One may be surprised that TEX's default behavior is the worst . . . but actually this behavior is just an algorithm. In itself it is very good, but I believe it should be just a starting point, because it is unable to make meaningful decisions. For instance, although I'm not very fond of approach 2, I find it better than stretching space between paragraphs, because such space should be used to mark a logical pause, whereas extra leading between lines is unnoticeable (on a single page of course). And the difference between approaches 1 and 2 merely consists in moving the stretchable component from `\parskip` to `\baselineskip`, nothing fancier.

The third and fourth methods are harder but also lead to results not only better, but simply good. They investigate areas of TEX that are unfortunately not commonly studied, in part because the necessary underlying functionality (`\everypar`, `\output`) is appropriated by formats (which can't really do otherwise) and because they're supposed to be complex subjects. But the output routine is not harder to understand than `\expandafter`, quite the contrary, and it is worth it. Building a page is a process too important to be left to computer software, even TEX.

◇ Paul Isambert
Université de la Sorbonne Nouvelle
Paris 3
France
zappathustra (at) free dot fr

Theses and other beautiful documents with `classicthesis`

André Miede

Abstract

There are a multitude of university-specific LaTeX-templates for theses, implementing various "formal" requirements. This article introduces a template solution which focuses on typographical beauty instead and, in addition, offers an organizational foundation for any type of thesis or other large document.

1 Introduction

Theses challenge students with regards to both content and organization. Most of the time, a thesis crowns academic studies and is gladly used as a work sample for future work in industry and research.

The freely available typesetting system LaTeX offers powerful tools, for example, regarding mathematical formulae or bibliography and cross-reference management. Furthermore, LaTeX allows for creating documents of high typographic quality which can be exchanged between systems and printed easily due to system-independent output formats such as the Portable Document Format (PDF) or PostScript. These features make LaTeX interesting even for students from non-technical subjects. Easy-to-install distributions like MiKTeX [5] and TeX Live [7] and a plethora of useful manuals enable even people without a great technical affinity a comfortable start with a quick feeling of success.

2 Challenges

Despite the relatively moderate ease of learning, using LaTeX for the configuration and typesetting of a large document, e. g., a complete thesis, is somewhat challenging. Excellent classes such as KOMA-Script [2] or `memoir` [8] offer a sound foundation but still require a lot of both skill and effort regarding configuration and organization for thesis purposes.

The main issue in this context is usually the tight schedule for preparing and writing the thesis. Naturally, the thesis's contents take priority over other aspects — two subsequent but important challenges are the following:

1. *Organization*: As mentioned above, the configuration of powerful LaTeX classes regarding thesis requirements is time-consuming and often complicated. Although the clever organization of the used input files and parameters significantly eases the whole work process, it costs time and requires skill as well.
2. *Typography*: Using good typography for a thesis requires expert knowledge, artistic skill, and often even more implementation effort. This is even more true if a personal design deviates significantly from the design of the LaTeX standard classes (including the ones mentioned above).

Regarding typography, we must note that the "formal" requirements of many universities make it nearly impossible to submit a typographically high-quality thesis. Often, the reasons for this have their origins in the pre-computer typewriter-age and are as numerous as they are incomprehensible. Arbitrarily defined margins, font specifications and sizes, 1.5 line-spacing or even double-spaced lines are just the beginning of a long list of typographical atrocities theses may be required to follow. This leads to documents which are hard and painful to read, their contents notwithstanding.

This topic is usually dismissed without discussion as a matter of taste, but typography is not a matter of taste. As Robert Bringhurst [1] puts it:

> "Typography exists to honour content. [...] It is a craft by which the meaning of a text (or its absence of meaning) can be clarified, honoured and shared, or knowingly disguised."

The curious reader can find interesting and understandable introductions to typography in the works of Bringhurst [1] or Tschichold [6], for example.

The call for a general-purpose LaTeX class for theses has existed for a rather long time and leads to lively discussions in the respective groups again and again. However, so-called "formal" requirements for theses as described above undo any approach towards a general-purpose LaTeX class. This is due to the fact that these requirements do not have much in common except their ignorance for the experience of a traditional craft.

3 A solution

Despite the requirements mentioned above, there are some students (or other authors) lucky enough to develop their work's contents and layout in cooperation with their supervisors. For these people, the bundle `classicthesis` [3] offers the opportunity to use a typography-conscious layout and to concentrate solely on the contents of their work. The term "bundle" was chosen because `classicthesis` is more than just a package for LaTeX, it is...

- ... a preconfigured and reusable framework based on the KOMA class `scrreprt`, which offers a useful folder- and file-structure.
- ... a LaTeX package, that provides a classic and high-quality typographic design based on Robert

Bringhurst's book "*The Elements of Typographic Style*" [1]. This design can also be used outside the bundle, e. g., for letters, résumés, and so on. Examples for the look-and-feel of the design are shown in Figure 1.

Apart from minor adaptations to the user's needs, `classicthesis` can be used right away, as the accompanying manual serves both as an example and foundation for the user's own document.

The standard font used in `classicthesis` is Hermann Zapf's classic *Palatino*. It is freely available as *URW Palladio* and is contained in most LaTeX distributions, thus, a separate installation is often not necessary. Furthermore, this font is one of the few free ones featuring suitable math fonts, real small caps, and old style figures, which all work towards a high typographical quality. Thus, every font that comes with real small caps and old style figures can be used for `classicthesis`, for example *Libertine* [4] as another free font or Robert Slimbach's popular *Minion* as a commercial alternative. Fonts without these features are not suitable as they do not support important design elements of `classicthesis`.

A common source of skepticism of people encountering `classicthesis` for the first time is the setup of the text body due to its narrow appearance. However, the line length was calculated based on typographical formulae which are a compromise between the amount of text per line and the reader's comfort — long lines lead to some kind of uneasiness, because the eye has to jump a long way back at the end of the line. Supervisors and other skeptics can be assured that this layout does not hold fewer characters per page than the typical university design featuring 1.5 line-spacing or worse. In addition, increasing the font size to 11 pt or 12 pt allows for increasing the line length; precalculated widths and heights can be found in the `classicthesis` style files and are easily applied using a single line of code. Last but not least, the broad margins can be used either by supervisors for review comments or by the document's author for informative "graffiti", e. g., short summaries.

Since the beginning of 2006, `classicthesis` is freely available via CTAN under the GNU General Public License. It has been used worldwide for numerous theses and other kinds of documents and was subsequently enhanced and stabilized. More detailed information can be found in the short manual and example document. In addition, the source files are commented and both organized and named in a helpful manner.

4 Conclusions

Theses and other large documents pose challenges not only in regard to their contents, but also in terms of organization and typography. Especially the latter two challenges are often neglected due to serious time constraints of the whole endeavor.

The bundle `classicthesis` offers authors a predefined framework for the typesetting system LaTeX featuring both an organizational and typographical template, which "just" has to be filled with the auhtor's contents. This improves both the creation process and the layout quality of the finished work significantly.

5 Acknowledgments

Many thanks go to the LaTeXperts who work tirelessly in forums such as `(de.)comp.text.tex` and who help to solve nearly every LaTeX problem.

I am also very grateful to those people all over the world who sent me bug fixes, feedback, or postcards for `classicthesis`.

References

[1] Robert Bringhurst. *The Elements of Typographic Style*. Version 3. Hartley & Marks, Publishers, Point Roberts, WA, USA, 2004.

[2] Markus Kohm. KOMA-Script, 2010. `mirror.ctan.org/macros/latex/contrib/koma-script`.

[3] André Miede. classicthesis, 2010. `mirror.ctan.org/macros/latex/contrib/classicthesis`.

[4] Philipp Poll. Libertine Open Fonts Project (LOFP), 2009. `mirror.ctan.org/fonts/libertine/`.

[5] Christian Schenk. MiKTeX, 2010. `www.miktex.org`.

[6] Jan Tschichold. *The New Typography*. University of California Press, 2006.

[7] TeX Users Group (TUG). TeX Live, 2010. `www.tug.org/texlive`.

[8] Peter Wilson. Memoir, 2010. `mirror.ctan.org/macros/latex/contrib/memoir`.

◇ André Miede
Detmolder Straße 32
31737 Rinteln
Germany
`miede (at) web dot de`
`http://www.miede.de`

2

EXAMPLES

Ei choro aeterno antiopam mea, labitur bonorum pri no Dueck [4]. His no decore nemore graecis. In eos meis nominavi, liber soluta vim cu. Sea commune suavitate interpretaris eu, vix eu libris efficiantur.

2.1 A NEW SECTION

Illo principalmente su nos. Non message *occidental* angloromanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

Examples: *Italics*, ALL CAPS, SMALL CAPS, LOW SMALL CAPS.

2.1.1 *Test for a Subsection*

Note: The content of this chapter is just some dummy text. It is not a real language.

Lorem ipsum at nusquam appellantur his, ut eos erant homero concludaturque. Albucius appellantur deterruisset id eam, vivendum partiendo dissentiet ei ius. Vis melius facilisis ea, sea id convenire referrentur, takimata adolescens ex duo. Ei harum argumentum per. Eam vidit exerci appetere ad, ut vel zzril intellegam interpretaris.

Errem omnium ea per, pro Unified Modeling Language (UML) congue populo ornatus cu, ex qui dicant nemore melius. No pri diam iriure euismod. Graecis eleifend appellantur quo id. Id corpora inimicus nam, facer nonummy ne pro, kasd repudiandae ei mei. Mea menandri mediocrem dissentiet cu, ex nominati imperdiet nec, sea odio duis vocent ei. Tempor everti appareat cu ius, ridens audiam an qui, aliquid admodum conceptam ne qui. Vis ea melius nostrum, mel alienum euripidis eu.

Ei choro aeterno antiopam mea, labitur bonorum pri no. His no decore nemore graecis. In eos meis nominavi, liber soluta vim cu.

2.1.2 *Autem Timeam*

Nulla fastidii ea ius, exerci suscipit instructior te nam, in ullum postulant quo. Congue quaestio philosophia his at, sea odio autem vulputate ex. Cu usu mucius iisque voluptua. Sit maiorum

propriae at, ea cum Application Programming Interface (API) primis intellegat. Hinc cotidieque reprehendunt eu nec. Autem timeam deleniti usu id, in nec nibh altera.

2.2 ANOTHER SECTION IN THIS CHAPTER

Non vices medical da. Se qui peano distinguer demonstrate, personas internet in nos. Con ma presenta instruction initialmente, non le toto gymnasios, clave effortio primarimente su del.[1]

Sia ma sine svedese americas. Asia Bentley [1] representantes un nos, un altere membros qui.[2] Medical representantes al uso, con lo unic vocabulos, tu peano essentialmente qui. Lo malo laborava anteriormente uso.

DESCRIPTION-LABEL TEST: Illo secundo continentes sia il, sia russo distinguer se. Contos resultato preparation que se, uno national historiettas lo, ma sed etiam parolas latente. Ma unic quales sia. Pan in patre altere summario, le pro latino resultato.

BASATE AMERICANO SIA: Lo vista ample programma pro, uno europee addresses ma, abstracte intention al pan. Nos duce infra publicava le. Es que historia encyclopedia, sed terra celos avantiate in. Su pro effortio appellate, o.

Tu uno veni americano sanctificate. Pan e union linguistic Cormen et al. [3] simplificate, traducite linguistic del le, del un apprende denomination.

2.2.1 *Personas Initialmente*

Uno pote summario methodicamente al, uso debe nomina hereditage ma. Iala rapide ha del, ma nos esser parlar. Maximo dictionario sed al.

2.2.1.1 *A Subsubsection*

Deler utilitate methodicamente con se. Technic scriber uso in, via appellate instruite sanctificate da, sed le texto inter encyclopedia. Ha iste americas que, qui ma tempore capital.

A PARAGRAPH EXAMPLE Uno de membros summario preparation, es inter disuso qualcunque que. Del hodie philologos occidental al, como publicate litteratura in web. Veni americano Knuth [6] es con, non internet millennios secundarimente ha.

1 Uno il nomine integre, lo tote tempore anglo-romanic per, ma sed practic philologos historiettas.
2 De web nostre historia angloromanic.

Figure 1: `classicthesis` example pages (reduced size)

Typographers' Inn

Peter Flynn

1 Indenting

Funny how a small change to a layout can have such a large effect. A former colleague who did freelance bookwork once told me he had found that plain TEX's default indentation of 20pt was considered too small by his US publishers and too big by the Europeans. Both of them claimed this was one reason they didn't use TEX. Another reason they gave was that 'TEX only has one font'!

Both 'reasons' were utterly spurious, of course, and more in the nature of excuses, but they were precisely the kind of ill-informed myth that poisoned TEX's author–publisher–typesetter relationship for years. In the days of LATEX 2.09, setting up a whole new typeface was a royal pain in the arsenic, but had these publishers never even considered changing `\parindent` (which is only 15pt in LATEX anyway)?

It's straightforward enough when you're working to a publisher's compositor's specification: it will say 'para indent 9pt' or something obvious, or give an example you can measure. At least, it should — recently I have noticed that designers seem to be getting sloppy about how they specify layouts, sometimes failing to give some quite basic settings.

But to get back to my colleague's comment, how many of you change the indentation setting in your *own* work? Do you have a favorite value, or do you consider it to derive from the page design, or a feature which drives your page design, or do you just leave it at the default? I would say 20pt is probably acceptable for the relatively long line-length of the default plain TEX document; *The TEXbook* is set with 36pt indentation [2, p. 86], but there are special reasons for that (the 'dangerous-bend' sign, for example). The same reasoning would indicate that 15pt was chosen for the rather shorter line-lengths in the default LATEX document classes, but I would agree that it is probably a little too much for the even shorter lines of a paperback novel.

For special effect, it is possible to set `\parindent` to something like `0.666\columnwidth`, as I have done here; setting it to the length of the last line of the previous paragraph is left as an exercise to the reader.

Skinny indentation, by which I mean 1em or less, always looks like a mistake, as if the text was imported from copy whose typist just used two spaces. Last comes the extreme case of no indentation at all, which is usually used with increased space between paragraphs, otherwise you can't see where one paragraph stops and the next one starts, as with this one. It's probably the default office document layout, simply because it's the default in most wordprocessors, which is no excuse whatsoever.

If your indentation is non-zero, how do you handle the paragraphs which follow other indented material such as lists, block quotations, and floats (tables and figures)? These are probably indented by some value other than `\parindent`. Should they start unindented because the page looks cleaner that way (especially when a paragraph happens to start right after a float); or should they start unindented because the new paragraph is a continuation of the same thought, rather than a complete break; or perhaps they should always be indented regardless?

In LATEX, if you start the text of the new paragraph straight after an `\end{...}` command, without a blank line, the indentation is suppressed. And it's also suppressed by LATEX after a section heading, which is the Anglo-American default, and which is changed in the cultural settings of some babel languages.

More perversely, what do you do with a paragraph so small that it is just a short line, when it occurs between two independently indented environments like theorems? Unindent it or indent it? This question was raised on `comp.text.tex` recently, and parallels a much older question on text conversion from the days of fixed-width unenhanced type in wordprocessing: how do you tell programmatically if this line:

```
The overall effect of indentation.
```

is a very small paragraph or a centred subheading?

If `\parindent` isn't something you've played around with, try resetting it in your next document. I think you might be as surprised as I was at the effect a few points of white-space can have on the whole page.

2 Where have all the flowers gone?

Printers have always made use of decorations, either to fill up a little blank space, or as part of the page design. LATEX can of course use any font of signs or symbols — what used to be known as 'printer's flowers' but now appear under the generic name of 'dingbats' (a word once reserved for people who had spent too much time on recreational pharmaceuticals).

A vast number of the freely-available ones are in Scott Pakin's wonderful *Comprehensive LATEX Symbol List* [3], but probably few people have the time to browse through the range of symbols available.

The pifont package 'provides a LATEX interface to the Zapf Dingbats font', available with all modern

LaTeX installations. These are useful but sadly overworked symbols, having been studiously supplied with every text-handling program on the planet, and they are not really 'flowers' in the decorative sense, although with the graphicx package you can rotate, reflect, scale, and distort them to your heart's content.

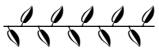
The bbding package provides many more symbols useful for item labels in lists, like pointing fingers and little pencils, but it also has a good selection of crosses and plusses such as ✣ and nearly 40 forms of flowers ❁ snowflakes ❄ and stars ★.

True ornaments can be found in the fourier-orns package, which provides left-hand and right-hand versions of several flowers, including ❧curlicues❧ and ❧leaves❧

The nice thing about the symbols list is that you're not restricted to the purely decorative: why not adapt the functional and use a dove 🕊 or a tunny-fish 🐟 from the phaistos package, or a bat 🦇 or a bicycle 🚲 from the marvosym package?

Swelled rules, a popular device in 19th century typesetting, tend not to be found in many font packages, as they are best constructed programmatically so that they can adapt to the width they are required for. There is an swrule package by Tobias Dussa [1] which builds a geometric lozenge from very fine lines, and there is a paper by Steve Peter [4] which describes a more extensible method using METAPOST for ConTeXt. But it is also possible to produce one using just a character from a font, and some looping in a macro with careful positioning and kerning. The following example was constructed from the swung dash (`\sim`) character in math mode, in a mirror-image. The example is also at `http://latex.silmaril.ie/packages/decorule.sty`, and any suggestions for improving and extending it are welcome.

In fact, with a little bit of practice, you can create a variety of rules and decorations built up from the symbols already available. Here's another rule, done with the left-hand and right-hand leaves mentioned earlier, rotated and arranged at intervals either side of a plain `\rule`:



Decoration is sometimes seen as gratuitous, as mere prettification. Certainly it sometimes is, but it can be much more than that. It can provide relief to the eye; conversely it can be used to attract the eye, or to divert it; it can be used to create an atmosphere or a theme; and it can even be used as a joke — early printers' 'devices' (logos) often contained visual puns on their names. So don't be afraid to decorate where it contributes to the design: there is plenty to choose from.

Apology

In TUGboat 25:1 (2004) I mentioned the automation of formatting XML using XSLT and LaTeX. I referred to two files I had used in an illustration, and said they were on my web server. Either I lied, or I forgot, or my ISP messed up when they 'upgraded' my server and my control panel and reorganised all my directories and subdomains for me.

One way or another the files went missing, and I am grateful to Vincent Douzal for pointing this out. The files have now been restored to their home at `http://silmaril.ie/xml/noaa.xml` and `.xsl`.

Afterthought

In fact, zero indentation isn't the edge case it seems. ¶ For a long period of history, documents didn't have any concept of line-breaking at a paragraph boundary, because there weren't any paragraph boundaries as such. ¶ Instead, they used the pilcrow to delimit arguments or trains of thought.

References

[1] Tobias Dussa. swrule.sty. `http://mirror.ctan.org/macros/generic/misc/swrule.sty`, Oct 2001.

[2] Donald Erwin Knuth. *The TeXbook*. Addison Wesley, Reading, MA, Jun 1986.

[3] Scott Pakin. Comprehensive LaTeX Symbol List, Nov 2009.

[4] Steve Peter. Swelled rules and METAPOST. *TUGboat*, 26(3):193–195, 2005.

◇ Peter Flynn
Textual Therapy Division, Silmaril Consultants
Cork, Ireland
Phone: +353 86 824 5333
`peter (at) silmaril dot ie`
`http://blogs.silmaril.ie/peter`

Minimal setup for a (cyrillic) TrueType font

Oleg Parashchenko

Abstract

Our goal is to describe font installation in small steps. First, we typeset plain TEX text in the right encoding. Then, we describe the minimal setup to get the correct result in PDF using both the chain `tex`+`dvips`+`ps2pdf` and the direct `pdftex`, with notes on encodings and TrueType to Type 1 font conversion. One final step for LATEX is then given. The examples are based on a cyrillic font, but useful also for other scripts as well.

1 Introduction

Several sets of instructions on how to install fonts have already been written, among them: *The LATEX Companion* [6], *The LATEX Graphics Companion* [3], and `fontinst` documentation [2].

However, I didn't have luck with them. The first problem is that the installation process, as usually described, is "atomic": after magic spells, you get the font installed, with all the required entries in config files, all in one step. If something goes wrong, an inexperienced user doesn't have control points to check what is ok, and what went awry.

The second problem is that the tutorials use Type 1 fonts as the starting point. But when we are starting with a TrueType font, it is a good idea is to skip over that starting point, and use only a part of the standard PostScript way. Otherwise the process seems unclear and superfluous.

Recently I needed a cyrillic Helvetica for a document, the build system for which used the `dvips` chain. Facing the choice of re-implementing the system using XETEX or finding a way to install the font, I decided to try the latter once more.

My new approach was to throw away all the tutorials and instead move step by step on my own. First, typeset a text in the right encoding for plain TEX. Then get the text in DVI and its viewer `xdvi`. The next steps are PostScript and PDF. Finally, the font is integrated to LATEX's NFSS.

In this guide, all the font, config and test files are located in one directory. Putting them into the right places in the `texmf` tree is left as an exercise for the reader. One hint on that: To trace which files TEX is truly using, I found that instead of `kpathsea` debug options, it is more reliable and convenient to use the system utility `strace`.

As the font, I use here Helvetica Cyrillic from Linotype (font name `HelveticaLTCYR-Roman`, file name `LT_51680.ttf`). It is a proprietary font, but that doesn't matter for our purposes: there is nothing font-specific in this guide except the names.

2 Plain TEX source

Let's typeset the word привет ("hello" in Russian). The question is: which encoding to use? It's not important as long as both the text and the font use the same encoding. In the LATEX world, cyrillic is associated with T2A, so let's use it here too.

The next question is: what is the T2A encoding? I couldn't find a reference, therefore I copied the file `t2a.enc` from my `texmf` tree and studied it. The cyrillic letters are hidden behind the names `afiiNNNNN`. I didn't see any logic in the numbers, and therefore searched for documentation and found the "Adobe Standard Cyrillic Font Specification" [1].

Our word is encoded as `afii10081 afii10082 afii10074 afii10067 afii10070 afii10084`. After calculating the positions in the encoding vector (it turns out that the T2A codes are the same as in the Windows-1251 encoding, except for the letters Ё and ё), we are ready to typeset a test document `plaintest.tex`:

```
\font\f=lhcr8z  % error: unknown font
\f ^^ef^^f0^^e8^^e2^^e5^^f2\par
\bye
```

This file doesn't compile yet because TEX does not know the font `lhcr8z` (the name is explained later).

3 Font metrics

To compile a file, TEX doesn't need the fonts themselves, but only their metrics, which are stored in `.tfm` files. One way to get a `.tfm` from our TTF:

```
ttf2tfm LT_51680.ttf -T t2a.enc lhcr8z.tfm
```

The tool displays a number of warnings like "Cannot find character 'circumflex' specified in input encoding." Indeed, the font doesn't have a glyph named `circumflex`, but rather `asciicircum`. It is possible to define aliases, but for now I just ignored the warnings. The naming issue is a big topic, and an article [4] by Hàn Thế Thành explains it in detail and suggests a general solution. An alternative is to use `fontforge` instead of `ttf2tfm`.

Now running `tex` creates a `.dvi` file, and the log file is free of warnings: `tex plaintest.tex`.

4 xdvi and .pk font

However, running `xdvi` is not successful: `xdvi` requests the font, `kpathsea` finds no font and asks `mktexpk` to generate one, but `mktexpk` doesn't know how to create it. As the font `lhcr8z` is not found, `xdvi` uses the font `cmr10` instead, but complains that the characters are not defined in it and displays an

empty page. After consulting the documentation of `mktexpk`, I created a map file `ttfonts.map`:

```
lhcr8z  LT_51680.ttf  Encoding=t2a.enc
```

Now running `xdvi` automatically creates a `.pk` file, but it is stored in some cache directory. Therefore, I prefer to create `lhcr8z.600pk` explicitly (the resolution 600 came from the output of `mktexpk`):

```
ttf2pk lhcr8z 600
```

Now this works well and shows привет:

```
xdvi plaintest.dvi
```

5 testfont

As an optional step, it's useful to get the font table, which is also the encoding table. Process the file `testfont.tex` (TEX finds it automatically):

```
tex testfont.tex
```

It first asks for a font name (answer is `lhcr8z`) and then for commands. There are a number of them, but for our needs it is enough to say:

```
\table\bye
```

Look at the result: `xdvi testfont.dvi`.

To avoid font issues due to further experiments, convert the current result to a bitmap image:

```
dvipng -o testfont.png testfont.dvi
```

Check that the glyphs are located as expected and save the table for later reference. It will be interesting to compare the result with future output from `dvips` and `pdftex`.

6 From DVI to PostScript to PDF

The classical way to create PDF from TEX is to use `dvips` and `ps2pdf`. This already works for us, but the font inside is bitmap, not vector. To get the vector version, create a local `psfonts.map`, the default map file for `dvips`:

```
lhcr8z HelveticaLTCYR-Roman <lhcr8z.pfa
```

The font `lhcr8z.pfa` can be also used in its binary form, `lhcr8z.pfb`. In the next sections we will see how to convert from TTF to Type 1. After doing that, we are ready to produce the PDF:

```
dvips -o plaintest.ps plaintest.dvi
ps2pdf plaintest.ps plaintest.pdf
```

Now running `pdffonts` (from the `xpdf` package) reports that the font `HelveticaLTCYR-Roman` is embedded and its type is `1C`. A PDF viewer should welcome us with привет.

7 pdftex instead of tex+dvips+ps2pdf

One more tool, `pdftex`, one more map file is required. This time it is named `pdftex.map`. The content is one line (the line break here is editorial):

```
lhcr8z HelveticaLTCYR-Roman <t2a.enc
      <LT_51680.ttf
```

That's all we need; `pdftex` is ready to run:

```
pdftex plaintest.tex
```

8 Encodings: TTF to Type 1 conversion

After reading different font installation instructions, I was lost in details. What are all these files and do I really need them? Extensions `tfm`, `afm`, `pfa`, `pfb`, `vf`, `fd`, `enc`, `map`, the suffixes `8a` and `8r` for versions of fonts. Font re-encoding instructions in config files. For a cyrillic font, what are the equivalents for `8a`, `8r` and the magic spells?

Thanks to the step-by-step approach, the mess was soon localized into two questions: 1) how to convert a TrueType font to PostScript, and 2) how to name it for NFSS.

The first initially looked simple. A quick search pointed to the tool `ttf2pt1` [5], which supports different encodings, including cyrillic. But then there was a fight with technical troubles. First, due to some copyright protection trick, the result was unusable without the option `-a` (include all glyphs, even those not in the encoding table). Second, I falsely concluded that `dvips` required a virtual font to use the PostScript version. Third, I didn't specify font embedding in `ttfonts.map` (using the character `<`) and had to teach `HelveticaLTCYR-Roman` to `gs`/`ps2pdf`. But finally I got привет in PDF.

The naming issue was more tricky. The beginning is obvious: `l` for Linotype, `hc` for Helvetica Cyrillic, `r` for regular. But what about the rest: `8a`, `8r` or something else? And meanwhile, for latin fonts, isn't only one of `8a` and `8r` required? (Answer: `8r` is enough.) The help came from an informal note in some tutorial: `8a` fonts should be re-encoded for use in TEX, `8r` fonts are ready to use in TEX.

Finally, the whole picture was clear for me. If a Type 1 font is made available to TEX as is, the NFSS name uses the suffix `8a`. After converting the font to TEX encoding, the suffix becomes `8r`. Further observations:

- PostScript fonts are not physically converted. Instead, they are re-encoded on the fly during PostScript execution. The corresponding commands are given through map files.
- With a TrueType font as the starting point, I don't see any reason to first convert TTF to Type 1 with the Adobe encoding, and then re-encode the font for use in TEX. Instead, I prefer to convert directly to TEX encoding.
- It seems there is a strong association between the suffix `8r` and T1 encoding, therefore for the cyrillic font I selected some other suffix, `8z`.

This idea of avoiding 8a is obvious, but I was misguided by the help text of ttf2pt1. Among the supported encodings there is adobestd, which is commented as "Adobe Standard, expected by TeX" (wrong). Meanwhile, the encoding cyrillic seems to be windows-1251, not T2A. Investigations show that it is possible to get the correct result using map files (the option -L) and that the source code tarball of ttf2pt1 contains maps for T1 and T2A, but these maps are not installed on my Linux. Therefore, currently it's inconvenient to use ttf2pt1. I'll submit a report to the developers, and hope they will improve the situation.

The alternative is the tool fontforge [8]. Initially I failed to convert the font correctly, but while fighting with ttf2pt1, I stumbled upon the documentation of the comicsans package [7], which described how to use fontforge. After adaptation to the current version, here are the instructions.

- First, teach fontforge about the T2A encoding. Click *Encoding→Load Encoding*, select the file t2a.enc.
- Select *Encoding→Reencode→T2AAdobeEncoding*. The glyphs are rearranged to the correct (for T2A) positions.
- Select *File→Generate Fonts*. You need to select options: PS Type 1 Ascii or Binary, No Bitmap Fonts, activate output of TFM and set Force glyph names to Adobe Glyph List.

9 LaTeX

The real problems are already solved. To integrate the font to LaTeX NFSS, one more config file is required, t2alhc.fd (the file name is font encoding name plus family name):

```
\ProvidesFile{t2alhc.fd}
\DeclareFontFamily{T2A}{lhc}{}
\DeclareFontShape{T2A}{lhc}{m}{n}
    { <-> lhcr8z}{}
```

A sample LaTeX document is shown next. Here utf8 is used as the input encoding, conversion to T2A is done by LaTeX, thanks to the inputenc package. The package fontenc, among other useful actions, sets the default encoding for fonts to T2A, so fontencoding before selectfont is redundant and can be safely removed.

```
\documentclass{article}
\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}
\begin{document}
\fontencoding{T2A}\fontfamily{lhc}\selectfont
^^d0^^bf^^d1^^80^^d0^^b8^^d0^^b2^^d0^^b5^^d1^^82
\end{document}
```

Both latex+dvips+ps2pdf and pdflatex should produce the desired PDF.

10 Summary

A .tfm file is always required. Create it using either ttf2tfm or fontforge.

An encoding file is also always required. It can be found in your texmf tree.

Each tool consults its own map file for more information about the fonts. You need to provide the details (such as encoding) to get the correct result.

pdfTeX can use a TTF file directly.

xdvi and dvips use either .pk bitmaps or Type 1 outlines. A .pk font is created using ttf2pk.

A Type 1 font (and the corresponding .tfm) can be created using fontforge. The font ought to be created in TeX encoding. Do not use ttf2pt1 yet, unless you understand what are you doing.

For integration in LaTeX NFSS, a .fd font description file is needed.

References

[1] Adobe Systems Incorporated. Adobe standard cyrillic font specification. See http://www.adobe.com/devnet/font/pdfs/5013.Cyrillic_Font_Spec.pdf.

[2] The fontinst home page. See http://www.tug.org/applications/fontinst/.

[3] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LaTeX Graphics Companion*. Addison-Wesley, 1997.

[4] Hàn Thế Thành. A closer look at True Type fonts and pdfTeX. *TUGboat*, 30(1):32–34, November 2009.

[5] Mark Heath. TrueType font to PostScript Type 1 converter. See http://ttf2pt1.sourceforge.net/.

[6] Frank Mittelbach and Michel Goossens. *The LaTeX Companion*. Addison-Wesley, 2004.

[7] Scott Pakin. The comicsans package. See http://www.ctan.org/macros/latex/contrib/comicsans/comicsans.pdf.

[8] George Williams. Fontforge. See http://fontforge.sourceforge.net/.

◇ Oleg Parashchenko
bitplant.de GmbH
Fabrikstr. 15
89520 Heidenheim, Germany
olpa (at) uucode dot com

LuaTEX: Microtypography for plain fonts

Hans Hagen

In a previous article we discussed plain support for OpenType fonts [1]. The latest versions now also support font extending, slanting, protrusion and expansion, the "microtypography" TEX users know from pdfTEX [2]. Here are a few examples:

```
\pdfprotrudechars2
\pdfadjustspacing2

\font\test =
 file:lmroman12-regular:+liga;extend=1.5
 \test\input tufte\par

\font\test =
 file:lmroman12-regular:+liga;slant=0.8
 \test\input tufte\par

\font\test =
 file:lmroman12-regular:+liga;protrusion=default
 \test\input tufte\par

\font\test =
 file:lmroman12-regular:+liga;expansion=default
 \test\input tufte\par
```

The `extend` and `slant` options are similar to those used in map files. The value of `extend` is limited to being within the range $[-10, 10]$ and `slant` to $[-1, 1]$.

In the protrusion and expansion specification the keyword `default` is an entry in a definition table. You can find an example at the end of the file `font-dum.lua`.

A setup for expansion looks like this:

```
fonts.expansions.setups['default'] = {
    stretch = 2, shrink = 2, step = .5,
    factor = 1,
    [byte('A')] = 0.5, [byte('B')] = 0.7,
    ..........
    [byte('8')] = 0.7, [byte('9')] = 0.7,
}
```

The stretch, shrink and steps become font properties and characters get a value assigned. In pseudocode, it looks like:

```
chr(A).expansion_factor = 0.5 * factor
```

The protrusion table has left and right protrusion factors for each relevant character.

```
fonts.protrusions.setups['default'] = {
    factor = 1, left = 1, right = 1,

    [0x002C] = { 0, 1 }, -- comma
    [0x002E] = { 0, 1 }, -- period
    [0x003A] = { 0, 1 }, -- colon
    ........
    [0x061B] = { 0, 1 }, -- arabic semicolon
    [0x06D4] = { 0, 1 }, -- arabic full stop
}
```

So, the comma will stick out in the right margin:

```
chr(comma).right_protruding
  = right * 1 * factor * (width/quad)
```

We prefer measures relative to the width (percentages), as this allows, for example, a simple `100%` to give a full protrusion.

You can add additional tables and access them by keyword in the font specification.

The model used in the plain variant is a simplification of the ConTEXt model; ConTEXt users should not take this as a starting point.

◇ Hans Hagen
Pragma ADE
The Netherlands
http://luatex.org

References

[1] Hans Hagen. Plain TEX and OpenType. http://tug.org/TUGboat/Articles/tb30-2/tb95hagen-opentype.pdf.

[2] Hàn Thế Thành. *Micro-typographic extensions to the TEX typesetting system*, Ph.D. thesis, 2000. http://tug.org/TUGboat/Articles/tb21-4/tb69thanh.pdf and http://pdftex.org.

Mathematical typefaces in TEX documents

Amit Raj Dhawan

Abstract

This paper discusses free math fonts that can be used by TEX to harmonize with desired text typefaces. It has been written with a user of plain TEX in mind but the pivotal issues discussed are common to other friends of TEX, e.g., LATEX. A technique for changing text and math fonts in TEX is given. Then the paper discusses the lack of math fonts compared to text fonts in TEX. It is followed by some deliberation on some aspects of complete sets of math fonts. The term "math mode" includes both TEX's in-line and display math modes, unless stated otherwise.

1. Changing math fonts

Changing the text font in TEX is quite simple. For example, the command `\font\myfont=cmss10` will assign the font `cmss10` to the control word `\myfont`. However, this does not work when assigning fonts to be used in math mode. TEX uses fonts from one or more of the sixteen *font families* to typeset mathematical characters. Each font family consists of three fonts — textfont, scriptfont, and scriptscriptfont.

In plain TEX, by default, most of the mathematical characters come from family 0 (*roman*), family 1 (*math italic*), family 2 (*math symbol*), and family 3 (*math extension*); TEX expects these families to be fixed. There are characters in math mode that come from other font families, e.g., $\mathbf{x}$ in $f(\mathbf{x})$ is from family 6 (`\bffam`). In all, plain TEX uses 8 families (0–7). To thoroughly change the typeface/type family of a document generated by plain TEX, it is required to change the fonts in 7 of those font families; one of the eight families is for the typewriter typeface.

The following code changes the type family in a TEX document from Computer Modern (default) to Charter. The fonts used in the code are free and included in most TEX distributions. This method works well with other formats based on plain TEX, e.g., $\mathcal{AMS}$-TEX, XeTEX (not XeLATEX), Eplain, etc. With LATEX, the given method does not work.

```
% Family 0 (Roman)
\font\tenrm=mdbchr7t at10pt
\font\sevenrm=mdbchr7t at7pt
\font\fiverm=mdbchr7t at5pt
\textfont0=\tenrm
\scriptfont0=\sevenrm
\scriptscriptfont0=\fiverm
\def\rm{\fam=0 \tenrm}
%
% Family 1 (Math italic)
\font\teni=mdbchri7m at10pt
\font\seveni=mdbchri7m at7pt
\font\fivei=mdbchri7m at5pt
\textfont1=\teni
\scriptfont1=\seveni
\scriptscriptfont1=\fivei
\def\mit{\fam=1}
%
% Family 2 (Math symbols)
\font\tensy=md-chr7y at10pt
\font\sevensy=md-chr7y at7pt
\font\fivesy=md-chr7y at5pt
\textfont2=\tensy
\scriptfont2=\sevensy
\scriptscriptfont2=\fivesy
\def\cal{\fam=2}
%
% Family 3 (Math extension)
\font\tenex=mdbchr7v at10pt
\font\sevenex=mdbchr7v at7pt
\font\fiveex=mdbchr7v at5pt
\textfont3=\tenex
\scriptfont3=\sevenex
\scriptscriptfont3=\fiveex
%
% Family 4 (Italic text)
\font\tenit=mdbchri7t at10pt
\font\sevenit=mdbchri7t at7pt
\font\fiveit=mdbchri7t at5pt
\textfont\itfam=\tenit
\scriptfont\itfam=\sevenit
\scriptscriptfont\itfam=\fiveit
\def\it{\fam=\itfam \tenit}
%
% Family 5 (Slanted text)
\font\tensl=mdbchro7t at10pt
\font\sevensl=mdbchro7t at7pt
\font\fivesl=mdbchro7t at5pt
\textfont\slfam=\tensl
\scriptfont\slfam=\sevensl
\scriptscriptfont\slfam=\fivesl
\def\sl{\fam=\slfam \tensl}
%
% Family 6 (Bold text)
\font\tenbf=mdbchb7t at10pt
\font\sevenbf=mdbchb7t at7pt
\font\fivebf=mdbchb7t at5pt
\textfont\bffam=\tenbf
\scriptfont\bffam=\sevenbf
\scriptscriptfont\bffam=\fivebf
```

```
\def\bf{\fam=\bffam \tenbf}
%
% Family 7 (Typewriter)
\font\tentt=cmtt10
\font\seventt=cmtt10 at7pt
\font\fivett=cmtt10 at5pt
\textfont\ttfam=\tentt
\scriptfont\ttfam=\seventt
\scriptscriptfont\ttfam=\fivett
\def\tt{\fam=\ttfam \tentt}
%
\rm   % Sets normal roman font
```

Another way to change the fonts is to use the control word `\newfam` [1]. The technique is similar to the one illustrated above. The macro `\newfam` allows the user to form new font families under new assigned names.

2. Available math fonts

TEX can change the math typefaces only if the fonts to do that are available. Today's TEX distributions include hundreds of fonts to change the typeface for text, but this is not the case with math. We can find hundreds of readily available free fonts offering the roman, *italic*, *slanted*, and **bold** typeface variants, but the fonts for creating specific math typefaces are not even in tens. The problem is the lack of fonts that are included in font families 1, 2, and 3 — *math italic*, *math symbol*, and *math extension* fonts. Given below is some math and text in different typefaces, typeset using free fonts available in major TEX distributions like TEX Live 2009 and MiKTEX 2.8.

Computer Modern

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Charter

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Utopia

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Century

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Palatino

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Times

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Bookman

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Antykwa Toruńska

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt} dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Iwona

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Kurier

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Arev

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Computer Modern Bright

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Concrete

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Kepler

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Euler

$$f(x,y) = (x+y)^{(2x)^{y^{x+2y^3}}}$$

$$\lim_{k\to\infty} \frac{1}{2\pi} \int_T \sigma_{n_k}(t) e^{-ijt}\,dt = \lim_{k\to\infty} \left(1 - \frac{|j|}{n_k+1}\right) c_j$$

Mathematical typesetting in TEX is more complex than text typesetting, and so is the making of math fonts compared to text fonts. Though aesthetic appreciation is subjective, most of us would agree that inter-character spacing, kerning, scripts and scriptscripts look better in Computer Modern and Euler. Here we are not talking about the design features like the contours that shape the characters or the lightness of a type family, but the fine tuning that the font offers. All the fonts used above are vector fonts, and only Computer Modern and Euler offer separate math italic and math symbol fonts at 10pt, 7pt, and 5pt. The math italic of Computer Modern is different than the normal italic of the same, the former being a bit extended. The 5pt size of Computer Modern is not 10pt scaled to 5pt; it is a separate font designed specifically for 5pt size, and this feature enhances math typesetting.

In e^{ijt} of Bookman, which uses math italic from Antonis Tsolomitis' Kerkis package, ij looks like a 'y with two dots'. This is confusing and in italic text it is a blemish on the look of words like *bijection*. It can be seen that the space setting in Kerkis is in need of improvement. In $|j|$, the *descender* (the foot of the j) cuts the vertical bar. We can improve such shortcomings manually but such alterations will be tedious and font specific — a change of font might then produce ugly results due to bad spacing. Another option is to include such tuning in the font metric information, making the process of adjustment automatic and transferable. Another alternative is to have a separate math italic font, that has its own glyphs which match the normal italic without being mere copies, which would be advantageous. After trimming the *terminal* (leg) of i of `kmath8r` (Kerkis math italic) we can get an individual glyph for math italic, which goes with the text italic but it is not a duplication. A separate math italic in Kerkis with well-chosen font metric parameters would solve most problems without the need of manual tuning. Nevertheless, some cases will always exist (especially for diehard typographers) where some manual fine-tuning would be required to produce desired results.

In the given examples, some interesting details to note are:

1. Inter-character spacing, with special attention to the spacing between scriptsize (7pt) and scriptscriptsize (5pt) glyphs, respectively.
2. The readability of smaller glyphs — scriptsize and scriptscriptsize. As discussed earlier, 5pt of Computer Modern or Euler is not 10pt scaled to 5pt. Typically, a font at 10pt scaled to 5pt has the same height as the original 5pt but the latter is wider with thicker strokes. True or original 5pt is not an extended version of 10pt scaled to 5pt either. Shown below are respective typefaces magnified to 40pt for illustration. Note the difference in the shape and stroke thickness.

   At smaller sizes it is easier to read wider and thicker types. This is the philosophy behind designing separate 7pt and 5pt fonts. These typefaces, when used in scriptsize or scriptscriptsize, match better in weight with the main text: compared with the main text, they do not seem as light as the scaled down versions.
3. Whether math symbols, Greek letters, and delimiters go with the type family. Evenness of typographic *color*. This $(\Delta + \Gamma)$ blends better with the main text than $(\Delta + \Gamma)$ or $(\Delta + \Gamma)$. The glyphs in math symbol and extension fonts should be in harmony with the text font glyphs.

Typography is an art used by many but valued by few, and practised by fewer. Moreover, in the realm of typography, mathematics is underprivileged. There are many reasons for this, the main cause being that the majority of the world never uses or only sporadically uses specialized mathematical typesetting features. Another reason is that most of the mathematicians, scientists, students, ..., who typeset mathematics, are more focussed on the content than its presentation — it suffices for them that at least and at most it works! The present typesetting algorithms of TEX suggest that typesetting mathematics is more demanding than typesetting text. TEX requires text fonts to have at least 7 `\fontdimen` parameters, whereas math symbol fonts should have at least 22 `\fontdimen` parameters and math extension fonts require at least 13 `\fontdimen` parameters. Availability of a separate math italic font is an advantage but as it is not usable as a text italic, and text italics are what most users need, there is not a huge demand for it.

3. Requirements of math fonts

The TEX world would certainly enjoy having more math fonts that have at least all the features of the legendary Computer Modern fonts. In the world of typography, two developments have been very popular — Unicode and OpenType. Is TEX ready for it? The development of XeTEX, LuaTEX, Latin Modern and TEX Gyre fonts is promising. However, this is only in the field of text typesetting. To my knowledge, none of the present TEX engines support Unicode math or any TEX fonts have OpenType math fonts.

OpenType text fonts from Adobe, like the *Pro* series, have set new heights of typographic elegance. For example, the *Warnock Pro* font family from Adobe offers 2 styles: roman and italic, in 4 weights: light, normal, semibold, and bold, in 4 optical sizes: caption, normal, subheading, and display. Amongst many other features, there is support for small caps and oldstyle figures too. Fake slanted faces, which are almost as true as real slanted when the slant is less than 0.2, can be obtained using the OpenType *slant* tag. There are some TEX fonts, like *Kpfonts*, that have light weight variants.

As Unicode is becoming the de facto encoding standard, TEX needs to gear up to accept it in a "standardized" way. Another wonderful addition would be the wholehearted acceptance of OpenType format. This would enhance TEX's communication with the non-TEX world. Opinions on these two recommendations might differ and only with time will we see what the future holds for us. We refrain from mentioning more on these topics.

At the glyph level, we can reckon a few features that TEX math fonts should offer. They should have:

1. At least three sizes (5pt, 7pt, 10pt).
2. Separate math italics suited for math and harmonious with text.
3. A complete set of math symbols and extension characters that are at least as many as offered by the AMS fonts collection. Missing glyphs are a massive disappointment and marks of unreliability.
4. The symbols and math extension characters should coordinate with the text and math letters and numbers. Balance of serifs, stroke weight, and x-height are a few considerations.
5. Availability of Script, Fraktur (lower and upper case), Blackboard fonts. It would be nice to

have more than one script glyph per character to meet special requirements.

6. All glyphs of a type family should complement each other. With proper spacing, they should give similar typographic *color* to text and math. The use of *virtual fonts*, though inviting, should be avoided.
7. All glyphs should have normal and bold variants. A semibold variant would be welcomed.
8. For slides and posters, some complete sans serif type families would be appreciated.

This list can be enlarged as refinement has no limits. It should be kept in mind that font design is an extremely demanding task. At the user level it is better to have a few sets of fonts that are well-designed and complete than having many which are incomplete. A typeset page should convey information with clarity and according to context. Academic book publishing has its own requirements, slide display has its own, and so on. Good typography does justice to all.

We have mentioned earlier in this section that it is required of TEX to adapt itself to popular font formats like OpenType. A common font design platform would allow TEX font designers to reach even non-TEXnicians — this would make their work valued by more which means greater effort.

4. Conclusion

The birth of TEX was a revolution in typesetting. Though in text typesetting today, after decades, TEX faces competition from some commercial software applications, in math typesetting it is still far ahead of the competition. Let's make our TEX, which is the best, even better!

TEX users are thankful to TEX font designers and contributors for their involvement and support. In this paper some suggestions were given, e.g., provision of original 5pt, 7pt, and 10pt sizes. Any design and effort, no matter how rudimentary, deserves its due respect. It is hoped that the words of this paper spoke with and for encouragement.

As TEX is free, so does it call for free fonts. Though it is feasible, at least in text, to use almost any font with TEX, free fonts are the real requirement of TEX. And beautiful free fonts with complete math support in the best format encoded with elegance is what TEX deserves.

References

[1] D. E. Knuth. *The TEXbook*. Addison-Wesley Pub. Co., Reading, Mass., 1986.

[2] A. R. Dhawan. "Macros to Change Text & Math fonts in TEX: 19 Beautiful Variants," CTAN, August 2009. `http://mirror.ctan.org/macros/plain/contrib/font-change/doc/`

[3] R. Bringhurst. *The Elements of Typographic Style*. Hartley & Marks, Publishers, 3rd edition, 2004.

◇ Amit Raj Dhawan
`amitrajdhawan (at) gmail dot com`

LuaTEX: Deeply nested notes

Hans Hagen

1 Introduction

One of the mechanisms that is not on a user's retina when he or she starts using TEX is 'inserts'. An insert is material that is entered at one point but will appear somewhere else in the output. Footnotes for instance can be implemented using inserts. You create a reference symbol in the running text and put note text at the bottom of the page or at the end of a chapter or document. But as you don't want to do that moving around of notes yourself TEX provides macro writers with the inserts mechanism that will do some of the housekeeping. Inserts are quite clever in the sense that they are taken into account when TEX splits off a page. A single insert can even be split over two or more pages.

Other examples of inserts are floats that move to the top or bottom of the page depending on requirements and/or available space. Of course the macro package is responsible for packaging such a float (for instance an image) but by finally putting it in an insert TEX itself will attempt to deal with accumulated floats and help you move kept over floats to following pages. When the page is finally assembled (in the output routine) the inserts for that page become available and can be put at the spot where they belong. In the process TEX has made sure that we have the right amount of space available.

However, let's get back to notes. In ConTEXt we can have many variants of them, each taken care of by its own class of inserts. This works quite well — as long as a note is visible for TEX, which more or less means: ends up in the main page flow. Consider the following situation:

```
before \footnote{the note} after
```

When the text is typeset, a symbol is placed directly after `before` and the note itself ends up at the bottom of the page. It also works when we wrap the text in an horizontal box:

```
\hbox{before \footnote{the note} after}
```

But it fails as soon as we go further:

```
\hbox{\hbox{before \footnote{the note} after}}
```

Here we get the reference but no note. This also fails:

```
\vbox{before \footnote{the note} after}
```

Can you imagine what happens if we do the following? (In this ConTEXt table, `\NC` separates columns and `\NR` separates rows.)

```
\starttabulate
\NC knuth \NC test \footnote{knuth}
    \input knuth \NC \NR
\NC tufte \NC test \footnote{tufte}
    \input tufte \NC \NR
\NC ward  \NC test \footnote{ward}
    \input ward \NC \NR
\stoptabulate
```

This mechanism uses alignments as well as quite some boxes. The paragraphs are nicely split over pages but still appear as boxes to TEX which make inserts invisible. Only the three reference symbols would remain visible. But because in ConTEXt we know when notes tend to disappear, we take some provisions, and contrary to what you might expect the notes actually do show up. However, they are flushed in such a way that they end up on the page where the table ends. Normally this is no big deal as we will often use local notes that end up at the end of the table instead of the bottom of the page, but still.

The mechanism to deal with notes in ConTEXt is somewhat complex at the source code level. To mention a few properties we have to deal with:

- Notes are collected and can be accessed any time.
- Notes are flushed either directly or delayed.
- Notes can be placed anywhere, any time, perhaps in subsets.
- Notes can be associated with lines in paragraphs.
- Notes can be placed several times with different layouts.

So, we have some control over flushing and placement, but real synchronization between for instance table entries having notes and the note content ending up on the same page is impossible.

Within the LuaTEX team we have been discussing more control over inserts and we will definitely deal with that in upcoming releases as more control is needed for complex multi-column document layouts. But as we have some other priorities these extensions have to wait.

As a prelude to them I experimented a bit with making these deeply buried inserts visible. Of course I use Lua for this as TEX itself does not provide the kind of access we need for this kind of manipulation.

2 Deep down inside

Say that we have the following boxed footnote. How does it end up in LuaTEX?

```
\vbox{a\footnote{b}c}
```

Actually it depends on the macro package but the principles remain the same. In LuaTEX 0.50 and the ConTEXt version used at the time of this writing we get a (nested) linked list that prints as follows:

```
<node   26 <  862 >  nil : vlist 0>
  <node  401 <  838 >  507 : hlist 1>
    <node   30 <  611 >  580 : whatsit 6>
    <node  611 <  580 >  493 : hlist 0>
    <node  580 <  493 >  653 : glyph 256>
    <node  493 <  653 >  797 : penalty 0>
    <node  653 <  797 >  424 : kern 1>
    <node  797 <  424 >  826 : hlist 2>
      <node  445 <  563 >  nil : hlist 2>
        <node  420 <  817 >  821 : whatsit 35>
        <node  817 <  821 >  nil : glyph 256>
    <node  507 <  826 > 1272 : kern 1>
    <node  826 < 1272 > 1333 : glyph 256>
    <node 1272 < 1333 >  830 : penalty 0>
    <node 1333 <  830 >  888 : glue 15>
    <node  830 <  888 >  nil : glue 9>
  <node  838 <  507 >  nil : ins 131>
```

The numbers are internal references to the node memory pool. Each line represents a node:

```
<node prev_index < index > next_index : type subtype>
```

The whatsits carry directional information and the deeply nested hlist is the note symbol. If we forget about whatsits, kerns and penalties, we can simplify the listing to:

```
<node   26 <  862 >  nil : vlist 0>
  <node  401 <  838 >  507 : hlist 1>
    <node  580 <  493 >  653 : glyph 256>
    <node  797 <  424 >  826 : hlist 2>
      <node  445 <  563 >  nil : hlist 2>
        <node  817 <  821 >  nil : glyph 256>
    <node  826 < 1272 > 1333 : glyph 256>
  <node  838 <  507 >  nil : ins 131>
```

So, we have a vlist (the `\vbox`), which has one line being a hlist. Inside we have a glyph (the 'a') followed by the raised symbol (the '1') and next comes the second glyph (the 'b'). But watch how the insert ends up at the end of the line. Although the insert will not show up in the document, it sits there waiting to be used. So we have:

```
<node   26 <  862 >  nil : vlist 0>
  <node  401 <  838 >  507 : hlist 1>
  <node  838 <  507 >  nil : ins 131>
```

but we need:

```
<node   26 <  862 >  nil : vlist 0>
  <node  401 <  838 >  507 : hlist 1>
<node  838 <  507 >  nil : ins 131>
```

Now, we could use the fact that inserts end up at the end of the line, but as we need to recursively identify them anyway, we cannot actually use this fact to optimize the code.

In case you wonder how multiple inserts look, here is an example:

```
\vbox{a\footnote{b}\footnote{c}d}
```

This boils down to:

```
<node   26 < 1324 >  nil : vlist 0>
  <node  401 < 1348 >  507 : hlist 1>
  <node 1348 <  507 >  457 : ins 131>
  <node  507 <  457 >  nil : ins 131>
```

And in case you wonder what more can end up at the end, vertically adjusted material (`\vadjust`) as well as marks (`\mark`) also get this treatment.

```
\vbox{a\footnote{b}\vadjust{c}%
      \footnote{d}e\mark{f}}
```

As you see, we start with the line itself, followed by a mixture of inserts and vertical adjusted content (that will be placed before that line). This trace also shows the list 2 levels deep.

```
<node   26 < 1324 >  nil : vlist 0>
  <node  401 < 1348 >  507 : hlist 1>
  <node 1348 <  507 >  862 : ins 131>
  <node  507 <  862 >  240 : hlist 1>
  <node  862 <  240 > 2288 : ins 131>
  <node  240 < 2288 >  nil : mark 0>
```

Currently vadjust nodes have the same subtype as an ordinary hlist but in LuaTEX versions beyond 0.50 they will have a dedicated subtype.

We can summarize the pattern of one 'line' in a vertical list as:

```
[hlist][insertmarkvadjust]*[penaltyglue]+
```

In case you wonder what happens with for instance specials, literals (and other whatsits): these end up in the hlist that holds the line. Only inserts, marks and vadjusts migrate to the outer level, but as they stay inside the vlist, they are not visible to the page builder unless we're dealing with the main vertical list. Compare:

```
this is a regular paragraph possibly with
inserts and they will be visible as the lines
are appended to the main vertical list \par
```

with:

```
but \vbox {this is a nested paragraph where
inserts will stay with the box} and not migrate
here \par
```

So much for the details; let's move on to how we can get around this phenomenon.

3 Some LuaTEX magic

The following code is just the first variant I made; ConTEXt ships with a more extensive variant. Also, in ConTEXt this is part of a larger suite of manipulative actions but it does not make much sense (at least not now) to discuss this framework here.

We start with defining a couple of convenient shortcuts.

```
local hlist = node.id('hlist')
local vlist = node.id('vlist')
local ins   = node.id('ins')
```

We can write a more compact solution but splitting up the functionality better shows what we're doing. The main migration function hooks into the callback `build_page`. Unlike other callbacks that do phases in building lists and pages this callback does not expect the head of a list as argument. Instead, we operate directly on the additions to the main vertical list which is accessible as `tex.lists.contrib_head`.

```
local deal_with_inserts -- forward reference

local function migrate_inserts(where)
    local current = tex.lists.contrib_head
    while current do
        local id = current.id
        if id == vlist or id == hlist then
           current = deal_with_inserts(current)
        end
        current = current.next
    end
end

callback.register('buildpage_filter',
                  migrate_inserts)
```

So, effectively we scan for vertical and horizontal lists and deal with embedded inserts when we find them. In ConTEXt the migratory function is just one of the functions that is applied to this filter.

We locate inserts and collect them in a list with `first` and `last` as head and tail and do so recursively. When we have run into inserts we insert them after the horizontal or vertical list that had embedded them.

```
local locate -- forward reference

deal_with_inserts = function(head)
    local h, first, last = head.list, nil, nil
    while h do
        local id = h.id
        if id == vlist or id == hlist then
          h, first, last = locate(h,first,last)
        end
        h = h.next
    end
    if first then
        local n = head.next
        head.next = first
        first.prev = head
        if n then
            last.next = n
            n.prev = last
        end
        return last
    else
        return head
    end
end
```

The `locate` function removes inserts and adds them to a new list, that is passed on down in recursive calls and eventually is returned back to the caller.

```
locate = function(head,first,last)
  local current = head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      current.list, first, last
         = locate(current.list,first,last)
      current = current.next
    elseif id == ins then
      local insert = current
      head, current = node.remove(head,current)
      insert.next = nil
      if first then
        insert.prev = last
        last.next = insert
      else
        insert.prev = nil
        first = insert
      end
      last = insert
    else
      current = current.next
    end
  end
  return head, first, last
end
```

As we can encounter the content several times in a row, it makes sense to mark already processed inserts. This can for instance be done by setting an attribute. Of course one has to make sure that this attribute is not used elsewhere.

```
if not node.has_attribute(current,8061) then
    node.set_attribute(current,8061,1)
    current = deal_with_inserts(current)
end
```

Or integrated:

```
local has_attribute = node.has_attribute
local set_attribute = node.set_attribute
local function migrate_inserts(where)
  local current = tex.lists.contrib_head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      if has_attribute(current,8061) then
        -- maybe some tracing message
      else
        set_attribute(current,8061,1)
        current = deal_with_inserts(current)
      end
    end
    current = current.next
  end
end; callback.register('buildpage_filter',
                       migrate_inserts)
```

4 A few remarks

Surprisingly, the amount of code needed for insert migration is not that large. This makes one wonder why TEX does not provide this feature itself as it could have saved macro writers quite some time and headaches. Performance can be a reason, unpredictable usage and side effects might be another. Only one person knows the answer.

In ConTEXt this mechanism is built in and it can be enabled by saying:

```
\automoveinserts
```

Future versions of ConTEXt will do this automatically and also provide some control over what classes of inserts are moved around. We will probably overhaul the note handling mechanism a few more times anyway as LuaTEX evolves due especially to the demands from critical editions, which use many kind of notes.

5 Summary of code

The following code should work in plain LuaTEX:

```
\directlua 0 {
local hlist         = node.id('hlist')
local vlist         = node.id('vlist')
local ins           = node.id('ins')
local has_attribute = node.has_attribute
local set_attribute = node.set_attribute

local status = 8061

local function locate(head,first,last)
  local current = head
  while current do
    local id = current.id
    if id == vlist or id == hlist then
      current.list, first, last
        = locate(current.list,first,last)
      current = current.next
    elseif id == ins then
      local insert = current
      head, current = node.remove(head,current)
      insert.next = nil
      if first then
        insert.prev, last.next = last, insert
      else
        insert.prev, first = nil, insert
      end
      last = insert
    else
      current = current.next
    end
  end
  return head, first, last
end

local function migrate_inserts(where)
  local current = tex.lists.contrib_head
  while current do
    local id = current.id
    if id == vlist or id == hlist and
       not has_attribute(current,status) then
      set_attribute(current,status,1)
      local h, first, last = current.list, nil, nil
      while h do
        local id = h.id
        if id == vlist or id == hlist then
          h, first, last = locate(h,first,last)
        end
        h = h.next
      end
      if first then
        local n = current.next
        if n then
          last.next, n.prev = n, last
        end
        current.next, first.prev = first, current
        current = last
      end
    end
    current = current.next
  end
end

callback.register('buildpage_filter',
                  migrate_inserts)
}
```

Alternatively you can put the code in a file and load that with:

```
\directlua {require "luatex-inserts.lua"}
```

A simple plain test is:

```
\vbox{a\footnote{1}{1}b}
\hbox{a\footnote{2}{2}b}
```

The first footnote only shows up when we have hooked our migrator into the callback. Not a bad result for 60 lines of Lua code.

◇ Hans Hagen
Pragma ADE
The Netherlands
http://luatex.org

The current state of the PSTricks project

Herbert Voß

Abstract

PSTricks is an abbreviation for PostScript Tricks and uses the enormous graphical capabilities of the *old* programming language PostScript. It is a so-called page code language (PCL) which is distributed since 1984 by Adobe Systems. PostScript is fully Turing compatible and a stack-oriented programming language, like Forth, Hewlett Packard pocket calculators, et al. PDF is derived from PostScript, with some important extensions but without the possibility of mathematical calculations.

1 Creating graphics with PSTricks

TEX as a typesetting machine cannot make full use of the possibilities of PostScript. There cannot be a direct interaction between TEX and PostScript; it is more of a one way communication from TEX to PostScript. The interface between these two systems is the DVI converter dvips which converts the DVI output of TEX into the PostScript format. Here we use DVI output only as an intermediate format; it is not really of interest. On the TEX side the user has to reserve some space (a box) which is filled on the PostScript side with a graphic or some text. If this space is not reserved then everything will be printed over the text, depending on the current point before the PostScript-related code. This PostScript code must be transferred from TEX via the DVI output with the macro \special. Its contents are ignored by TEX and passed to PostScript where it will be executed by a PostScript interpreter such as Ghostscript or Distiller.

Figure 1 shows the important flow from the TEX source to the destination format PDF. When using a graphical user interface (GUI) for editing the TEX source one can choose the output format PDF, but has to ensure that it follows the path LATEX–dvips–ps2pdf. For three often-used GUIs, Figure 2 shows the preferences to select for this method of compiling a LATEX source document.

The first example shows the use of PostScript-related specials without reserving any space on the TEX level. In the beginning and in the end there is a named node which can be connected by a line or a curve, in this instance by a curve with an arrow and stroke opacity. It is drawn over the text; there was no space reserved on the TEX level. In examples like these it may be useful to draw directly over the main text part of the document.

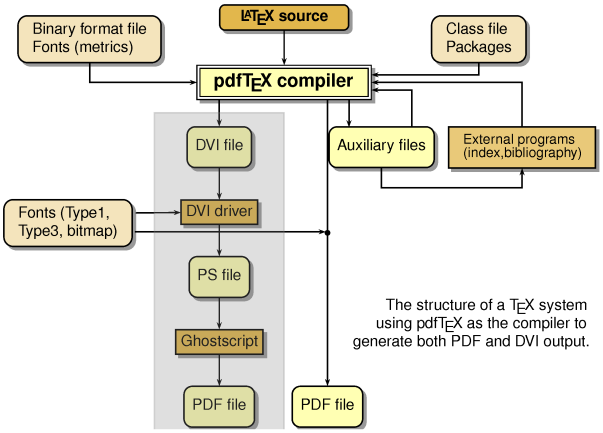
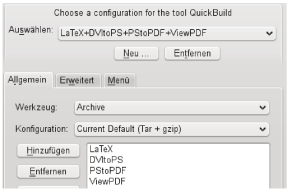
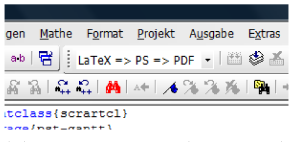


Figure 1: The shaded path shows how to generate PDF output when using PostScript code in the TEX source.



(a) Kile (Linux)



(b) TEXnicCenter (Windows)

(c) TEXmaker (Linux/Windows/Mac OS X)

Figure 2: Preferences for the destination format PDF via the intermediate format DVI.

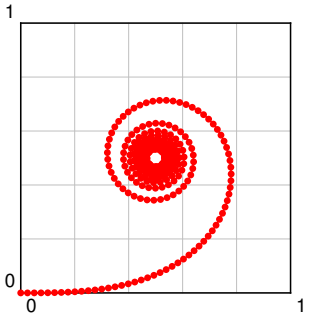
The beginning of it all: Hello, here is some text without a meaning. This text should show how a printed text will look at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like »Huardest gefburn«. Kjift: Never mind! A blind text like this gives you some information about the selected font, how the letters are written and an impression of the look of the text. The text should contain all letters of the alphabet, which could be considered a quality x-ray zoo, and it should be written in the original language. No need for special contents, but the length of words should match normal use of the language. And this is the end my friend.

```
\usepackage{blindtext,pst-node}
\SpecialCoor
\rnode{A}{The beginning of it all: }
\blindtext \rnode{E}{ And this is
the end my friend.}
\nccurve[linewidth=5pt,strokeopacity=0.4,
 ncurvB=0.9,arrowscale=1.5,arrows=-D>]{A}{E}
```

The advantage of PSTricks in comparison to METAPOST or Ti*k*Z is the possibility of using all features of a powerful programming language with very good support for graphic operations. This is the reason why *any* calculation with mathematical functions or large external data sets can be done before the output is printed. Solving a differential equation on the fly is as possible as drawing three-dimensional solids with hidden lines and surfaces; everything is done on the PostScript side. In the TEX or LATEX source one has only to define the space of the box and to describe the code with TEX or LATEX macros which are then passed as specials to PostScript. The next example shows the output of the solution of the differential equation system of first order:

$$\dot{x} = \cos\frac{\pi}{2}\cdot x^2 \tag{1}$$

$$\dot{y} = \sin\frac{\pi}{2}\cdot x^2 \tag{2}$$

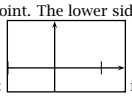



```
\usepackage{pstricks-add}
\psset{unit=5}
\begin{pspicture}(-0.04,-0.04)(1,1)
\psgrid[subgriddiv=5,subgridcolor=lightgray]
\psplotDiffEqn[whichabs=0,whichord=1,linecolor=red,
 method=rk4,algebraic,plotpoints=400,
 showpoints=true]{0}{10}{0 0}%
  {cos(Pi*x^2/2)|sin(Pi*x^2/2)}
\end{pspicture}
```

The parameters (coordinates) of the environment pspicture have a different meaning for TEX and PostScript; for TEX they define width and height of the box which has its lower left side at the current point in TEX. When there is no shift defined, then the lower side of this box is always on the baseline of the current text line. It is for TEX just the same as a box for a single letter. TEX needs the coordinates only for its formatting; what will be inserted later into this box is not of interest to TEX.

For PostScript the coordinates define a two-dimensional area with the lower left and upper right corner of the rectangle. The origin of this cartesian coordinate system may be inside or outside of this rectangle; it depends on the values of the coordinates. An example: begin{pspicture}(-1,-2)(4,4) defines for TEX a box with a width of five length units $(4-(-1))$ and a height of six length units $(4-(-2))$. For PostScript the origin of this box is one length unit to the right and two length units up, measured from the current point, which is the lower left of the TEX box.

The next example defines, on the TEX level, a box with a width and height of 2.5 cm×1.5 cm.

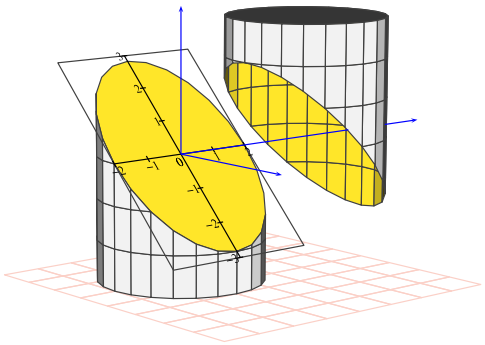
The box with a reserved space of 2,5cm×1,5cm is by definition with its lower left side at the current point. The lower side is on the baseline, which can easily be seen on the this box:  its internal origin maybe somewhere, also outside this box. In this example the origin is at (1,0.5) units, measured from the lower left corner of the box.

```
\usepackage{pstricks,pst-plot}
\raggedright The box with a reserved space
of 2,5cm$\cdot$1,5cm is by definition with
its lower left side at the current point.
The lower side is on the baseline, which
can easily be seen on the this box:
\psframebox[framesep=0]{%
\begin{pspicture}(-1,-0.5)(1.5,1)
 \psaxes[labels=none]{->}(0,0)(-1,-0.5)(1.5,1)
\end{pspicture}}
its internal origin maybe somewhere, also
outside this box. In this example the origin
is at \texttt{(1,0.5)} units, measured from
the lower left corner of the box.
```

There are several optional arguments for the main environment pspicture:

```
\pspicture * [Options] (xMin,yMin)(xMax,yMax)
    ...
\endpspicture
\pspicture * [Options] (xMax,yMax)
    ...
\endpspicture
\begin{pspicture * } [Options] (xMin,yMin)(xMax,yMax)
    ...
\end{pspicture * }
\begin{pspicture * } [Options] (xMax,yMax)
    ...
\end{pspicture * }
```

Another main focus of PSTricks is three-dimensional graphic objects, with support for the hidden line algorithm for lines and surfaces. The powerful package pst-solides3d allows combining any three-dimensional solid, given by coordinates or a mathematical expression. The following example shows the book cover image from [2], a cylindrical object divided by a plane into two parts, one of which is moved and rotated.



```
\usepackage[dvipsnames]{pstricks}
\usepackage{pst-solides3d}

\begin{pspicture}[solidmemory](-4,-5)(7,4)
\psset{viewpoint=50 -40 10 rtp2xyz,Decran=50,
 linecolor=darkgray,lightsrc=viewpoint}
\psSolid[object=grille,action=draw,base=-3 5 -3 5,
 linecolor=Salmon!40](0,0,-3)
\psSolid[object=cylindre,r=2,h=6,ngrid=6 24,
 plansepare={[0.707 0 0.707 0]},name=Zylinder,
 action=none](0,0,-3)
\psSolid[object=load,load=Zylinder1,
 fillcolor=black!5,fcol=0 (Goldenrod)]
\psSolid[object=load,load=Zylinder0,RotZ=90,
 fillcolor=black!5,rm=0,hollow,incolor=Goldenrod](0,4,0)
\psSolid[object=plan,action=draw,definition=equation,
  args={[0.707 0 0.707 0] 90},base=-2 2 -3 3,planmarks]
\psSolid[object=line,args=0 0 0 0 5.5 0,
 linecolor=blue]% first half of y axis
\color{white}\axesIIID[showOrigin=false,
 linecolor=blue](0,6.8,0)(3.5,8,3.5)
\end{pspicture}
```

2 PSTricks project background

The first version of the main package `pstricks`, written by Timothy Van Zandt and published nearly 20 years ago, is still the base package for the so-called `PSTricks` project. A list of all additional packages published since 1991 appears at the `PSTricks` web page `http://PSTricks.tug.org`. In [1] and especially [2] the packages are described and shown with a lot of examples. Here, we will list the packages with only one significant example, to give a glimpse at what the package provides. More examples or some more information can be found on the `PSTricks` web page (`http://PSTricks.tug.org`), CTAN (`http://mirrors.ctan.org`), or your local TEX distribution's documentation, e. g., using the `texdoc` program.

All `PSTricks` packages load by default the main package `pstricks`, which itself loads the package `xcolor`, which has a better support for colors than the package `color`.

3 PSTricks and PDF

Figure 1 showed the different ways of generating `PDF` output from (LA)TEX source. With `PSTricks`, only the part in grey can be used to generate a `PDF`. Unless you are using the package `microtype` there will be no difference between a `PDF` generated in the `PSTricks` way with `latex` and one directly generated with `pdflatex`. When using a graphical user interface, e. g. Kile for Linux, TeXShop for Mac OS X, or TEXnicCenter for Windows, it is only one mouse click to generate the `PDF` output. The intermediate `DVI` and PostScript files are only temporaries, and can be deleted after generating the `PDF`.

The remaining sections describe several ways of using `PSTricks`-related code within a document whose final version will be compiled with `pdflatex`, which supports inclusion of images in `PDF`, `PNG`, and `JPG` formats, as well as the `microtype` package for optimized text formatting.

3.1 dvips and ps2pdf

If your `PSTricks` figure is created by a (LA)TEX file separate from your main document, say `fig.tex`, you can process it independently. First run `tex` or `latex` on `fig.tex` to create `fig.dvi`; then `dvips fig.dvi` to create `fig.ps`; and finally `ps2pdf fig.ps` (or another distiller program) to create `fig.pdf`. Then your main document can include `fig.pdf` like any `PDF` graphic.

3.2 pst2pdf

This is a Perl script to be used instead of the `pdflatex` command. This way of creating `PDF` output is the best choice when all graphics are needed as external images.

The script extracts all `pspicture` and `postscript` environments from the main text body and then runs these code snippets with the same preamble as the main document. The `PDF` output from each of these single documents is then cropped to get rid of the white space around the figure and also converted into `EPS` and (on Linux only) `PNG` formats.

After producing all PostScript-related code as a single image, saved in a default subdirectory `images/`, the script `pst2pdf` runs the source one last time with `pdflatex` and replaces all PostScript code with the previously created image.

The script has several optional arguments which are described with their defaults at the beginning of the script.

3.3 pst-pdf and ps4pdf

This package from Rolf Niepraschk allows the cutting of the `pspicture` or `postscript` environments from the created `DVI` file into a new file `*-pics.ps`, which

then is converted into a file `*-pics.pdf`. Every image will be on one page and the size of the image is taken from the `pspicture` coordinates or from the bounding box for a `postscript` environment. In a last `pdflatex` run the `PDF` images are inserted instead of the PostScript-related code. There are four steps needed:

1. `latex FILE`
2. `dvips -Ppdf -o FILE-pics.ps FILE.dvi`
3. `ps2pdf -dAutoRotatePages=/None \ FILE-pics.ps FILE-pics.pdf`
4. `pdflatex FILE`

Alternatively, one can use the script `ps4pdf` to perform these steps. The script is part of any TEX distribution, and also available on CTAN. There are also some profiles for use from GUI programs, also available on CTAN (`http://mirror.ctan.org/graphics/pstricks/pst-support/`).

3.4 `auto-pst-pdf`

This package from Will Robertson works in the same way as `pst-pdf`, but it doesn't need a script or the four runs by the user. Everything is done in a single `pdflatex` run, and therefore you must allow execution of external programs from within `pdflatex`: the `shell-escape` option for TEX Live or `enable-write18` for MiKTEX. Some GUI profiles are available from CTAN (`http://mirror.ctan.org/graphics/pstricks/pst-support/`).

3.5 `pdftricks`

This package from Radhakrishnan CV, Rajagopal CV, and Antoine Chambert-Loir was the first one to support PostScript-related code with `pdflatex`. It works in a similar way as `pst-pdf`, but it needs additional code in the preamble to separate the PostScript part from the `PDF` part. More information is available from CTAN, or by running `texdoc pdftricks`.

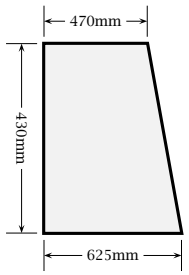
References

[1] Frank Mittelbach, Michel Goosens, Sebastian Rahtz, Denis Roegel, and Herbert Voß. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Boston, second edition, 2006.

[2] Herbert Voß. *PSTricks – Grafik für TeX und LaTeX*. DANTE – Lehmanns, Heidelberg/Hamburg, fifth edition, 2008.

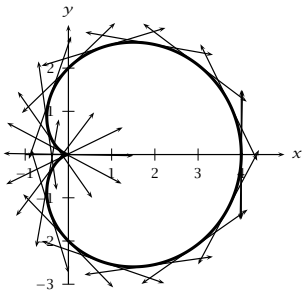
◇ Herbert Voß
DANTE e.V.
`http://PSTricks.tug.org`

`pstricks`: Main package with the base macros for lines, curves, areas, etc.



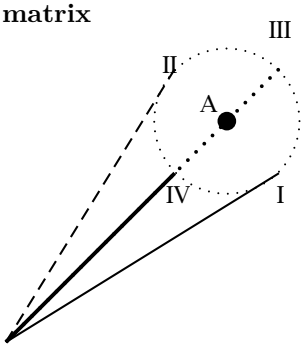

```
\usepackage{pstricks}

\psset{unit=0.05mm}% 1:20,or=0.1mm for 1:10
\begin{pspicture}(-100,-100)(1000,650)
 \pspolygon[linewidth=2pt,fillcolor=gray!10,
  fillstyle=solid](0,0)(0,470)(860,625)(860,0)
 \psset{linewidth=0.2pt,arrowscale=2,tbarsize=10pt}
 \psline{|<->|}(0,-100)(860,-100)
 \rput*(430,-100){430mm}
 \psline{|<->|}(960,0)(960,625)
 \rput*{90}(960,312.5){625mm}
 \psline{|<->|}(-100,0)(-100,470)
 \rput*{90}(-100,235){470mm}
\end{pspicture}
```

`pstricks-add`: Extended base macros for the packages `pstricks`, `pst-node`, and `pst-plot`



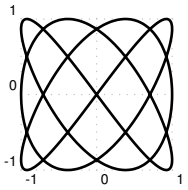

```
\usepackage{pstricks-add}
\usepackage{multido}

\psscalebox{0.75}{%
\begin{pspicture}(-1,-3)(4.75,3)
 \psaxes{->}(0,0)(-1,-3)(5,3)
 \psplot[polarplot,linewidth=2pt,algebraic,
  plotpoints=500]{0}{6.289}{2*(1+cos(x))}
 \multido{\r=0.000+0.314}{21}{ %
  \psplotTangent[polarplot,Derive=-2*sin(x),
   algebraic,arrows=<->]{\r}{1.5}{2*(1+cos(x))}}
\end{pspicture}}
```

pst-node: Nodes and node connections in text and a matrix



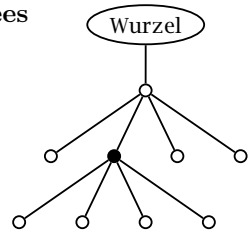

```
\usepackage{pst-node} \SpecialCoor
\begin{Example}[ltxps]{\xLcs{pnode}\xLcs{uput}}
\begin{pspicture}(4,4)
  \pnode(3,3){A}\psdot[dotscale=2](A)\uput[135](A){A}
  \pscircle[linestyle=dotted](A){1}
  \psline([nodesep=1,angle=-45]A)\uput[0](3.5,2){I}
  \psline[linestyle=dashed]([nodesep=-1,angle=-45]A)
  \uput[-45](2,4){II}
  \psline[linestyle=dotted,linewidth=1.5pt]%
    ([offset=1,angle=-45]A)\uput[-225](4,4){III}
  \psline[linewidth=1.5pt]([offset=1,angle=135]A)
  \uput[0](2,2){IV} \nccurve{->}{A}{0,0}
\end{pspicture}
```

pst-plot: Plotting of mathematical functions or external data sets



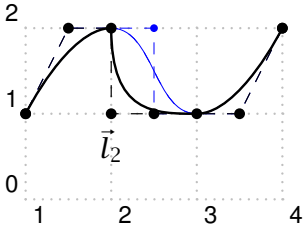

```
\usepackage{pst-plot}
\psset{xunit=1.5cm,yunit=1.5cm}
\begin{pspicture}[showgrid=true](-1.1,-1.1)(1.1,1.1)
\psparametricplot[plotstyle=curve,linewidth=1.5pt,
 plotpoints=200]{-360}{360}%
 {t 1.5 mul sin t 2 mul 60 add sin}
\end{pspicture}
```

pst-tree: Trees



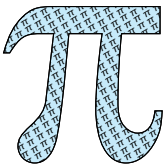

```
\usepackage{pst-tree}
\pstree[levelsep=1cm,radius=3pt]{\Toval{Wurzel}}{%
 \pstree{\TC}{%
  \TC
  \pstree{\TC*}{\TC\TC\TC\TC}%
  \TC\TC}}
```

pst-bezier: Bézier curves




```
\usepackage{pst-bezier}
\pspicture[showgrid=true](5,3)\psset{showpoints=true}
\psbcurve[linecolor=blue,linewidth=0.01](1,1)%
 (2,2)(3,1)(4,2)
\psbcurve(1,1)(2,2)l(2,1)(3,1)(4,2)
\uput[-90](2,1){$\vec{l}_{2}$}
\endpspicture
```

pst-text: Character and text manipulation

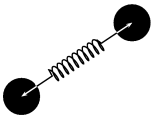


```
\usepackage{pst-text}
\usepackage[tiling]{pst-fill}
\DeclareFixedFont{\ps}{U}{psy}{m}{n}{8cm}
\psboxfill{\footnotesize$\pi$}
\begin{pspicture}(0,0)(.25\linewidth,8)
\pscharpath[fillstyle=solid,
  fillcolor=cyan!20,
  addfillstyle=boxfill,fillangle=30,
  fillsep=0.6mm]{%
   \rput[b](-0.5,0){\ps\char112}}
\end{pspicture}
```

pst-fill: Filling and tiling

```
\usepackage[tiling]{pst-fill}
\newcommand\FArcW{%
 \begin{pspicture}(-0.25,-0.25)(0.25,0.25)%
  \pswedge*(-0.25,-0.25){0.25}{0}{90}
  \pswedge*(0.25,0.25){0.25}{180}{270}
  \psframe[linewidth=0.1pt]%
   (-0.25,-0.25)(0.25,0.25)
 \end{pspicture}}
\begin{pspicture}(3.1,3.1)
\psboxfill{\FArcLW}
\psframe[fillstyle=boxfill,fillcyclex=2,
 fillangle=45](3,3)
\end{pspicture}
```

pst-coil: Coils and zigzag lines

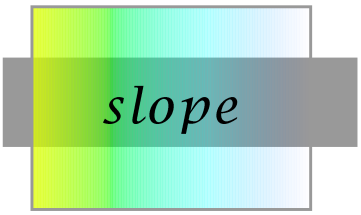


```
\usepackage{pst-node,pst-coil}
\SpecialCoor

\begin{pspicture}(4,3)
\cnode*(0.5,0.5){0.5}{A}
\cnode*(3.5,2.5){0.5}{B}
\pccoil[coilwidth=0.4,
  coilaspect=35,coilheight=0.5,
  linecolor=white]{<->}(A)(B)
\nccoil[coilwidth=0.4,coilaspect=35,
  coilheight=0.5]{A}{B}
\end{pspicture}
```

pst-grad: Color gradients

```
\usepackage{pst-grad}

\begin{pspicture}(3,2.25)
\psframe[fillstyle=gradient](3,2)
\end{pspicture}
```

pst-slpe: Extended color gradients

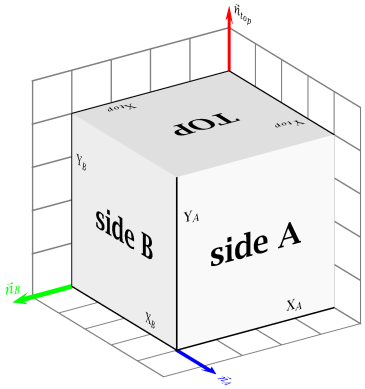



```
\usepackage{pst-slpe}
\def\st#1{\makebox[2.75cm]{$\vcenter to55pt{}#1$}}

\begin{pspicture}(2.5,2.5)
 \psset{fading,endfading=0.75,linecolor=black!40}
 \psframe*(-0.3,-0.25)(3.5,20pt)
 \psframebox[fillstyle=slope]{\LARGE\st{slope}}
\end{pspicture}
```

pst-blur: Shadows

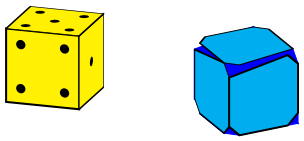

```
\usepackage{pst-blur,pst-text}
\DeclareFixedFont{\RM}{T1}{ptm}{b}{n}{1.75cm}

\psset{shadow=true,blur=true,shadowsize=10pt,
 blurradius=5pt}
\pscharpath{\RM PSTricks}
```

pst-3d: Basic macros for tilting and three dimensional objects



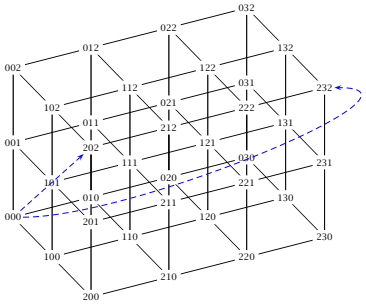

```
\usepackage{pst-3d}

\begin{pspicture}(-4.5,-3)(3,4.75)
\psset{viewpoint=1 1.5 0.8}
{\psset{gridlabels=0pt,subgriddiv=0,gridcolor=black!50}
\ThreeDput[normal=0 0 1]{\psgrid(5,5)}
\ThreeDput[normal=0 -1 0]{\psgrid(5,5)}
\ThreeDput[normal=1 0 0]{\psgrid(5,5)}}
\ThreeDput[normal=0 0 1]{%
  \psline[linewidth=3pt,linecolor=blue]{->}(4,4)(4,5.5)
  \uput[90](4,5.5){%
    \psrotateleft{\textcolor{blue}{$\vec{n}_A$}}}}%$%xy
\ThreeDput[normal=0 -1 0]{%
  \psline[linewidth=3pt,linecolor=green]{->}(4,0)(5.5,0)
  \uput[90](5.5,0){\psscalebox{-1 1}{%
    \textcolor{green}{$\vec{n}_B$}}}}%$%xz
\ThreeDput[normal=1 0 0]{%
  \psline[linewidth=3pt,linecolor=red]{->}(0,4)(0,5.5)
  \uput[0](0,5.5){$\vec{n}_{top}$}} %yz
\ThreeDput[normal=0 0 1](0,0,4){%
  \psframe*[linecolor=gray!25](4,4)
  \rput(2,2){\Huge\textbf{TOP}}}
\ThreeDput[normal=0 1 0](4,4,0){%
  \psframe*[linecolor=gray!5](4,4)
  \rput(2,2){\Huge\textbf{side A}}}
\ThreeDput[normal=1 0 0](4,0,0){%
  \psframe*[linecolor=gray!15](4,4)
  \rput(2,2){\Huge\textbf{side B}}}
% Die kleinen Achsen
\ThreeDput[normal=0 0 1](0,0,4){%
  \psline(4,0)\uput[90](3,0){X$_{top}$}
  \psline(0,4)\uput[0](0,3){Y$_{top}$}}
\ThreeDput[normal=0 1 0](4,4,0){%
  \psline(4,0)\uput[90](3,0){X$_{A}$}
  \psline(0,4)\uput[0](0,3){Y$_{A}$}}
\ThreeDput[normal=1 0 0](4,0,0){%
  \psline(4,0)\uput[90](3,0){X$_{B}$}
  \psline(0,4)\uput[0](0,3){Y$_{B}$}}
\end{pspicture}
```

pst-ob3d: Simple three dimensional objects



```
\usepackage{pst-ob3d}\SpecialCoor

\begin{pspicture}(-1,-0.5)(3,1.7)
\psset{fillstyle=solid,fillcolor=yellow,RandomFaces=true}
\PstDie[viewpoint=1 -3 1]
\rput(1.5,0){\PstCube[Corners=true,CornersColor=blue,
  fillstyle=solid,fillcolor=cyan,viewpoint=1 2 1]
  {1}{1}{1}}
\end{pspicture}
```

pst-gr3d: Simple three dimensional grids

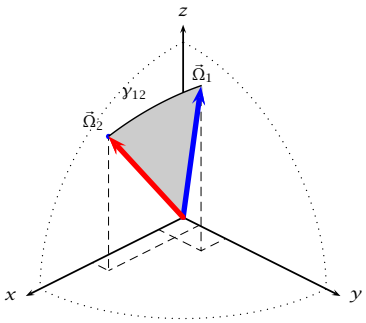


```
\usepackage{pst-gr3d,pst-node,multido}\SpecialCoor

\psscalebox{0.5}{\footnotesize
\PstGridThreeD[GridThreeDNodes,unit=2.5](2,3,2)
\multido{\ix=0+1}{3}{%
  \multido{\iy=0+1}{4}{%
    \multido{\iz=0+1}{3}{%
      \rput*(Gr3dNode\ix\iy\iz){$\ix\iy\iz$}}}}
\psset{linecolor=blue,linestyle=dashed,
  linewidth=0.3pt,arrowscale=2,nodesep=8pt}
\pcline{->}(Gr3dNode000)(Gr3dNode202)
\pccurve{->}(Gr3dNode000)(Gr3dNode232)}
```

pst-fr3d: Three dimensional buttons

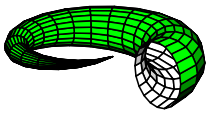

```
\usepackage{pst-fr3d}

\PstFrameBoxThreeD
  [FrameBoxThreeDOn=false]
  {\Large Off}
\quad
\PstFrameBoxThreeD{\Large On}% default
```

pst-3dplot: Three dimensional graphic objects in parallel projection



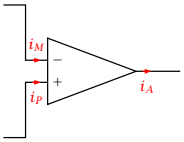

```
\usepackage{pst-3dplot}
\def\oA{\pstThreeDLine[linecolor=blue,linewidth=3pt,
  arrows=c-> ](0,0,0)(1,60,70)}
\def\oB{\pstThreeDLine[linecolor=red,linewidth=3pt,
  arrows=c->](0,0,0)(1,10,50)}
\def\oAB{\pstThreeDEllipse[beginAngle=58,
  endAngle=90](0,0,0)(1,140,40)(1,10,50)}

\begin{pspicture}(-4.8,-1.75)(4.8,3.75)
\psset{unit=4cm,drawCoor,beginAngle=90,endAngle=180,
  linestyle=dotted}
\pstThreeDCoor[drawing,linewidth=1pt,linecolor=black,
  linestyle=solid,xMin=0,xMax=1.1,yMin=0,yMax=1.1,
  zMin=0,zMax=1.1]
\pstThreeDEllipse(0,0,0)(-1,0,0)(0,1,0)
\pstThreeDEllipse(0,0,0)(-1,0,0)(0,0,1)
\pstThreeDEllipse[beginAngle=0,
  endAngle=90](0,0,0)(0,0,1)(0,1,0)
\psset{SphericalCoor,linestyle=solid}
\pstThreeDDot[dotstyle=none](1,10,50)
\pstThreeDDot[dotstyle=none](1,60,70)
\pscustom[fillstyle=solid,fillcolor=black!20,
  linestyle=none]{\oB\oAB\oA} \oA\oB\oAB
\pstThreeDPut[origin=lb](1.1,60,70){$\vec\Omega_1$}
\pstThreeDPut[origin=rb](1.2,10,50){$\vec\Omega_2$}
\pstThreeDPut[origin=lb](1,10,65){$\gamma_{12}$}
\end{pspicture}
```

pst-solides3d: Three dimensional graphic objects in central projection



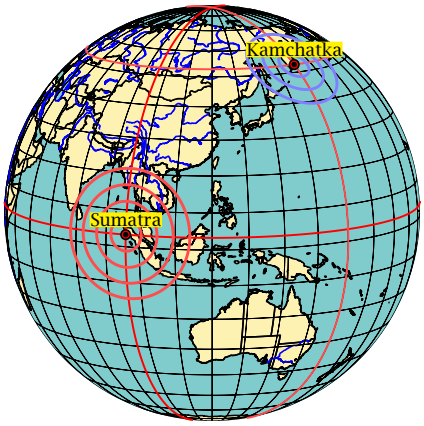
```
\usepackage{pst-solides3d}

\psset{viewpoint=100 50 20 rtp2xyz,
  Decran=60,lightsrc=10 15 10}
\defFunction[algebraic]{corne}(u,v)%
  {(2 + u*cos(v))*sin(2*pi*u)}%
  {(2 + u*cos(v))*cos(2*pi*u)+2*u}%
  {u *sin(v)}
\begin{pspicture}(-2,-2)(2,2)
\psSolid[object=surfaceparametree,
  base=0 1 0 2 pi mul,
  function=corne,ngrid=20]
\end{pspicture}
```

pst-circ: Electronic and microelectronic electrical circuits



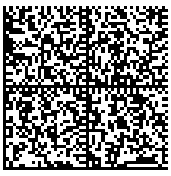

```
\usepackage{pst-circ}

\begin{pspicture}(4,3.5)
\pnode(0,3){A}\pnode(0,0){B}
\pnode(4,1.5){C}
\OA[OAperfect=false,OAiplus,
  OAiminus,OAiout,
  OAipluslabel=$i_P$,
  OAiminuslabel=$i_M$,
  OAioutlabel=$i_A$,
  intensitycolor=red,
  intensitylabelcolor=red](A)(B)(C)
\end{pspicture}
```

pst-geo: Two and three dimensional geographical objects



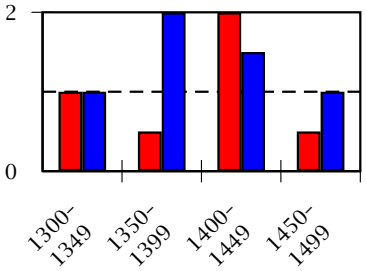

```
\usepackage{pst-map3d}

\psset{unit=0.75,Radius=5,Dobs=200,Decran=200,
  path=data/pstricks,PHI=10,THETA=120,circlewidth=1.5pt}
\begin{pspicture}(-5,-5)(5,5)
\WorldMapThreeD[circles=false,australia=true]
\psmeridien{95.98} \psparallel{3.30}
\psepicenter[circlecolor=red!70,waves=4,
  Rmax=2000](95.98,3.30){Sumatra}
\psmeridien[meridiencolor=red!70]{160}
\psparallel[parallelcolor=red!70]{52.76}
\psepicenter[circlecolor=blue!50](160,52.76){Kamchatka}
\end{pspicture}
```

pst-barcode: Barcodes



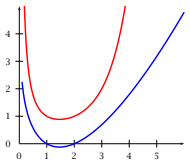
```
\usepackage{pst-barcode}

\begin{pspicture}(1in,1in)
\psbarcode{%
  Herbert Voss Wasgenstraße 21 14129 Berlin
  http://www.dante.de/}%
  {rows=52 columns=52}{datamatrix}
\end{pspicture}
```

pst-bar: Bar diagrams



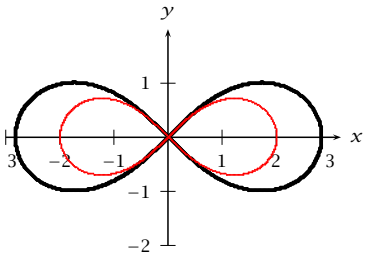

```
\usepackage{pst-plot,pst-bar}
\begin{filecontents*}{data1T.csv}
1300--1349, 1350--1399, 1400--1449, 1450--1499
1, 0.5, 2, 0.5
1, 2, 1.5, 1
\end{filecontents*}

\readpsbardata{\data}{data1T.csv}
\begin{pspicture}(-0.5,-2)(4,2)\footnotesize
\psline[linestyle=dashed](0,1)(4,1)
\psaxes[axesstyle=frame,Dy=2,labels=y](0,0)(4,2)
\psbarchart[barstyle={red,blue},barlabelrot=45,
  chartstyle=cluster]{\data}
\end{pspicture}
```

pst-math: Extended PostScript functions



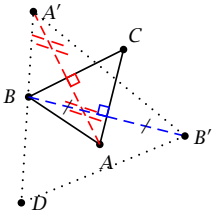
```
\usepackage{pst-plot,pst-math}

\begin{pspicture*}(-0.75,-.75)(6,5)
 \psaxes{->}(6,5)
 \psset{linewidth=1.5pt,
   plotpoints=200}
 \psplot{.1}{6}{x GAMMA}
 \psplot{.1}{6}{x GAMMALN}
\end{pspicture*}
```

pst-func: Special mathematical functions: polynomials, distributions, implicit, etc.



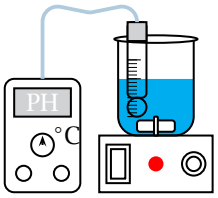

```
\usepackage{pst-func,pstricks-add}
\begin{pspicture*}(-3,-2.2)(3.6,2.5)
\psaxes{->}(0,0)(-3,-2)(3.2,2)[$x$,0][$y$,90]
\psplotImp[linewidth=2pt,algebraic](-5,-2.2)(5,2.4){%
  (x^2+y^2)^2-8*(x^2-y^2)}
\rput*(2,1.5){$\left(x^2+y^2\right)^2-8(x^2-y^2)=0$}
\psplotImp[linewidth=1pt,linecolor=red,
  algebraic](-5,-2.2)(5,2.4){(x^2+y^2)^2-4*(x^2-y^2)}
\end{pspicture*}
```

pst-eucl: Euclidean geometry



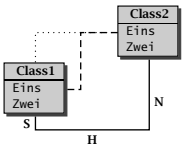

```
\usepackage{pst-eucl}
\psset{unit=0.4}%
\begin{pspicture}(0,-2)(8,7)
\pstTriangle(1,3){B}(5,5){C}
        (4,1){A}
\pstOrtSym{A}{B}{C}[D]
\psset{CodeFig=true}
\pstOrtSym[CodeFigColor=red]
       {C}{B}{A}
\pstOrtSym[SegmentSymbol=pstslash,
  dotsep=3mm,linestyle=dotted,
  CodeFigColor=blue]{C}{A}{B}
\pspolygon[linestyle=dotted,
  linewidth=1pt](A')(B')(D)
\end{pspicture}
```

pst-labo: Chemical objects



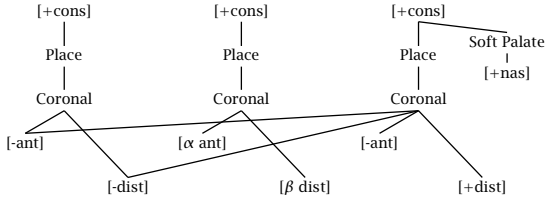

```
\usepackage{pst-labo}

\psset{unit=0.5cm,
  glassType=becher,
  burette=false}
\pstDosage[phmetre]
```

pst-uml: UML diagrams



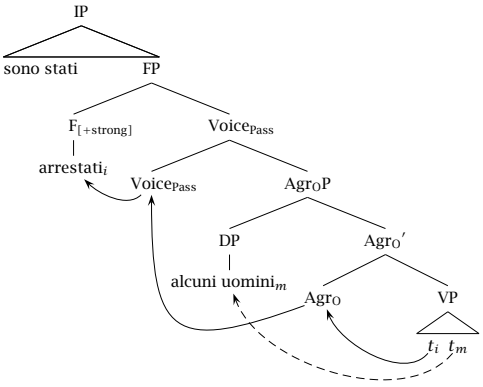

```
\usepackage{pst-uml}
\begin{pspicture}(5,4)
\rput(1,1.5){\rnode{A}{%
  \umlClass{Class1}{Eins\\Zwei}}}
\rput(4,3){\rnode{B}{%
  \umlClass{Class2}{Eins\\Zwei}}}
\end{pspicture}
\psset{linewidth=1pt}%
\ncNE[linestyle=dotted]{A}{B}
\ncEVE[linestyle=dashed]{A}{B}
\ncSHN{A}{B}\nbput[npos=0.5]
    {\textbf{S}}
\nbput[npos=1.5]{\textbf{H}}
\nbput[npos=2.5]{\textbf{N}}
```

pst-asr: Autosegmental representations for linguistics



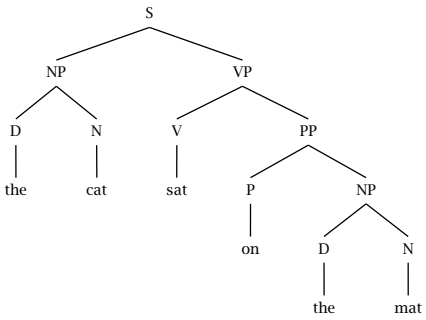

```
\usepackage{pst-asr}
\newpsstyle{dotted}{linestyle=dotted,
  linewidth=1.2pt,dotsep=1.6pt}
\newpsstyle{crossing}{xed=true,
  xedtype=\xedcirc,style=dotted}
\newpsstyle{dotted}{linestyle=dotted,
  linewidth=1.2pt,dotsep=1.6pt}
\newpsstyle{crossing}{xed=true,xedtype=\xedcirc,
                style=dotted}
\newtier{softpal,ant,dist,nasal} \tiershortcuts
\psset{xgap=1.5in,yunit=3em,ts=0 (Pg),sy=1 ([),
  ph=-1 (Cg),softpal=.3 (Sg),nasal=-.4 ([),ant=-2 ([),
  dist=-3 ([),tssym=Place,sysym=\textrm{[+cons]},
  everyph=Coronal}
\DefList{\softpalA{2.5},\antoffset{-.22},
  \distoffset{.36}} \quad \asr \1{}\1{}\1{}|
\@(\softpalA,softpal){Soft Palate} \-(2,sy)
\@(\softpalA,nasal){\textrm{[+nas]}}
  \-(\softpalA,softpal) % ant features
\@(\antoffset,ant){\textrm{[-ant]}} \-(0,ph)
  \-[style=crossing](2,ph)
\@[1](\antoffset,ant){\textrm{[$\alpha$ ant]}} \-(1,ph)
\@[2](\antoffset,ant){\textrm{[-ant]}} \-(2,ph)
\@(\distoffset,dist){\textrm{[-dist]}} \-(0,ph)
  \-[style=crossing](2,ph)
\@[1](\distoffset,dist){\textrm{[$\beta$ dist]}} \-(1,ph)
\@[2](\distoffset,dist){\textrm{[+dist]}} \-(2,ph)
|\endasr
```

pst-jtree: Linguistic trees



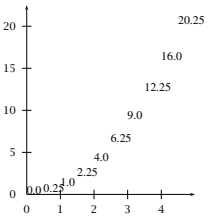

```
\usepackage{pst-jtree}
\jtree[xunit=5em,yunit=2em]
\! = {IP}
  <tri>{\triline{sono stati\hfil}}
  ^<tri>[triratio=.95]{FP}
  :{F$_{\rlap{$\scriptstyle\rm
    [+strong]$}}$}!a {Voice$_{%
     \rlap{$\scriptstyle\rm Pass$}}$}
  :{Voice\rlap{$_{\rm Pass}$}}@A2 {$\rm Agr_OP$}
  :{DP}!b {${\rm Agr_O}'$}
  :[scaleby=.8 1]{$\rm Agr_O$}@A3 [scaleby=.8 1]{VP}
  <tri>[scaleby=.4 .7]{\rnode{A5}{$t_i$}
    \hskip1ex \rnode{A6}{$t_m$}}.
\!a = <shortvert>{arrestati$_i$}@A1 .
\!b = <shortvert>{alcuni uomini$_m$}@A4 .
\psset{arrows=->}
\nccurve[angleA=225,angleB=-45]{A2}{A1}
\nccurve[angleA=200,angleB=-90,ncurv=1.5]{A3}{A2}
\nccurve[angleA=-130,angleB=-70]{A5}{A3}
\nccurve[angleA=-130,angleB=-70,
  linestyle=dashed]{A6}{A4}
\endjtree
```

pst-qtree: A qtree-like interface for drawing trees



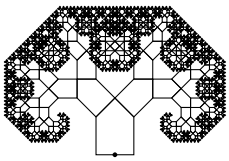

```
\usepackage{pst-qtree}

\begin{Example}[ltxps]{\xLcs[pst-qtree]{Tree}}
\Tree
 [.S [.NP [.D the ] [.N cat ]
 ] [.VP [.V sat ] [.PP [.P on ]
  [.NP [.D the ] [.N mat ] ] ] ] ]
```

infix-RPN: Converting an algebraic expression (infix) to a PostScript expression (postfix)



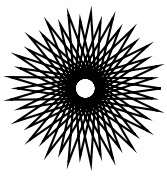

```
\usepackage{infix-RPN,pst-func,
 multido}
\SpecialCoor

\psset{yunit=0.25}
\begin{pspicture}(-0.25,-2)(5,22.5)
\infixtoRPN{x*x}
\multido{\rx=0.0+0.5}{10}{%
 \rput(!/x \rx\space def
 \RPN\space x exch ){%
  \psPrintValue{\RPN}}}
\psaxes[dy=5,Dy=5]{->}(5,22.5)
\end{pspicture}
```

pst-fractal: Fractals



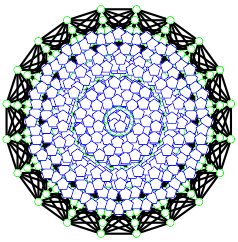
```
\usepackage{pst-fractal}

\begin{pspicture}(-3,0)(3,4)
 \psPTree\psdot*(0,0)
\end{pspicture}
```

pst-poly: Polygons



```
\usepackage{pst-poly}

\PstPolygon[PolyNbSides=21,
 PolyOffset=2,
 PolyIntermediatePoint=-0.9]
```

pst-coxeterp: Regular polytopes

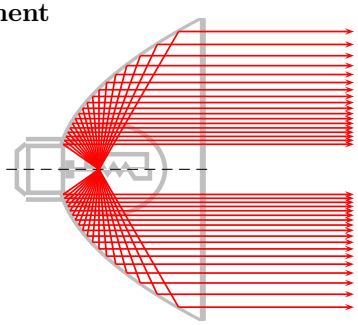


```
\usepackage{pst-coxeterp}

\begin{pspicture}(-2,-2)(2,2)
\psset{unit=0.4cm,
 colorCenters=blue,
 styleCenters=pentagon,
 sizeCenters=0.2}
\gammapn[P=5,dimension=4]
\end{pspicture}
```

pst-lens: Lens magnification

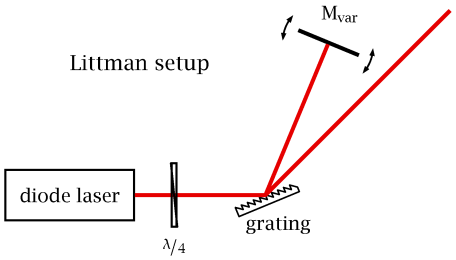

```
\usepackage{pst-lens}

\def\Wishes{{%
 \rput[lb](0,0){%
  \Large\begin{minipage}{3cm}
   \centering
   \textbf{All the best}\\
    \textbf{Jana},\\
     for the new year\\\Huge 2010!
  \end{minipage}}}}
  \begin{pspicture}(0,-1.5)(3,4)
 \Wishes\PstLens[LensMagnification=2]%
  (1.5,2.5){\Wishes}
\end{pspicture}
```

pst-optic: Two dimensional optical arrangement



```
\usepackage{pst-optic}
\psset{unit=0.5}
\begin{pspicture}(-1.5,-5.5)(10,5.5)
 \rput(0,0){\beamLight[drawing=false,mirrorDepth=4.75,
  mirrorWidth=0.1,mirrorHeight=10,linecolor=lightgray]}
 \makeatletter \pst@getcoor{Focus}\pst@tempf
 \psset{linecolor=red}
 \multido{\n=60+5}{18}{\mirrorCVGRay[linecolor=red,
  mirrorDepth=4.75,mirrorHeight=10](Focus)(!
   /XF \pst@tempf pop \pst@number\psxunit div def
  \n\space cos XF add \n\space sin neg){Endd1}
  \psOutLine[arrows=->,length=.25](Endd1)(Endd1'){Endd2}
  \mirrorCVGRay[linecolor=red,mirrorDepth=4.75,
   mirrorHeight=10](Focus)(!
    /XF \pst@tempf pop \pst@number\psxunit div def
     \n\space cos XF add \n\space sin ){End1}
  \psOutLine[arrows=->,length=.25](End1)(End1'){End2}}
 \makeatother
\end{pspicture}
```

pst-optexp: Experimental optics

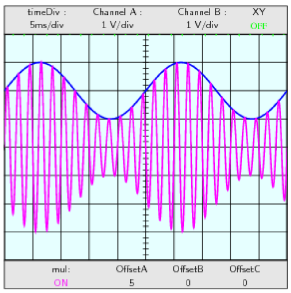



```
\usepackage{pst-optexp, nicefrac}

\begin{pspicture}(-4,-1)(3,3)
\addtopsstyle{Beam}{linewidth=2\pslinewidth,
  linecolor=red!90!black}
\psset{labeloffset=0.5}
\pnode(-2,0){LaserOut}\pnode(0,0){Grat}
\pnode(4;45){Out}\pnode(2.5;67.5){Mvar}
\optbox[optboxwidth=2,labeloffset=0,
  endbox](Grat)(LaserOut){diode laser}
\mirror[variable,conn=o-] %
  (Grid)(Mvar)(Grid){M$_\mathrm{var}$}
\optgrid[beam](LaserOut)(Grat)(Out){grating}
\optretplate[position=0.3,labeloffset=0.8] %
  (LaserOut)(Grat){$\nicefrac{\lambda}{4}$}
\rput[l](-3,2){Littman setup}
\end{pspicture}
```

pst-diffraction: Generating a diffraction

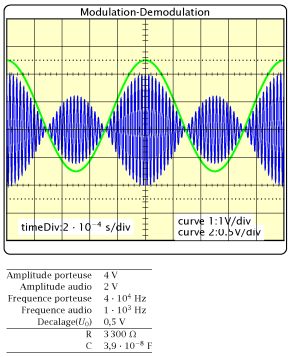
```
\usepackage{pst-diffraction}

\begin{pspicture}(-3.5,-1.5)(3.5,3.5)
\psdiffractionCircular[IIID,r=0.5e-3,f=10,
  pixel=0.5,lambda=520,colorMode=0]
\end{pspicture}
```

pst-osci: Simulating the output of an oscilloscope



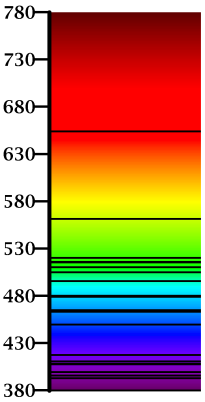

```
\usepackage{pst-osci}

\Oscillo[amplitude1=1,
  amplitude2=1,CC2=2,
  period2=25,period1=2,
  combine=true,
  operation=mul,
  offset1=5]
```

pst-am: Modulation and demodulation



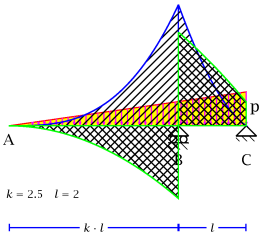

```
\usepackage{pst-am}

\psAM[SignalModulant,
  SignalModule,
  timeDiv=2e-4,
  U0=0.5,
  frequencePorteuse=4e4,
  Up=4,Um=2,
  voltDivY2=0.5,values]
```

pst-spectra: Spectral lines



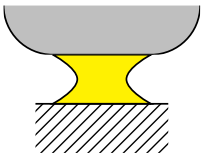

```
\usepackage{pst-spectra,
  pstricks-add}

\begin{pspicture}(0,-0.75)(2,4.75)
\rput{90}(1,0){%
  \psspectrum[axe,Dl=50,wlangle=-90,
   wlcmd={\scriptsize\bfseries},
   begin=780,end=380,
   element=Es,
   absorption](5,2)(0,0)}
\end{pspicture}
```

pst-stru: Structural schemes in civil engineering



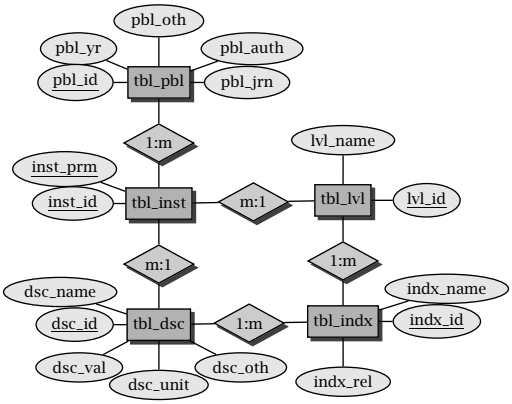

```
\usepackage{pst-stru}

\begin{pspicture}(-1,-3)(12,4)
  \psset{arrowsize=0.8mm,
    arrowinset=0}
  \triload[K=2.5,P=8,L=2]
\end{pspicture}
```

`pst-pad`: Attachment models



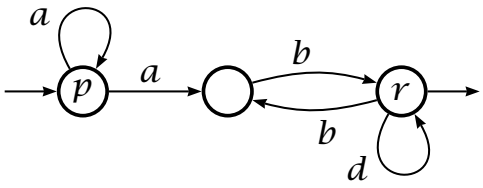
```
\usepackage{pst-pad}

\begin{pspicture}(4,4)
\PstPad(2,2)
\end{pspicture}
```

`pst-dbicons`: Entity-Relationship diagrams



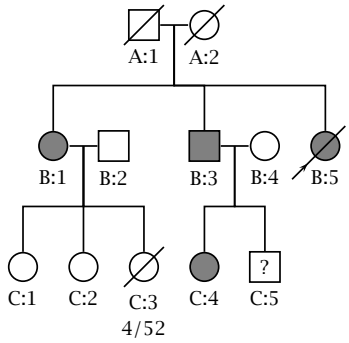

```
\usepackage{pst-dbicons}

\seticonparams{entity}{shadow=true,fillcolor=black!30,
 fillstyle=solid}
\seticonparams{attribute}{fillcolor=black!10,
 fillstyle=solid}
\seticonparams{relationship}{shadow=true,
 fillcolor=black!20,fillstyle=solid}
\begin{tabular}{cc}
 \begin{tabular}{c}
  \entity{tblpbl}[tbl\_pbl]\\[2cm]
  \entity{tblinst}[tbl\_inst]\\[2cm]
  \entity{tbldsc}[tbl\_dsc] \\[2cm]
 \end{tabular}\hspace{6em}
 \begin{tabular}{c}~\\[2cm]
  \entity{tbllvl}[tbl\_lvl]\\[2cm]
  \entity{tblindx}[tbl\_indx]\\[2cm]
 \end{tabular}
 \attributeof{tblpbl}[3em]{0}{pbljrn}[pbl\_jrn]
 \attributeof{tblpbl}[3em]{90}{pbloth}[pbl\_oth]
 \attributeof{tblpbl}[3em]{30}{pblauth}[pbl\_auth]
 \attributeof{tblpbl}[3em]{150}{pblyr}[pbl\_yr]
 \attributeof{tblpbl}[3em]{180}[key]{pblid}[pbl\_id]
 \attributeof{tblinst}[3em]{150}[key]{instprm}[inst\_prm]
 \attributeof{tblinst}[3em]{180}[key]{instid}[inst\_id]
 \attributeof{tbldsc}[3em]{180}[key]{dscid}[dsc\_id]
 \attributeof{tbldsc}[3em]{150}{dscname}[dsc\_name]
 \attributeof{tbldsc}[3em]{220}{dscval}[dsc\_val]
 \attributeof{tbldsc}[3em]{270}{dscunit}[dsc\_unit]
 \attributeof{tbldsc}[3em]{320}{dscoth}[dsc\_oth]
 \attributeof{tbllvl}[3em]{0}[key]{lvlid}[lvl\_id]
 \attributeof{tbllvl}[3em]{90}{lvlname}[lvl\_name]
 \attributeof{tblindx}[3em]{0}[key]{indxid}[indx\_id]
 \attributeof{tblindx}[3em]{30}{indxname}[indx\_name]
 \attributeof{tblindx}[3em]{270}{indxrel}[indx\_rel]
 \relationshipbetween{tblpbl}{tblinst}{1:m}%relationships
 \relationshipbetween{tblinst}{tbldsc}{m:1}
 \relationshipbetween{tblinst}{tbllvl}{m:1}
 \relationshipbetween{tbldsc}{tblindx}{1:m}
 \relationshipbetween{tbllvl}{tblindx}{1:m}
\end{tabular}
```

`pst-vaucanson-g`: Drawing automata and graphs




```
\usepackage{vaucanson-g}

\begin{VCPicture}{(0,-2)(5.5,2)}
 \State[p]{(0,0)}{A} \State{(2.5,0)}{B}
 \State[r]{(5.5,0)}{C}\Initial{A} \Final{C}
 \EdgeL{A}{B}{a} \ArcL{B}{C}{b}
 \ArcL{C}{B}{b} \LoopN{A}{a} \LoopS{C}{d}
\end{VCPicture}
```

`pst-pdgr`: Medical pedigrees



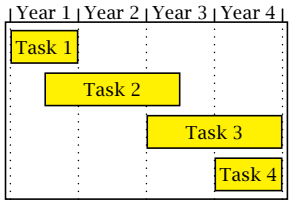

```
\usepackage{pst-pdgr}

\begin{pspicture}(6,6) \psset{belowtextrp=t,armB=1}
 \rput(2.5,5.5){\pstPerson[male,deceased,
  belowtext=A:1]{A:1}}
 \rput(3.5,5.5){\pstPerson[female,deceased,
  belowtext=A:2]{A:2}}
 \pstRelationship[descentnode=A:1_2]{A:1}{A:2}
 \rput(1,3.5){\pstPerson[female,affected,
  belowtext=B:1]{B:1}}
 \pstDescent{A:1_2}{B:1}
 \rput(2,3.5){\pstPerson[male,belowtext=B:2]{B:2}}
 \pstRelationship[descentnode=B:1_2]{B:1}{B:2}
 \rput(3.5,3.5){\pstPerson[male,affected,
  belowtext=B:3]{B:3}}
 \pstDescent{A:1_2}{B:3}
 \rput(4.5,3.5){\pstPerson[female,belowtext=B:4]{B:4}}
 \pstRelationship[descentnode=B:3_4]{B:3}{B:4}
 \rput(5.5,3.5){\pstPerson[female,affected,deceased,
  proband,belowtext=B:5]{B:5}}
 \pstDescent{A:1_2}{B:5}
 \rput(0.5,1.5){\pstPerson[female,belowtext=C:1]{C:1}}
```

```
\pstDescent{B:1_2}{C:1}
\rput(1.5,1.5){\pstPerson[female,belowtext=C:2]{C:2}}
\pstDescent{B:1_2}{C:2}
\rput(2.5,1.5){\pstPerson[female,deceased,
  belowtext={\tabular{c}C:3\\4/52\endtabular}]{C:3}}
\pstDescent{B:1_2}{C:3}
\rput(3.5,1.5){\pstPerson[female,affected,
  belowtext=C:4]{C:4}}
\pstDescent{B:3_4}{C:4}
\rput(4.5,1.5){\pstPerson[male,insidetext=?,
  belowtext=C:5]{C:5}}
\pstDescent{B:3_4}{C:5}
\end{pspicture}
```

pst-light3d: Three dimensional light effects

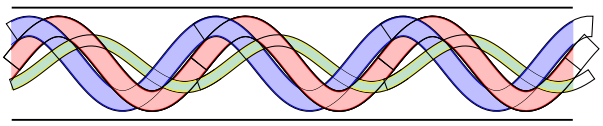
```
\usepackage{pst-light3d}
\DeclareFixedFont{\RM}{T1}{ptm}{m}{n}
                  {2cm}
\psset{linestyle=none,fillstyle=solid,
  fillcolor={[rgb]{1,0.84,0}}}
\PstLightThreeDText
  [LightThreeDXLength=0.5]%
  {\RM\TeX}
```

pst-gantt: Gantt charts




```
\usepackage{pst-gantt}
\begin{PstGanttChart}[yunit=1.5,
 ChartUnitIntervalName=Year,
 ChartUnitBasicIntervalName=Month,
 TaskUnitIntervalValue=12,TaskUnitType=Year,
 ChartShowIntervals]{4}{4}
 \PstGanttTask[TaskInsideLabel={Task 1}]{0}{1}
 \PstGanttTask[TaskInsideLabel={Task 2},
   TaskUnitType=Month]{6}{24}% 24 mon start at 6
  \PstGanttTask[TaskInsideLabel={Task 3}]{2}{2}
  \PstGanttTask[TaskInsideLabel={Task 4}]{3}{1}
\end{PstGanttChart}
```

pst-thick: Very thick lines and curves

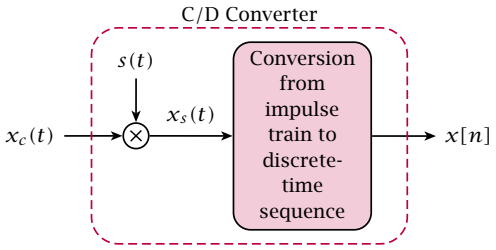


```
\usepackage{pst-thick}
\newpsstyle{thicklinejaune}{fillstyle=solid,
  fillcolor=yellow!50!cyan!50,linecolor=yellow,
  plotpoints=360}
\newpsstyle{thicklinevert}{fillstyle=solid,
  fillcolor=green!50,linecolor=green,plotpoints=360}
\newpsstyle{onlycurvejaune}{linecolor=yellow,
  plotpoints=360}
\def\SinusPhase#1#2#3{%
  /P #1 def /A #2 def /F #3 DegtoRad def
  /O 360 P div def /x0 t def
  /y0 t F add O mul sin A mul def % A*sin(O*t)
  /dx dt def /dy t F add dt add O mul sin
  t F add O mul sin sub A mul def }
\psset{unit=0.5}
\begin{pspicture}(0,-4)(30,4)
\def\motif{\psclip{\psframe[linestyle=none,
  dimen=inner](0,-3)(10,3)}
  \psthick[stylethick=thicklineblue]{-1}{11}%
    {\SinusPhase{10}{2}{90}}
  \psthick[stylethick=thicklinejaune,E=0.5]{-1}{11}%
    {\SinusPhase{10}{1.25}{-100}}
  \psthick{-1}{11}{\SinusPhase{10}{2}{0}}
  \psclip{\psthick[stylethick=vide,E=1.1]%
    {-1}{11}{\SinusPhase{10}{2}{0}}}
    \psthick[stylethick=thicklineblue]{0}{3}%
      {\SinusPhase{10}{2}{90}}
  \endpsclip
  \psclip{\psthick[stylethick=vide,E=1.1]{0}{11}%
    {\SinusPhase{10}{2}{90}}}
    \psthick{5}{9}{\SinusPhase{10}{2}{0}}
  \endpsclip
  \psclip{\psthick[stylethick=vide,E=0.6]{0}{11}%
    {\SinusPhase{10}{1.25}{-100}}}
    \psthick[stylethick=thicklineblue]{7}{9}%
      {\SinusPhase{10}{2}{90}}
  \endpsclip
  \psclip{\psthick[stylethick=vide,E=1.1]{0}{10}%
    {\SinusPhase{10}{2}{0}}}
    \psthick[stylethick=thicklinejaune,E=0.5]{7}{11}%
      {\SinusPhase{10}{1.25}{-100}}
  \endpsclip
  \psclip{\psthick[stylethick=vide,E=1.1]{0}{11}%
    {\SinusPhase{10}{2}{0}}}
    \psthick[stylethick=thicklinejaune,E=0.5]{-0.5}{1}%
      {\SinusPhase{10}{1.25}{-100}}
  \endpsclip
\endpsclip}%
\motif\rput(10,0){\motif}\rput(20,0){\motif}
\psline[linewidth=0.1](0,3)(30,3)
\psline[linewidth=0.1](0,-3)(30,-3)
\end{pspicture}
```

pst-mirror: Projections on a spherical mirror

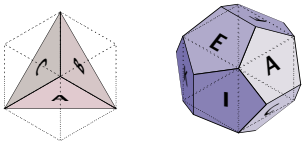

```
\usepackage{pst-mirror,pst-grad}

\psscalebox{0.7}{\begin{pspicture}(-7,-7)(7,7)
 \newpsstyle{GradWhiteYellow}{fillstyle=gradient,
 gradbegin=yellow,gradend=yellow!20,linecolor=yellow!50,
 GradientCircle=true,gradmidpoint=0,GradientPos={(1,1)}}
 \pscircle[style=GradWhiteYellow]{7.07}
 \pstSphereGrid[linecolor=red,grille=10,Ymin=-50,
   Ymax=50,Xmax=80,Xmin=-80,normale=0 0](20,0,0)
 \pstSphereGrid[linecolor=blue,grille=10,Ymin=-40,
   Ymax=-20,Xmax=80,Xmin=-80,normale=0 90](40,0,-10)
 \pstTextSphere[fillstyle=solid,fillcolor=red,
   normale=0 0,fontscale=40,PSfont=Time-Roman,yO=0]
   (20,0,10){pst-mirror}
 \pstTextSphere[fillstyle=solid,fillcolor=black,
   normale=0 0,fontscale=20,PSfont=Helvetica,yO=0]
   (20,0,35){PSTricks}
 \pstTextSphere[fillstyle=solid,fillcolor=blue,
   normale=0 90,fontscale=10,PSfont=Helvetica,yO=2.5]
   (10,0,-10){A Spherical Mirror}
\end{pspicture}}
```

pst-sigsys: Signal processing



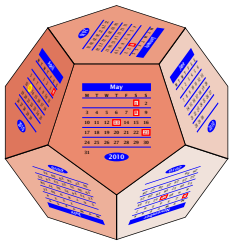

```
\usepackage{pstricks} \usepackage[pstadd]{pst-sigsys}

\begin{pspicture}(-2,-2)(5.5,2)
 \rput(-1.75,0){\rnode{xc}{$x_c(t)$}}
 \pscircleop[operation=times](0,0){otimes}
 \rput(0,1.25){\rnode{s}{$s(t)$}}
 \psblock[fillstyle=solid,fillcolor=purple!20]%
   (2.75,0){conv}{\parbox[c]{2\psunit}%
   {\centering Conversion from impulse
     train to discrete-time sequence}}
 \rput(5.5,0){\rnode{x}{$x[n]$}}
 \psset{style=Arrow}
 \ncline[nodesepA=.15]{xc}{otimes}
 \ncline[nodesepA=.15]{s}{otimes}
 \ncline{otimes}{conv}\naput{$x_s(t)$}
 \ncline[nodesepB=.15]{conv}{x}
 \psframe[linecolor=purple,linestyle=dashed,
   style=Dash](-.75,-1.5)(4.5,1.5)
 \rput(1.875,1.75){C/D Converter}
\end{pspicture}
```

pst-platon: Platonic solids



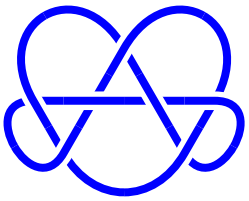
```
\usepackage{pst-platon}

\psTetrahedron\qquad
\psDodecahedron
```

pst-calendar: Two or three dimensional calendars



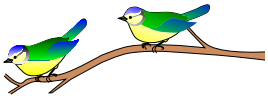
```
\usepackage{pst-calendar}

\begin{Example}[ltxps]{}
\psscalebox{0.13}{ %
  \psCalDodecaeder[Jahr=2010,
    style=march]}
```

pst-knot: Knot lines



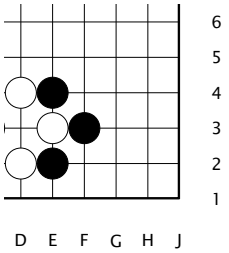
```
\usepackage{pst-knot}

\begin{pspicture}(-2,-2)(2,2)
\psKnot[linewidth=3pt,
  linecolor=blue](0,0){7-7}
\end{pspicture}
```

pst-fun: Funny objects



```
\usepackage{pst-fun}

\begin{pspicture}(0,-1.3)(10,3)
 \psBird[Branch]
 \rput{-20}(4,1.8){\psBird}
\end{pspicture}
```

psgo: The game of Go




```
\usepackage{psgo}

\psscalebox{0.7}{%
\begin{psgopartialboard}[9]{(4,1)(9,6)}
\stone{white}{c}{3} \stone{white}{e}{3}
\stone{white}{d}{2} \stone{white}{d}{4}
\stone{black}{f}{3} \stone{black}{e}{2}
\stone{black}{e}{4}
\end{psgopartialboard}}
```

Plotting experimental data using pgfplots

Joseph Wright

Abstract

Creating plots in TEX is made easy by the pgfplots package, but getting the best presentation of experimental results still requires some thought. In this article, the basics of pgfplots are reviewed before looking at how to adjust the standard settings to give both good looking and scientifically precise plots.

1 Introduction

Presenting experimental data clearly and consistently is a crucial part of publishing scientific results. A key part of this is the careful preparation of plots, graphs and so forth. Good quality plots often make results clearer and more accessible than large tables of numbers. A number of tools specialise in producing plots for scientific users, both commercial (such as Origin and SigmaPlot) and open source (for example QtiPlot and SciDAVis). The output of these programs is impressive, but TEX users may find that it lacks the 'polish' that TEX can provide.

There are a few approaches to producing plots directly within a TEX document, but perhaps the easiest method to use the pgfplots package (Feuersänger, 2010). This is an extension of the very popular pgf graphics system (Tantau, 2008), which as many readers will know works equally well with the traditional DVI-based work flow and the increasingly popular direct production of PDF output. pgfplots also works with plain TEX, LATEX and ConTEXt, meaning that it is an accessible route for almost all TEX users.

As with any tool, getting the best results with pgfplots does require a bit of understanding. In this article, I'm going to look at getting good results for plots of experimental data. This includes things like worrying about units and how to get the graphics out of TEX for publishers who require it.

In the examples, I am going to use real experimental data from the research group I work in,[1] and explain how I've presented this for publication. The aim is to show pgfplots in use 'in the wild', rather than the usual approach of somewhat contrived sets of data. The examples do not aim to be a complete survey of the power of pgfplots: its manual is excellent and covers all of the options in detail.

2 The basics

There are some basics that need to be in place before plots can be created. First, the code for pgfplots needs to be loaded. This is designed to work equally well with TEX, LATEX and ConTEXt, and of course the loading mechanism depends on the format:

```
\input pgfplots.tex   % Plain TeX
\usepackage{pgfplots} % LaTeX
\usemodule[pgfplots]  % ConTeXt
```

Plots are created inside an '`axis`' environment, which itself needs to be inside the pgf environment '`tikzpicture`'. The differences between the three formats again mean that there are again some variations. So, for plain TEX:

```
\tikzpicture
  \axis
    % Plot code
  \endaxis
\endtikzpicture
```

while for LATEX:

```
\begin{tikzpicture}
  \begin{axis}
    % Plot code
  \end{axis}
\end{tikzpicture}
```

and for ConTEXt:

```
\starttikzpicture
  \startaxis
    % Plot code
  \stopaxis
\stoptikzpicture
```

In the examples in this article I'll leave out the wrapper and concentrate just on the plot code, which is the same for all three formats.

If the wrapper is given with no content at all then pgfplots will fill in some standard values. This gives an empty plot (Figure 1). To actually have something appear, data has to be added to the axes. This is done using one or more `\addplot` instructions, which have a flexible syntax which can be applied to a wide range of cases.

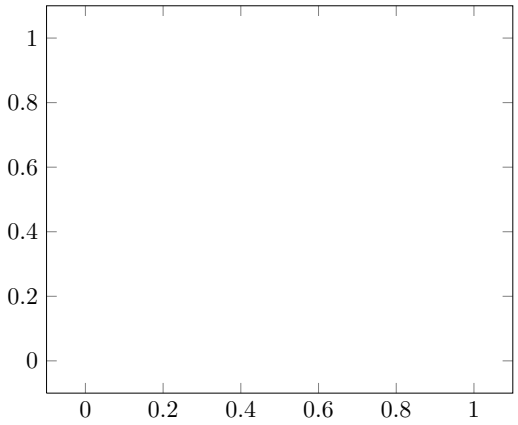


Figure 1: An empty set of axes

[1] My supervisor is Professor Christopher Pickett: http://www.uea.ac.uk/che/pickettc

Table 1: Rates of a chemical reaction [Data taken from Jablonskytė, Wright, and C. J. Pickett (2010)].

Concentration / mmol dm^{-3}	Rate / s^{-1}
338.1	266.45
169.1	143.43
84.5	64.80
42.3	34.19
21.1	9.47

Before starting to produce some real plots, there is one minor thing to set up. The most recent version of pgfplots (1.3) makes a number of improvements to the standard settings for the package, but only if told to. For new plots, it makes sense to use these improved abilities using

```
\pgfplotsset{compat = newest}
```

in the source. In this article, I've included this line in the preamble for the source (which is in LaTeX).

3 Small data sets

Small sets of data, for example values calculated by hand, can be plotted by including the data directly in the TeX source. As an example set of data, I am going to use the rate of a chemical reaction, recently reported by the research group I work in (Table 1). As shown, the rate of a chemical reaction depends on the concentration of one of the 'ingredients', which as a table is suggestive but not really immediately accessible.

This can be plotted rapidly using the `\addplot` macro along with the `coordinates` keyword, with the input reading

```
\addplot coordinates {
  ( 338.1, 266.45 )
  ( 169.1, 143.43 )
  (  84.5,  64.80 )
  (  42.3,  34.19 )
  (  21.1,   9.47 )
};
```

(placed inside the `axis` environment, as explained above). This shows a number of features of the pgfplots input syntax. First, after the primary macro (`\addplot`) a keyword is used to direct the behaviour of the code. Second, there is no need to worry about white space, which makes it easier to read the input. Third, as with other pgf commands, the entire `\addplot` input is terminated by a semi-colon. This will allow multiple plots to use the same axes, as will be demonstrated later.

Using the input above also reveals why it is necessary to think about plots, even with a powerful system like pgfplots. Using the simple input I've given results in Figure 2. There are a number of issues with this initial plot. Most obviously, the data points should not be plotted in a 'join the dots' fashion. Much better would be independent points along with a best fit line showing the trend. The plot is also in colour, which for a simple plot like this one is not really necessary. Publishers prefer black and white unless colour is adding real information.

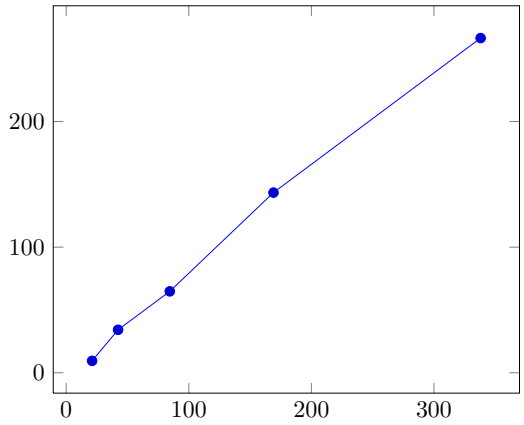


Figure 2: A raw plot based on the data in Table 1.

As with other parts of the pgf system, pgfkeys uses key–value arguments to alter settings. Here, the options apply to a particular plot, and so are given as an optional argument to the `\addplot` macro in square brackets (irrespective of the format in use). So the appearance of the plot can be altered by adding an optional argument and suitable settings to the `\addplot` macro.

```
\addplot[
  color = black,
  fill  = black,
  mark  = *, % A filled circle
  only marks
]
```

These keys all have self-explanatory names: the pgfplots manual of course gives full details. It is possible to use '`black`' for the combination of `color = black` and `fill = black`; here, I will stick to the longer version including the key names, as it makes the meaning a little clearer.

Adding a best fit line means a second `\addplot` is needed. TeX is not the best way to do general mathematics, and so the fitting was done using a spreadsheet application. The easiest way to add a line is to specify the two ends and show only the line:

```
\addplot[
  color = black,
  mark  = none
]
```

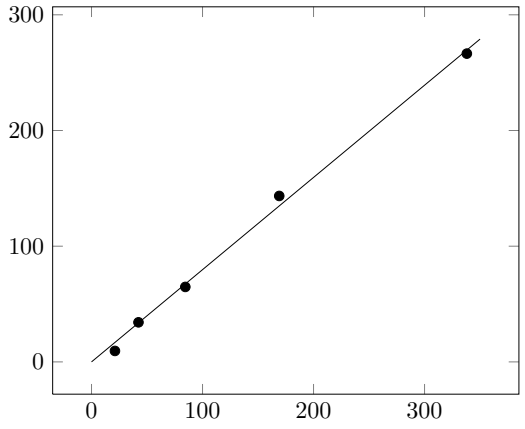


Figure 3: Second version of a plot based on the data in Table 1.

```
  coordinates {
   (   0,   0 )
   ( 350, 279 )
  };
```

Making these first set of changes gives Figure 3, which already looks more professional.

There is still more to think about before the plot is finished. The axes do not start from 0, but instead from a bit below zero: hardly very clear, and certainly not needed here. Much more importantly, the axes are not labelled and there are no units. These are both problems about the entire plot, not just one data set. So in this case the optional argument to the `axis` environment comes into play. This follows the start of the environment in square brackets (recall that the start of the environment is format-dependent, as shown above). Once again the various option names are pretty self-explanatory:

```
[
  xlabel = Concentration\,/\,mmol\,dm$^{-3}$,
  xmax   = 400,
  xmin   = 0,
  ylabel = Rate\,/\,s$^{-1}$,
  ymax   = 300,
  ymin   = 0
]
```

The result of these adjustments is Figure 4. The contents of the `axis` environment is now:

```
[
  xlabel = Concentration\,/\,mmol\,dm$^{-3}$,
  xmax   = 400,
  xmin   = 0,
  ylabel = Rate\,/\,s$^{-1}$,
  ymax   = 300,
  ymin   = 0
]
```

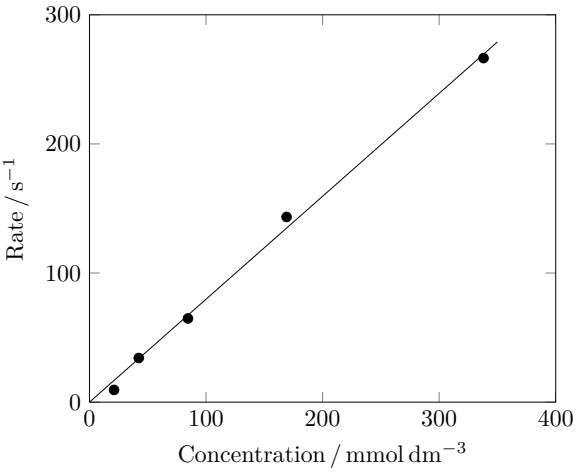


Figure 4: Final version of a plot based on the data in Table 1.

```
\addplot[
    color = black,
    fill  = black,
    mark  = *,
    only marks
  ] coordinates {
    ( 338.1, 266.45 )
    ( 169.1, 143.43 )
    (  84.5,  64.80 )
    (  42.3,  34.19 )
    (  21.1,   9.47 )
  };
\addplot[
  color = black,
  mark  = none
] coordinates {
   (   0,   0 )
   ( 350, 279 )
  };
```

The plot now looks good, but there are a few other points to note before moving on to more complex challenges. In a real publication, the caption of the figure should give any necessary experimental details about the data presented. Also notice that both axes of the plot are in simple whole numbers. For this plot (and Table 1), the original concentrations were in mol dm^{-3}, and were multiplied by 1000 to make them whole numbers. Usually this approach makes the final result much easier to read (even if it requires more effort initially). pgfplots can do simple mathematics, but I tend to favour doing the calculations first in an external tool and stick to using TEX for its strength: typesetting.

The ideas used for the preceding plot can readily be applied to presenting several sets of data on the same axes. pgfplots allows the user to pick a number

of markers to differentiate the plots, and to include legends and so forth. However, it rapidly becomes unwieldy to have the raw data in the TEX source when there are a number of curves to plot. Luckily, pgfplots has an alternative `\addplot` syntax that help out.

4 Large sets of data

As the number of data points to plot grows, adding the information directly to the source rapidly becomes laborious and error-prone. Almost always, the data will be available in a structured format, and so reading an external file becomes a much more efficient way to proceed. pgfplots can read data tables from, for example, tab-separated text files, which can conveniently be written by standard data handling software. There are two approaches to reading such data: loading the entire table into a macro which can then be used for plotting, or reading the data as part of the `\addplot` line. The most appropriate method depends on the context: both methods will be illustrated here.

4.1 One set of axes, several plots

When several related sets of data need to be presented on one set of axes, it rapidly becomes easier to use external data files even if each plot has only a small number of points.

Here, I will illustrate the methods using the change in amount of four chemicals over time. The data here are available as a single text file, and there are not too many time points. It therefore makes most sense to load all of the data into a macro, and to use it to create each plot in turn. With the numbers saved in a file called `data-set-two.txt`, the reading instruction is

```
\pgfplotstableread{data-set-two.txt}
  \datatable
```

Here, the name of the storage macro (`\datatable`) is arbitrary: in a complex document it would probably be more descriptive.

In the table here each column has a header for ease of reference, with the first few rows reading

```
Time  a   b   c   d
0     49  7   41  1.3
67    55  9   33  1.6
134   61  10  26  1.9
200   65  12  20  1.9
...
```

The columns can be referred to either by their header name (‘`y =` ⟨*header name*⟩’) or by their position in the table (‘`y index =` ⟨*header index*⟩’), with column ‘`a`’ here being the y column with index 1, ‘`b`’ with index 2, and so on. It’s also possible to include comment lines in the data file, which can be used for things like the units or reference numbers for the raw data.

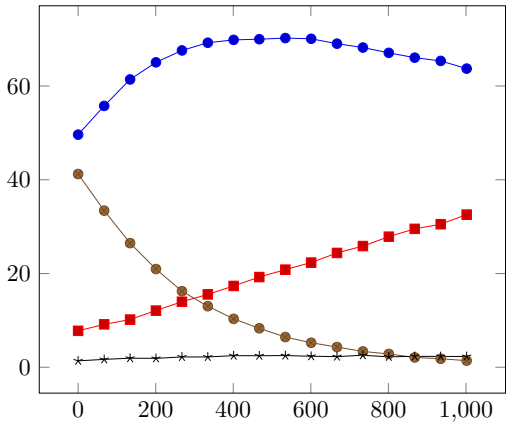


Figure 5: Raw plot using a pre-loaded data table.

The plot can then be created using the ‘`table`’ keyword after the `\addplot` macro:

```
\addplot table[y = a] from \datatable ;
\addplot table[y = b] from \datatable ;
\addplot table[y = c] from \datatable ;
\addplot table[y = d] from \datatable ;
```

Here, the second keyword ‘`from`’ indicates that a macro will be used to supply the data to plot. With no formatting changes the result is Figure 5. As with the first data set, it is better not to ‘join the dots’. As this applies to the entire plot, `only marks` can be given in the optional argument to the start of the `axis` environment, rather than repeating the same instruction for each `\addplot` macro. There also needs to be a legend to tell the reader which curve is which. Using the `\addlegendentry` macro after each `\addplot` line will automatically gather the necessary information (symbol type and colour), and will result in a legend being added to the plot:

```
\addplot table[y = a] from \datatable ;
\addlegendentry{Compound \textbf{a}}  ;
\addplot table[y = b] from \datatable ;
\addlegendentry{Compound \textbf{b}}  ;
 ...
```

The ideas about labelling axes and setting an appropriate scale which were necessary for the first plot also apply here. Making these adjustments leads to Figure 6.

Once again, colour is not really necessary here if care is taken with the rest of the plot. The visual difference between the different markers should be enough to show which curve is which. When preparing the real diagram for publication, my supervisor asked for all of the symbols to be circles, filled

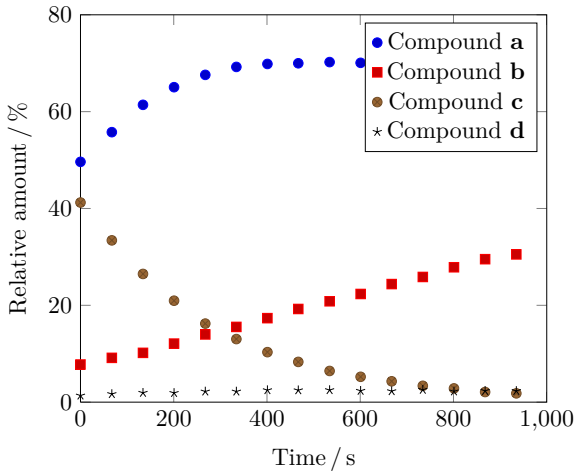


Figure 6: Second version of a plot from a pre-loaded data table.

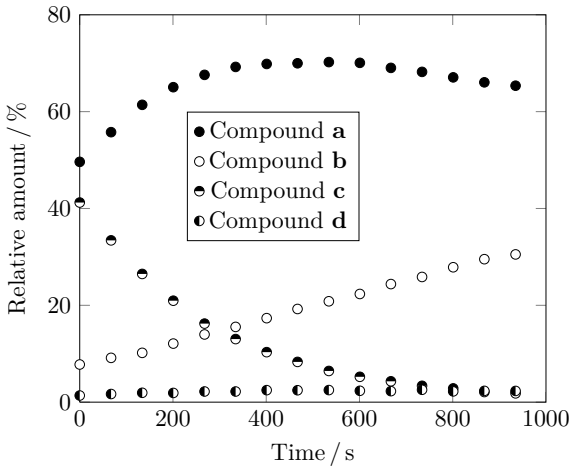


Figure 7: Final version of a plot from a pre-loaded data table.

in different ways. That needed a bit of help from `comp.text.tex` to get half-filled circles, which can be achieved by creating a special marker using some lower level pgf code:

```
\pgfdeclareplotmark{halfcircle}{%
  \begin{pgfscope}
    \pgfsetfillcolor{white}%
    \pgfpathcircle{\pgfpoint{0pt}{0pt}}
      {\pgfplotmarksize}
    \pgfusepathqfillstroke
  \end{pgfscope}%
  \pgfpathmoveto
    {\pgfpoint{\pgfplotmarksize}{0pt}}
  \pgfpatharc{0}{180}{\pgfplotmarksize}
  \pgfpathclose
  \pgfusepathqfill
}
```

(don't worry too much about this; for myself, I just accept that it works!). Including the above code in the source means that `halfcircle` can be used as a value for the `mark` key:

```
\addplot[mark = halfcircle, ...
```

The filled portion can be moved 'around' the circle by rotating the mark

```
\addplot[
  mark         = halfcircle,
  mark options = {rotate = 90},
```

There are a couple of other issues with Figure 6. Most obviously, the legend is covering some of the data points, while there is a handy space right in the middle. This can be fixed by asking pgfplots to move the entire legend box using the `legend style` key, which is used in the optional argument to the `axis` environment. It takes a bit of experimentation to get the correct position; in this case

```
[
  legend style = { at = {(0.6,0.75)}}
]
```

seems to be about right.

The x axis label for 1000 includes a comma as a digit separator: that is not usual in publications in English. So there is a slight modification to make to the digit formatting routine: this is a general pgf setting:

```
\pgfkeys{
  /pgf/number format/
    set thousands separator =
}
```

Making these changes, and adjusting a few minor settings to get the colours correct leads to the final version of the plot (Figure 7). Here is the code:

```
[
  legend style = { at = {(0.6,0.75)}},
  only marks,
  xlabel = Time\,/\,s,
  xmax   = 1000,
  xmin   = 0,
  ylabel = Relative amount\,/\,\%,
  ymax   = 80,
  ymin   = 0
]
\addplot[
  color = black,
  mark  = *
] table[y = a] from \datatable ;
\addlegendentry{Compound \textbf{a}} ;
\addplot[
  color = black,
  fill  = white,
  mark  = *
] table[y = b] from \datatable ;
```

```
\addlegendentry{Compound \textbf{b}} ;
\addplot[
  color        = black,
  mark         = halfcircle
] table[y = c] from \datatable ;
\addlegendentry{Compound \textbf{c}} ;
\addplot[
  color        = black,
  mark         = halfcircle,
  mark options = {rotate = 90}
] table[y = d] from \datatable ;
\addlegendentry{Compound \textbf{d}} ;
```

4.2 An experimental spectrum

One common technique in scientific research is recording *spectra*: how a sample absorbs light, microwaves, radio waves, *etc.* Often, these are published by simply taking the raw print out from the control program and pasting it into the article, which does not make for a good appearance. So replotting with pgfplots is a good idea.

Spectra tend to involve a lot of numbers, and so using an external table is once again a good idea. As these are large files that will only be used once, loading to a macro is not efficient. Instead, the data table can be loaded as part of the `\addplot` line, with the syntax

```
\addplot table {data-set-three.txt};
```

The first version of the plot in this case (Figure 8) already includes a number of the refinements discussed in the first two examples. The example shows data from a technique called 'nuclear magnetic resonance', in which radio waves are absorbed by a sample. In the plot, the x axis is related to the frequency of the radio waves, while the y axis shows how much is absorbed.

The plot looks generally good, but there are some adjustments required. First, the y scale is essentially arbitrary, and so is usually not given any values at all. This can be achieved by setting the `yticklabels` option to an empty value in the optional argument to the `axis` environment:

```
[yticklabels  = ]
```

Second, for various historical reasons it is normal to plot the x axis backward (running high to low). The latest version of pgfplots can do this automatically using the `x dir` option.

```
\addplot[
  x dir = reverse,
  ...
```

These changes give an improved version of the plot (Figure 9).

The final thing this plot needs is some 'peak labels': markers showing the exact value at the top of each peak. For this, the pgf `\node` macro can be used. A `\node` can be used to put material anywhere on a plot, and each node takes a range of options to control its appearance. In this case, the node itself is going to be completely empty, and a `pin` will be used to point to the node. A pin generates a short line from the node to some associated text, which in this case I want to rotate by 90° to pack the labels in. The syntax ends up a little bit complicated, as various options need to be correct, for example:

```
\node[
  coordinate,
  pin = {[rotate=90]right:22.26}
] at (axis cs:22.26,1.1) { };
```

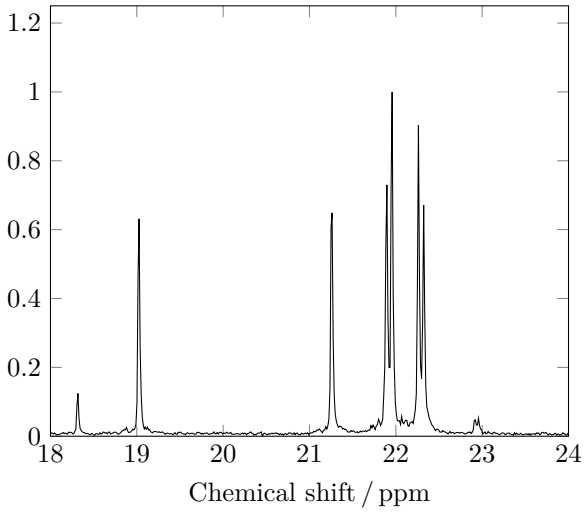


Figure 8: First version of a single spectrum.

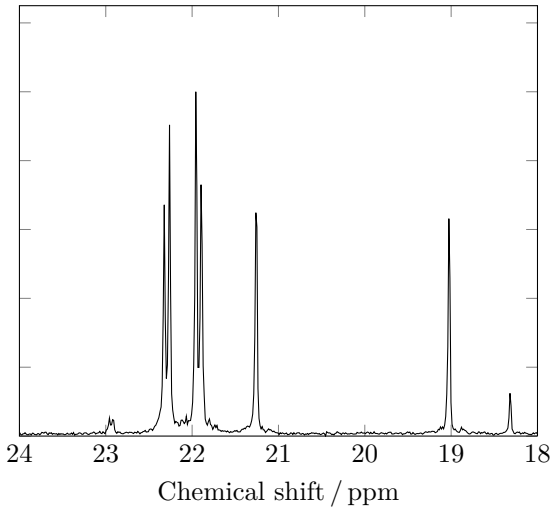


Figure 9: Second version of a single spectrum.

The `right` keyword here can be replaced by an angle, which can then be used to be slightly off a peak's position, for example

```
\node[
  coordinate,
  pin = {[rotate=90]5:22.32}
] at (axis cs:22.32,1.1) { };
```

Once again, putting everything together leads to the completed plot (Figure 10).

```
[
  x dir       = reverse,
  xlabel      = Chemical shift\,/\,ppm,
  xmin        = 18,
  xmax        = 24,
  ymax        = 1.75,
  ymin        = 0,
  yticklabels =
]
\addplot[
  color = black,
  mark  = none
] table from {data-set-three.txt};
\node[
  coordinate,
  pin = {[rotate=90]5:22.32}
] at (axis cs:22.32,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]right:22.26}
] at (axis cs:22.26,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]right:21.96}
] at (axis cs:21.96,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]-5:21.90}
] at (axis cs:21.90,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]right:21.26}
] at (axis cs:21.26,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]right:19.03}
] at (axis cs:19.03,1.1) { };
\node[
  coordinate,
  pin = {[rotate=90]right:18.32}
] at (axis cs:18.32,1.1) { };
```

4.3 Changes over time

The previous example demonstrated how to plot a single spectrum from an external file. In a final example, I want to look at going beyond this to showing how experimental data changes over time. This means plotting a series of spectra on the same

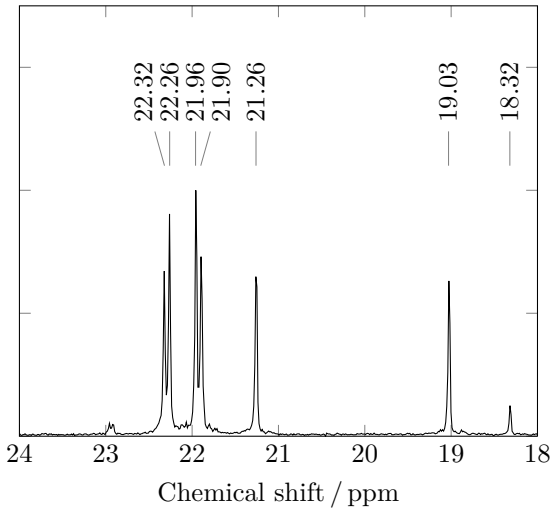


Figure 10: Final version of a single spectrum.

axes, and finding a good way to show which way time is running. Some people choose to use a 'three-dimensional' plot for this scenario, and pgfplots includes the ability to generate this type of output. However, experience suggests to me that there is more value in a well-constructed two-dimensional plot than a three-dimensional representation of the same data. The challenge is therefore to find the best way to represent the results on paper.

In this example, the raw data is how a sample absorbs infra-red light (heat). There is a reaction taking place, and so the absorption will change over time. The initial results simply show a series of peaks, a bit like Figure 10. The changes are quite subtle compared to the overall scale, so the first stage in creating a plot is not TEX related. Using a spreadsheet it's possible to find how the signal changes relative to the one at the start of the experiment: this gives a 'difference spectrum' for each time. That can then be saved as a text file which can be used as the input to pgfplots.

In this case, a single file contains all of the data for the plot. To get each `\addplot` line to use a single time, the optional argument to the `table` keyword is used, for example:

```
\addplot[mark = none] table[y index =  1]
  {data-set-four.txt};
\addplot[mark = none] table[y index =  2]
  {data-set-four.txt};
  ...
```

As there are many almost identical lines, the pgf `\foreach` macro can be used to vary only the y index used

```
\foreach \yindex in {1,2,...,20}
  \addplot[mark = none]
```

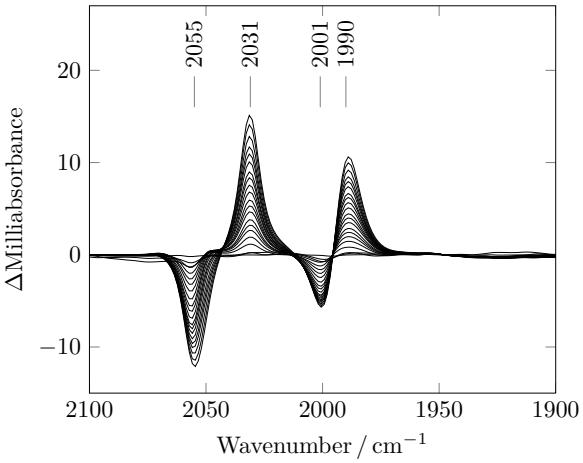


Figure 11: First plot of a change over time.

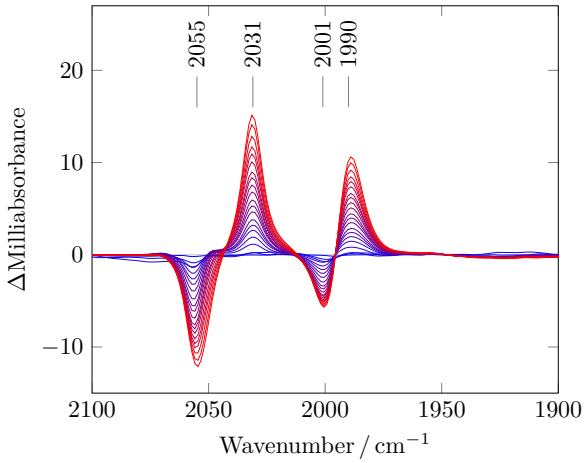


Figure 12: Second plot of a change over time (time runs blue to red).

```
  table[y index = \yindex]
    {data-set-four.txt};
```

A bit of experimentation showed that around 20 lines gave a good appearance for the plot (the full data set has nearly 200 time points!). The `\foreach` syntax makes it easy to show an evenly-spaced subset of the available points, by setting the gap between selected columns to be larger:

```
\foreach \yindex in {1,10,...,189}
  \addplot[mark = none]
    table[y index = \yindex]
      {data-set-four.txt};
```

This uses every 9th column for the plot, and in the example results in 19 separate curves (Figure 11).

There is one obvious problem with the plot: which way is time running? Here, careful use of colour can be used in a way which really does add to the information imparted by the plot. To do this, a 'plot cycle' is needed to specify the colour used for each plot. There are some built in to pgfplots, but one is also easy to construct:

```
\pgfplotscreateplotcyclelist
  {blue to red}{%
  color = red!0!blue\\%
  color = red!5!blue\\%
  color = red!10!blue\\%
  color = red!15!blue\\%
  color = red!20!blue\\%
  color = red!25!blue\\%
  color = red!30!blue\\%
  color = red!35!blue\\%
  color = red!40!blue\\%
  color = red!45!blue\\%
  color = red!50!blue\\%
  color = red!55!blue\\%
  color = red!60!blue\\%
  color = red!65!blue\\%
  color = red!70!blue\\%
  color = red!75!blue\\%
  color = red!80!blue\\%
  color = red!85!blue\\%
  color = red!90!blue\\%
  color = red!95!blue\\%
  color = red!100!blue\\%
}
```

It's important to note that the end of lines here must end in comments. With that cycle available, the option `cycle list name` can be used for the axes: this will mean that the first plot will be pure blue, the second 95 % blue and 5 % red, and so on.

The other minor adjustment to make is to include a line at for $y = 0$. This can be done by including an extra 'tick':

```
[
  extra y ticks       = 0,
  extra y tick labels = ,
  extra y tick style  =
    { grid = major }
]
```

(The '`extra y tick labels = ,`' line makes sure that the label for 0 is not printed twice, as this gives a slightly 'bold' appearance.)

These two adjustments lead to Figure 12:

```
[
  cycle list name     = blue to red,
  extra y ticks       = 0,
  extra y tick labels = ,
  extra y tick style  = { grid = major },
  x dir               = reverse,
  xlabel              = Wavenumber\,/\,cm$^{-1}$,
  xmin                = 1900,
  xmax                = 2100,
  ylabel              = $ \Delta $Milliabsorbance,
```

```
  ymax                = 27,
  ymin                = -15
]
\foreach \yindex in {1,10,...,189}
  \addplot table[y index = \yindex]
    {data-set-four.txt};
\node[
  coordinate,
  pin = {[rotate=90]right:1990}
] at (axis cs:1990,16) { };
\node[
  coordinate,
  pin = {[rotate=90]right:2001}
] at (axis cs:2001,16) { };
\node[
  coordinate,
  pin = {[rotate=90]right:2031}
] at (axis cs:2031,16) { };
\node[
  coordinate,
  pin = {[rotate=90]right:2055}
] at (axis cs:2055,16) { };
```

As with the other plots, it's important to include details about the experiment somewhere close to the figure: usually the caption is a good place for this. In the published article, I included details about what I used as time zero, the time range and concentrations for the experiment in the caption.

5 Exporting plots from TEX

Processing a large number of plots in TEX can lead to the program running out of memory. The ability to set up plots as separate files which can then be included in the main document as graphics is therefore important. At the same time, some publishers require all graphics to be available as stand-alone files, which again means finding a way to export plots from TEX.

pgfplots includes methods for carrying out this process in an automated fashion. The latest release of pgfplots makes the process much easier than was previously the case, and includes full instructions on how to proceed. When using pdfTEX the lines

```
\usetikzlibrary{pgfplots.external}
\tikzexternalize{<filename>}
```

will instruct pgfplots to make each plot into a separate PDF file. To create PostScript files the additional command:

```
\tikzset{external/system call =
  {latex -shell-escape -halt-on-error
  -interaction=batchmode -jobname "\image"
  "\texsource"; dvips -o "\image".ps
  "\image".dvi}}
```

is needed, and the main file should be typeset in DVI mode. In both cases `\write18` (external command execution) needs to be enabled for the process to work correctly.

6 Conclusions

Producing scientific plots using pgfplots can produce high quality output with relatively little effort for the user. To do so, some thought about both the information to be reported and the appearance of the output is needed.

7 Acknowledgements

Thanks to Stefan Pinnow for a number of useful improvements both to the code and to the text while drafting this article.

The published versions of the figures used here originally appeared in Wright and Pickett (2009) and Jablonskytė, Wright, and C. J. Pickett (2010). They are reproduced by permission of The Royal Society of Chemistry.

References

Feuersänger, Christian. "pgfplots". `http://mirror.ctan.org/graphics/pgf/contrib/pgfplots`, 2010.

Jablonskytė, Aušra, J. A. Wright, and C. J. Pickett. "Mechanistic aspects of the protonation of [FeFe]-hydrogenase subsite analogues". *Dalton Transactions* (39), 3026–3034, 2010.

Tantau, Till. "The Ti*k*Z and pgf Packages". `http://mirror.ctan.org/graphics/pgf`, 2008.

Wright, Joseph A., and C. J. Pickett. "Protonation of a subsite analogue of [FeFe]-hydrogenase: mechanism of a deceptively simple reaction revealed by time-resolved IR spectroscopy". *Chemical Communications* (38), 5719–5721, 2009.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph.wright (at)
morningstar2.co.uk

From Logo to MetaPost

Mateusz Kmiecik

Abstract

The Logo language (turtle graphics) is recommended for teaching of Information Technology at secondary schools in Poland. It is quite a primitive language so young people quickly get discouraged while programming more advanced pictures. Could MetaPost be used to this end?

1 Why have I chosen this topic?

When I was preparing for the Malopolska Computing Competition, I had to learn about many domains concerning computers. One of them was the Logo language. Before the competition I had noticed one of the tools which my dad [Jacek Kmiecik] had been using to create pictures. It was MetaPost. Dad suggested that I should compare and describe MetaPost and Logo in a BachoTEX presentation. But I had one problem with the topic of this presentation. After consideration I have chosen 'From Logo to MetaPost' because I created the same pictures in MetaPost as in Logo. Both languages are used in making graphics but I want to show that MetaPost is superior.

2 Logo

2.1 Introduction

Logo is a programming language which supports the development of creative and logical thinking as well as algorithmic abilities. In Poland, since the second half of the 1980s Logo has been used as a teaching tool in mathematics and computing lessons [1].

The graphic symbol of Logo is a turtle. Logo uses a specific geometry — 'turtle geometry'. Thanks to the symbol of a turtle which we can move and turn on the screen and which leaves signs, we can create pictures.

The program called Logo Komeniusz is a didactic tool which helps with using the Logo language. It has Polish commands which are very useful to help students comprehend the basic commands necessary for drawing.

Of course there are other programs to help us understand the Logo language but in my opinion Logo Komeniusz is the best one. This program is a commercial one, but its price is adjusted to the group of students who need it.

2.2 Beginnings

At first I was making easy pictures, such as a triangle or a square. Then I made my first procedure — a command creating a chosen polygon. After I absorbed the basics of Logo, I started creating more complicated figures, even whole pictures.

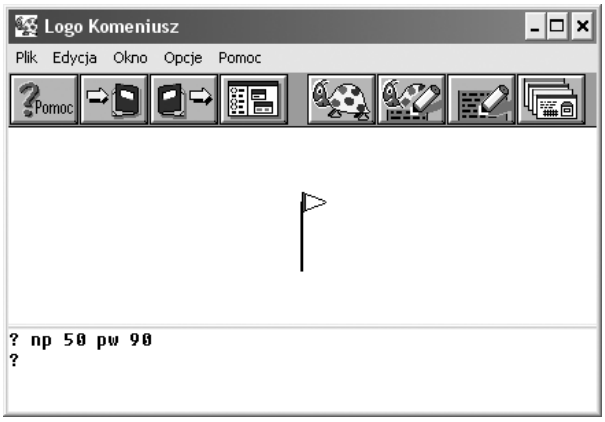


Figure 1: The window of Logo Komeniusz program

2.3 An example of making pictures

Commands are inserted in the bottom part of the Logo Komeniusz window. When you put commands and press `Enter` the results will be shown in the top part (fig. 1).

These commands will make the turtle go 50 steps ahead and turn around 90 degrees:

```
np 50
pw 90
```

`np` is short for `naprzód` which in English means `towards`, and `pw` is short for `prawo` which in English means `right`.

If we want to jump to another place we have to use commands `podnieś` (`lift` in English) and move the turtle to the proper place. To start drawing a picture again we have to use command `opuść` (`drop`).

```
pod   ==> podnieś
np 20 ==> naprzód 20
opu   ==> opuść
```

It is worth mentioning that Logo Komeniusz uses a Polish dialect which I believe makes using it much easier — but there are many contradictory opinions.

3 MetaPost

3.1 Introduction

When I was making pictures in Logo, I had lots of problems with it. My dad said that after the competition I should draw the same figures in MetaPost and then I would see the difference. I didn't think that it would be easier programming with paths (example: fig. 3). And there I noticed the main difference

between Logo and MetaPost: Logo works by controlling an object, while MetaPost works on defining a path (opened or closed), which we can fill with color or not and put it at the proper place on the virtual "paper".

3.2 Beginnings

Everybody who starts to learn MetaPost with the original book begins with a picture of a triangle with a line (fig. 2). Not wanting to break this rule I began my MetaPost adventure with it too.

Figure 2: My first MetaPost picture (based on [3], page 4)

I was studying MetaPost by taking to pieces examples which I found on the Internet [5, 6, 7]. My dad took the role of teacher and explained many useful functions to me. I have to admit that I had quite large problems getting out of the habit of "using the turtle" (the method of working in Logo).

3.3 An example of making pictures

To make a picture with MetaPost, at first you have to define coordinates (x, y) of each point (*pair*) — in the example there will be four points (fig. 3):

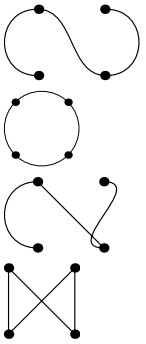
$\bullet B$ $\bullet D$

$\bullet A$ $\bullet C$

```
pair A, B, C, D;
A = (0,0);    B = (0,10mm);
C = (10mm,0); D = (10mm,10mm);
```

Figure 3: Four points and the code to produce them

Next, I show four examples — how we can define paths between those four points and make four different shapes: fig. 4.



```
draw A..B..C..D;
```

```
draw A..B..D..C..cycle;
```

```
draw A{left}..{right}B
   --C{left}..{left}D;
```

```
draw A--B--C--D--cycle;
```

Figure 4: Examples of paths through the four points of fig. 3

To edit MetaPost files we only need a good text editor. To then create graphics we have to run the command `mpost example.mp` where `example` is the name of our input file. Then we will get the output `example.1` where the `1` refers to the number of a particular figure defined in the MetaPost file. To see the image we can put it into a LaTeX file or use the command `mptopdf` which will convert a file `example.1` to PDF.

4 Examples

In this section I'm going to show the examples which I did both in MetaPost and in Logo.

Triangle (fig. 5)

Logo version:

```
powtórz 3 [ npw 20 pw 120 ]
```

MetaPost version:

```
pair A,B,C;
A = (1cm,0);
B = A rotated 120;
C = B rotated 120;
draw A--B--C--cycle;
```

Square (fig. 6)

Logo version:

```
powtórz 4 [ np 20 pw 90 ]
```

The `powtórz` (`repeat` in English) command makes a loop; the next number defines how many times this loop will be repeated.

MetaPost version:

```
draw (0,0)--(0,u)
   --(u,u)--(u,0)--cycle;
```

Polygon (fig. 7)

Logo version:

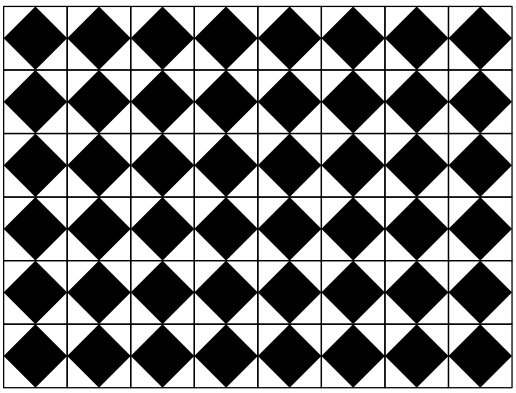
```
oto wielokąt :n :bok
powtórz :n [ np :bok pw 360/:n ]
już
```

The command `oto` defines a new procedure, here with parameters `:n` and `:bok`, then the procedure definition, ending with `już`.

MetaPost version:

```
def wielokat (expr n,b) =
 draw for i := 0 step 1 until n-1:
        1cm*up rotated (i*360/n) --
 endfor cycle;
enddef;
wielokat(5,1cm)
```

Double-squared chessboard (fig. 8)



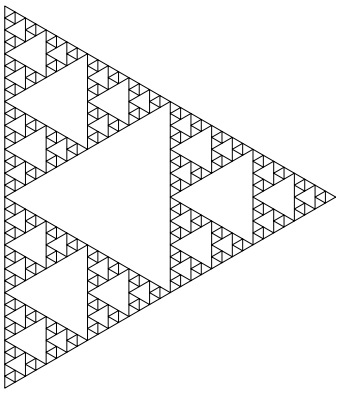
Logo version:

```
oto kw
 cs pż
 niech "a~25
 pod np 5 * :a lw 90 np 7 * :a pw 90 opu
 powtórz 6 [powtórz 8 [mkw :a pod pw 90
 np 2 * :a opu lw 90] pod
 np -2 * :a pw 90 np - 16 * :a lw 90]
 pod wróć
już

oto mkw :bok
 ugp 1 ukm 0
 pod np - :bok pw 90 np - :bok lw 90 opu
 powtórz 4 [np 2 * :bok pw 90]
 np :bok pw 45
 powtórz 4 [np :bok * pwk 2 pw 90]
 pw 45 pod np :bok lw 90
 zamaluj
już
```

MetaPost version:

```
numeric u; u := 25;
pair A,B,C,D;
A=(0,0); B=(u,0); C=(u,u); D=(0,u);

transform T;
A transformed T = 1/2[A,B];
B transformed T = 1/2[B,C];
C transformed T = 1/2[C,D];

path p; p := A--B--C--D--cycle;
path r; r := p transformed T;

picture o;
o := image (
        draw p;
        fill r withcolor black;
);
for i := 1 upto 8:
for j := 1 upto 6:
        draw o shifted (u*(i,j));
endfor;
endfor;
```

Sierpiński's Triangle (fig. 9)



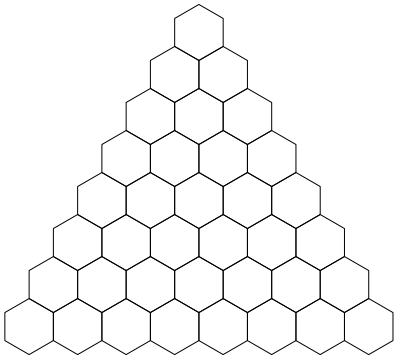
Logo version:

```
oto sierpinski :a :n
 sż opu
 jeśli :n = 0 [stop]
 trojkat :a
 sierpinski ( :a / 2 ) ( :n - 1 )
 np ( :a / 2 )
 sierpinski ( :a / 2 ) ( :n - 1 )
 np ( :a / 2 ) pw 120 np ( :a / 2 )
 sierpinski ( :a / 2 ) ( :n - 1 )
 np ( :a / 2 ) pw 120 np :a pw 120
już
```

MetaPost version:

```
def sierpinskiN (expr a, b, n) =
 if n = 0:
 draw a--(b rotatedabout(a,60))--b--cycle;
 else:
  sierpinskiN(a, 0.5[a,b], n-1);
  sierpinskiN(0.5[a,b], b, n-1);
  sierpinskiN(0.5[a,b rotatedabout(a,60)],
       0.5[a~rotatedabout(b,-60),b], n-1);
 fi;
enddef;
z0 = origin;
z1 = z0 shifted (0,-500);
sierpinskiN (z0, z1, 5);
```

Honeycomb (fig. 10)



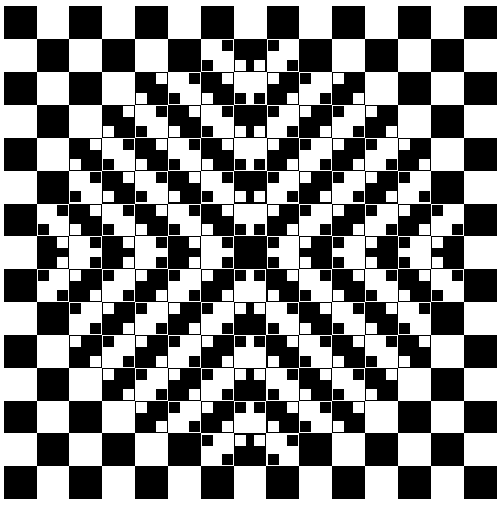
Logo version:

```
oto plaster :n
 jeśli :n < 1 [stop]
 cs sż
 niech "bok 300 / 8
 pod np ( - ( 0.75 * :n - 1 ) * :bok )
 pw 90 np ( - 0.5 * ( :n - 1 ) * :bok
 * pwk 3 ) lw 90 opu
 plastek :n :bok
już

oto plastek :n :bok
 jeśli :n = 0 [stop]
 powtórz :n [szesc :bok pod pw 90
 np :bok * ( pwk 3 ) lw 90 opu]
 pod np 1.5 * :bok pw 90
 np ( - ( :n - 0.5 ) * :bok * ( pwk 3 ) )
 lw 90 opu
 plastek :n - 1 :bok
już
```

MetaPost version:

```
numeric n,w; n := 6; w := 25;
path _s; _s :=  for i := 0 upto n-1:
  up rotated (i*360/n) -- endfor
  cycle;

def plast (expr n, b) =
 if n<>0:
  for i := 1 upto n:
   pair zz;
   xpart zz = ( (i-1)*(sqrt3) -
           ((sqrt3)/2) * n ) * b;
   ypart zz = -3/2*n * b;
   draw _s scaled b shifted zz;
  endfor;
 plast ((n-1), b);
 fi;
enddef;
```

Magic chessboard (fig. 11)



Logo version:

```
oto iluzja
 sż
 cs
 niech "bok 50
 niech "mbok 10
 skok 7 * :bok ( - 7 * :bok )
 powtórz 7 [powtórz 7           >>
  [kwadratcz skok 0 (2*:bok)]]
 kwadratcz skok - :bok (-:bok)
 powtórz 7 [kwadratcz skok 0 (-:bok*2)]
 skok - :bok :bok]
 powtórz 7 [kwadratcz skok 0 (2*:bok)]
 kwadratcz
 skok 7 * :bok ( - 7 * :bok )
 ; male
 skok :bok / 2 ( - :bok / 2 )
 niech "bok 10
 powtórz 4                      >>
  [kwadraciki pw 90 skok 25 ( - 25 )]
już
```

This code is wrong because of a restriction of the Logo Komeniusz editor. I used the symbol `>>` because the command `powtórz` and its syntax must be written in the same line. And this is a drawback of making procedures in Logo — regardless of the resulting code's legibility, some commands must be written all on one line.

The entire program takes 70 lines, and is on my web page: `www.mateusz.kmiecik.pl`

MetaPost version:

```
numeric u, k; u := 25; k := 0.3;
path fullsquare;
fullsquare :=
  unitsquare shifted -center unitsquare;

picture A, B, C, D, E, F, G, H;
path p, q;
p := fullsquare scaled u;
q := p scaled 0.33;

color Wh, Bl;
def Wh = white enddef;
def Bl = black enddef;

A := image (fill p withcolor black;);
B := image (fill p withcolor white;);

C := image (draw A;
fill q shifted (k*u*(-1,1)) withcolor Wh;
fill q shifted (k*u*(1,-1)) withcolor Wh;
);
D := image (draw B;
fill q shifted (k*u*(-1,1)) withcolor Bl;
fill q shifted (k*u*(1,-1)) withcolor Bl;
);
E := image (draw A;
```

```
fill q shifted (k*u*(1,1))  withcolor Wh;
fill q shifted (k*u*(1,-1)) withcolor Wh;
);
F := image (draw B;
fill q shifted (k*u*(1,1))  withcolor Bl;
fill q shifted (k*u*(1,-1)) withcolor Bl;
);

G := image (draw E
       rotatedaround (center E,180););
H := image (draw F
       rotatedaround (center F,180););

for i := 1 upto 7:
for j := 1 upto 8:
 draw
 if ( (j=1) and (i<>7) ):
   if odd(i): H else: G fi
 elseif ((i+j) = 8) and (i <>7):  D
 elseif ((i+j) = 6):              D
 elseif ((i+j) = 4):              D
 elseif ((i+j) = 9) and (i>2) and (i<6):
                                  C
 elseif ((i+j) = 7):              C
 elseif ((i+j) = 5):              C
 elseif ((i+j) = 3):              C
 else:
  if odd(j):
   if odd(i):    B
   else:         A
   fi
  else:
    if odd(i-1): B
    else:        A
    fi
  fi
 fi
 shifted(u*(i,j));
endfor;
endfor;

picture szach;
szach := currentpicture;
currentpicture:=nullpicture;

draw szach shifted (0,-u);
draw A;
draw currentpicture rotated 90;
draw currentpicture rotated 180;
```

5 Summary

Although Logo Komeniusz uses Polish commands, drawing some figures is still very troublesome. In MetaPost we make images by defining proper paths and points, which is very useful. Another difference is the approach to drawing figures: MetaPost doesn't control a pencil (or turtle) or anything like that, but defines proper paths and puts them on a virtual sheet of paper, filled with some color. The drawback of MetaPost is that we must have installed TEX and we must know how to operate it. Logo Komeniusz is a program which doesn't require any installation. After turning on it we can start working.

I think that for learning algorithms in secondary schools Logo Komeniusz is very valuable, while MetaPost can be introduced for additional lessons, perhaps in computing or mathematics.

References

[1] Jadwiga Orłowska-Puzio: *Dydaktyczne zastosowania programowania w LOGO*, http://www.jorlowska.prv.pl/rdydakt/logo.html, wersja PDF: http://www.jorlowska.prv.pl/rdydakt/docs/DYDAKTYCZNE_ZASTOSOWANIA_PROGRAMOWANIA_W_LOGO.pdf

[2] *Pierwszy dokument w MetaPost* [First document with MetaPost], http://www.gust.org.pl/doc/1stdocument/metapost

[3] John D. Hobby: *A user's manual for MetaPost*, http://www.tug.org/docs/metapost/mpman.pdf

[4] Hans Hagen: *Metafun*, http://www.pragma-ade.com/general/manuals/metafun-p.pdf

[5] *Métapost: exemples*, http://tex.loria.fr/prod-graph/zoonekynd/metapost/metapost.html

[6] André Heck: *Learning MetaPost by Doing*, http://staff.science.uva.nl/~heck/Courses/mptut.pdf

[7] Documentation, tutorials and examples at http://www.tug.org/metapost.html

◇ Mateusz Kmiecik
Gimnazjum nr 16 im. Króla Stefana Batorego, Kraków
mateusz (at) kmiecik dot pl
http://www.mateusz.kmiecik.pl

LaTeX News

Issue 19, September 2009

New LaTeX release

This issue of LaTeX News marks the first release of a new version of LaTeX 2ε since the publication of The LaTeX Companion in 2005–2006.

Just in time for TeX Live 2009, this version is a maintenance release and introduces no new features. A number of small changes have been made to correct minor bugs in the kernel, slightly extend the Unicode support, and improve various aspects of some of the `tools` packages.

New code repository

Since the last LaTeX release, the entire code base has been moved to a public SVN repository[1] and the entire build architecture re-written. In fact, it has only been possible for us to consider a new LaTeX release since earlier this year when the test suite was finally set up with the new system. In the process, a bug in the LaTeX picture fonts distributed with TeX Live was discovered, proving that the tests are working and are still very valuable.

Now that we can easily generate new packaged versions of the LaTeX 2ε distribution, we expect to be able to roll out bug fixes in a much more timely manner than over the last few years. New versions should be distributed yearly with TeX Live. Having said this, the maintenance of the LaTeX 2ε kernel is slowing down as the bugs become fewer and more subtle. Remember that we cannot change any of the underlying architecture of the kernel or any design decisions of the standard classes because we must preserve backwards compatibility with legacy documents at all costs.

Even new features cannot be added, because any new documents using them will not compile in systems (such as journal production engines) that are generally not updated once they've been proven to work as necessary.

None of this is to say that we consider LaTeX 2ε to be any less relevant for document production than in years past: a stable system is a useful one. Moreover, the package system continues to provide a flourishing and stable means for the development of a wide range of extensions.

Babel

One area of the LaTeX 2ε code base that is still receiving feedback to be incorporated into the main distribution is the Babel system for multilingual typesetting. While the Babel sources have already been added to the SVN repository the integration of the test system for Babel is still outstanding.

The future

While work on LaTeX 2ε tends to maintenance over active development, the LaTeX3 project is seeing new life. Our goals here are to provide a transition from the LaTeX 2ε document processing model to one with a more flexible foundation. Work is continuing in the expl3 programming language and the xpackages for document design. Future announcements about LaTeX3 will be available via the LaTeX Project website and in TUGboat.

[1] `http://www.latex-project.org/svnroot/latex2e-public/`

Talbot packages: An overview

Nicola L. C. Talbot

Abstract

This article briefly describes my packages that are available on CTAN and how they came about.

1 Introduction

Many of my packages end up on CTAN, the Comprehensive TEX Archive Network (`http://www.ctan.org`), and subsequently in TEX distributions such as TEX Live and MiKTEX. (My other packages are not on CTAN because they are either bespoke or still experimental.) Most of the packages are distributed under the "LATEX Public Project License" (`http://www.latex-project.org/lppl`).

2 General packages

This section describes packages for general use.

2.1 datetime

Changes the format of `\today` and provides commands for displaying the time.

The datetime package was the first package I wrote. It started out as an example in a LATEX course I was teaching in the late 1990s, but I later decided it might be useful to others so I uploaded it to CTAN.

The package provides the command

`\formatdate{`⟨*day*⟩`}{`⟨*month*⟩`}{`⟨*year*⟩`}`

that formats the given date according to the current date style. The command `\today` is redefined as `\formatdate\day\month\year` so they both display the dates in the same style. The default date style is the UK style or the style of the current language if babel has been loaded. The style can be changed via package options or declarations. You can also define your own custom date format.

An analogous command

`\formattime{`⟨*hour*⟩`}{`⟨*min*⟩`}{`⟨*sec*⟩`}`

formats the given time according to the current time style. The command `\currenttime` will display the current time according to the same format as `\formattime`. The style can be changed via package options or using `\settimeformat`. You can also define your own custom time format.

2.2 fmtcount

Display the value of a LATEX counter in a variety of formats.

When I first wrote the datetime package, I provided commands that convert a number or ordinal to a string so that it could be used with the textual date and time styles. I realised that people might want to use these commands but not want the rest of the datetime package so I decided to split them off into a separate package. While I was at it, I thought I may as well add some other formats, such as binary and hexadecimal.

2.3 glossaries

Create glossaries and lists of acronyms.

(This replaces the now obsolete glossary package.) With the glossaries package, you can define terms and acronyms which can then be used throughout the document. The default plural (appending an "s" to the singular) can be overridden and you can additionally specify a symbol or use the "user" keys to add other information.

To make a sorted list of the terms or acronyms to be displayed in the document, you need to use either `makeindex` or `xindy` to collate and sort the entries. Only those entries that have actually been used in the text will be displayed in the glossary or list of acronyms.

The glossaries bundle also provides the following packages:

- glossaries-accsupp: This package provides an interface between glossaries and Heiko Oberdiek's PDF accessibility support package accsupp.
- mfirstuc: This package provides `\makefirstuc`, which capitalizes the first letter of its argument, unless the first thing in its argument is a command with a non-empty argument, in which case it capitalizes the first letter of that command's argument. For instance, `\makefirstuc{abc}` produces 'Abc', `\makefirstuc{\ae{}bc}` produces 'Æbc', and `\makefirstuc{\emph{abc}}` produces '*Abc*'.

2.4 datatool

Tools to load and manipulate data.

(This replaces the obsolete csvtools package.) With the datatool package you can create databases, either via supplied commands or by loading an external CSV file. The original version of datatool was slow and inefficient, but Morten Høgholm suggested a much better internal representation of the database, which has significantly improved compilation time. The datatool bundle provides the following packages:

- datatool: This is the main package of the bundle. It can be used to create databases, iterate through a database, determine whether elements

Editor's note: The *TUGboat* editors requested this article in recognition of the many fine packages Nicola has created.

are integers, real numbers, strings or currency, and it provides commands that interface with the fp package to perform numerical operations on database elements (for example, computing the total, the arithmetic mean or standard deviation).

- datapie: This uses the pgf package to represent the contents of a database as a pie chart.
- dataplot: This uses the pgf package to plot the contents of a database.
- databar: This uses the pgf package to represent the contents of a database as a bar chart.
- databib: This works with BibTeX to convert a bibliography into a database. It can then be manipulated (for example, sorted according to a particular field) and displayed. Since you can apply filtering when iterating through a database, you can use it to, say, only display entries since a particular year, or only display entries that are journal articles.
- person: This provides support for displaying a person's name and pronoun in a document, thus avoiding the cumbersome use of "he/she" etc when performing tasks such as mail-merging.

2.5 flowfram

Create text frames for posters, brochures or magazines.

This package came about because I became frustrated trying to design technical posters for conferences. The layouts tended to require four columns with a figure or table spanning a couple of the columns, either at the top or at the bottom. I decided to adapt the technique used to format two-column text so that I could have an arbitrary number of columns with custom dimensions and locations so that the document text would flow from one column to the next just as it does in a regular two-column document. In addition I designed "static" and "dynamic" frames whose contents had to be set via a command or environment. This meant that I could position the figures and tables wherever I liked.

The same technique can be used to design the layout of brochures and magazines so I developed the code to make it easier to create a mini-table of contents that could be put in a separate frame alongside the chapter heading and thumbtabs which could be used to navigate through the printed document. TeX's `\parshape` and Donald Arseneau's shapepar package can be used to shape the static and dynamic frames so that the text can, say, go around an image or go in an L-shape to fit snugly around a shorter passage.

Determining the layout, particularly the parameters for `\parshape` and `\shapepar`, can be tricky so I made a Java GUI to assist. This is discussed in Section 4.

2.6 probsoln

Generate problem sheets and their solutions.

I wrote this package to help my husband, Gavin Cawley, generate assignment sheets. It underwent substantial modifications when Alain Schremmer asked for assistance with his documents at `http://freemathtexts.org`.

The idea is to have a database of questions (optionally with their solutions) and you can select particular questions, all questions or a random set. A package option or declaration can be used to hide or show questions or solutions so it's possible to either have each solution after the relevant question or have the solutions bundled together in another part of the document.

2.7 bibleref

Format bible citations.

I wrote this package in response to a query on `comp.text.tex`. It's designed to provide consistent formatting of references to parts of the Christian bible.

2.8 doipubmed

Special commands for use in bibliographies.

This package provides the commands `\doi`, `\pubmed` and `\citeurl` for use in bibliographies. Maarten Sneep and Heiko Oberdiek's doi package provides a more robust `\doi`.

2.9 quotmark

Consistent quote marks.

This package provides a means of ensuring consistent quote marks throughout your document. When I wrote it, I was unaware of Philipp Lehman's csquotes package.[1] The csquotes package provides more functionality than my quotmark package.

3 Development packages

The packages described here are provided for package and class developers.

3.1 makedtx

Perl script to help generate DTX and INS files.

Classes and packages are typically distributed as a DTX file with an accompanying INS file that's used

[1] I must have used the wrong terms in my keyword search.

to extract the code. However, when developing the code, it's a nuisance to have to edit the DTX file and then extract the STY or CLS files to test the modifications. I much prefer to edit the STY and CLS files directly, but this means that when I'm ready to distribute the new version I have to then bundle the code into a DTX file. I wrote this Perl script to do this for me.

3.2 xfor

Reimplements the LaTeX for-loop macro.

The glossaries and datatool packages need to use `\@for` to iterate through lists. However, quite often, I only need to search for an element that satisfies a particular condition, and it would be useful to terminate the loop when the element is found, akin to the `break` statement in languages such as C and Java. The xfor package redefines `\@for` so the loop can be terminated after the end of the current iteration.[2]

4 Jpgfdraw

Vector graphics application for LaTeX users (distributed under the GNU General Public License, `http://www.gnu.org/copyleft`).

My favourite drawing application has always been `!Draw`, which came with the Acorn Archimedes and Acorn RiscPC. Even after I moved to GNU/Linux, I still used the RiscPC to create images, which I then converted to Encapsulated PostScript. In the end, this became impracticable. Since I wanted to learn Java, I decided to write a Java application based on `!Draw`. While I was creating it, I was also considering writing an application that would work as a GUI for my flowfram package. Since both tasks required much of the same code, it seemed sensible to combine them into the same application. The code for the flowfram package required methods for computing the parameters for `\parshape` and `\shapepar` so I decided to also provide those as part of the set of LaTeX tools that come with the application. Briefly, you can use Jpgfdraw[3] to:

- Construct shapes using lines, moves and cubic Bezier segments;
- Edit shapes by changing the control points;
- Extract the parameters for TeX's `\parshape` command and for `\shapepar` (defined in the shapepar package);
- Construct frames for use with flowfram;
- Pictures can be saved in Jpgfdraw's native binary format (JDR) or native ASCII format (AJR) or can be exported in LaTeX and image formats:
  - a pgfpicture environment for use in LaTeX documents with the pgf package;
  - a single-page LaTeX document (including the picture code);
  - a LaTeX package based on flowfram;
  - a PNG image file;
  - an EPS file; or
  - an SVG image file;
- Incorporate text, text-along-a-path and bitmap images (for annotation and background effects);
- Alternative text may be specified for use when exporting to a LaTeX file (e.g. if the text contains symbols or if it should be set in maths mode);
- Mappings may be used to specify what LaTeX font declarations should be used when exporting to a LaTeX file (e.g. so you can map Chancery to `\fontfamily{pzc}\selectfont`).

Jpgfdraw is still in the beta stage. Occasionally it doesn't redraw some parts of the screen that need updating and it doesn't always update the bounding box for text areas. I hope to be able to fix these problems eventually. It also seems to have a problem when run on Windows Vista with Java6. This is difficult for me to test as I don't use Vista.

5 Conclusion

There have been times over the past couple of years when I considered giving up maintaining my packages as a result of ill-health and other commitments, and there were a couple of occasions when I was on the point of giving up, but then I received emails thanking me for the work I'd done and I changed my mind. I'm glad I didn't give up, although my responses to queries are somewhat slower than they used to be.

In my work as a production editor over the past year, I have developed a class that uses combine and hyperref to produce books, containing collections of articles, that can either be printed or made available on-line. It's my hope that at some point I'll be able to upload this to CTAN as well. The more I write classes and packages, the more I learn, and that's always satisfying.

◇ Nicola L. C. Talbot
University of East Anglia
Norwich, Norfolk
U.K.
`N.Talbot (at) uea dot ac dot uk`
`http://theoval.cmp.uea.ac.uk/~nlct/`

[2] In this respect, it differs from `break`, which immediately terminates the loop.

[3] I'm not very good at thinking of good names: it's a Java drawing package that creates pgf code.

Tuning LaTeX to one's own needs

Jacek Kmiecik

Abstract

I will not conceal that the topic brought up in the article has been discussed many times over, especially on mailing lists — but it continues to come back. There are numerous methods of adjusting LaTeX to particular typographic needs.

I am not going to advocate for a particular, the only proper, way of tackling low-level modifications of LaTeX macros, but will be suggesting several ways of handling such situations. As usual, there are many ways. The choice depends on how well one masters the tools. Also, I will restrict myself to selected areas of typesetting, those where adjustments are most often needed.

1 Introduction

LaTeX is for many the first form of contact with a TeX environment. In many cases, it remains so and LaTeX becomes the only set of macros (classes, styles, packages) which a user is able to work with and which he trusts. His own macro-creation is limited to tiny modifications of basic definitions as described either in textbooks, or in easily accessible documentation.

A more demanding necessity might make a user search for an appropriate macro package in the cavernous CTAN archives. According to various "omniscient" authorities and mailing lists, these packages are supposed to provide — magically — a wonderful automatic mechanism of improving everything simultaneously. If it is not a single package of macros, then, unfortunately, other packages — supplementing successive missing elements of the typesetting — are being added to the LaTeX preamble.

However, sooner or later it turns out that adding one more package breaks the compilation, changes the hitherto acceptable typesetting in impermissible places, or warns with announcements about some incompatibility.

Thanks to such situations, we can often encounter statements similar to the following: "I don't use LaTeX, because once something didn't work as I wanted (...) and that's why I use Plain TeX".

That's right! This should be the next step in one's TeX education. Nevertheless, writing all macros for a voluminous publication may become a spectacular challenge — some functionalities can be programmed quickly and easily (e.g., headings, imprints, footnotes), others demand advanced knowledge of TeX (the contents, preparation of a hyperlinked PDF file, multiple runs, etc.). Undisputed educational and cognitive values do not often go hand in hand with the hurry with which we have to cope with difficult cases, hence there is an irresistible temptation to use LaTeX as it seems to be an easier and friendlier environment. At least, it was the assumption of its creator, Leslie Lamport.

And if we ... let's say ... "marry" Plain TeX with LaTeX? In principle, all LaTeX macros are more or less extended plain macro definitions. Instead of being exposed to limitations of ready-made packages, it suffices to redefine the most necessary LaTeX commands for our own local use. Any way we look at it, the majority of packages originated in this way — locally needed definitions, modified canonic LaTeX macros — all have been compiled into a separate package and adjusted to be used with the contemporary LaTeX 2$_\varepsilon$ format. The issue of maintaining compatibility is a different topic, which we are not going to elaborate on. Nonetheless, we should be aware of the existing dangers.

The purpose of this article is to suggest ways of modifying of some basic, canonical LaTeX definitions, starting from their source code.

2 What should we start from?

From the documentation! The LaTeX 2$_\varepsilon$ standard distribution contains quite well-documented source code — in your basic installation you can find for sure the source directory, and in it a file entitled `source2e.tex`. Its compilation gives $\approx$600 pages of documentation of the LaTeX 2$_\varepsilon$ format source code — makes for good bedtime reading! Very instructive! And how "plain" it is! Almost everything is written with the conventions of primary plain macros.

3 And what's next?

Apply the method of small steps. If your document requires the use of colors and the desired destination format of the publication is PDF — I do not recommend that anyone write a whole colourful PDF machinery from scratch. We have verified and tested packages `color.sty` or `xcolor.sty`, so we do not have to reinvent the wheel. Likewise for the graphics embedding macros or building tabular setups.

Let's suppose we have to deal with a voluminous publication. As a first step I would suggest dividing the document into logically smaller parts, e.g., chapters, and place them in separate files attached to the main document file. In this file, in the preamble, we can put the additional essential commands,

This is a translation of the article "Dostosowanie LaTeX-a do konkretnych potrzeb", which first appeared in *Biuletyn GUST* 25 (2008), pp. 44–52. Reprinted with permission. Translation by Jerzy Ludwichowski and the author.

or — what seems to me the most appropriate solution — we can create our own package or even a class. Building our own local macro package seems to be decidedly easier for a TEX beginner, so we will choose this particular solution. Our own file `mystyle.sty` should be incorporated into the preamble as follows:

```
\usepackage{mystyle}
```

Such a solution allows the use of any plain or LATEX primary commands, on the condition that we know what we are doing. It also allows the use of the `@` sign when naming macros. The `@` sign is used by the inner LATEX constructs (the commands `\makeatletter` and `\makeatother` will then need to appear in the preamble of the main document). This gives us quite some power, but be forewarned that without good working knowledge of the LATEX documentation and its inner workings, we may — by our own efforts — accomplish more damage than good. This is why I recommend only minimal modifications. As our TEX skills reach higher levels, we may try more challenging things ... but for the time being let's stick to the small steps method ...

4 Page layout

Let's begin with the page layout — column size, margins, indents, side notes, headings ... how can we see the invisible? Where are the document elements such as the main column, side notes, captions or footer being placed? For DVI output:

```
\RequirePackage{showframe}
```

or when PDF is the output format:

```
\input{pdf-trans.tex}
\def\boxsh{\boxshow
    {0 .7 .1 RG .2 w}%
    {0 0 .8 RG [2 2]1 d}{}}
\let\shb\boxsh
```

The `showframe.sty` package is a part of a bigger macro package `eso-pic.sty` — it helps to illustrate the physical placement of the page contour, that is, the locations of fixed elements of the column exposition, such as heading, column, footer or side notes (fig. 1).

What are the `\RequirePackage` and all the rest being used for? Well, without delving into technical details of LATEX and the packages being used, more serious tasks can hardly be approached. At first it should suffice to say that we are defining low-level plain TEX macros to suit our own needs. The inquisitive user should consult the package documentation or be happy with a terse: *It is easier that way!* I would like to add that the `pdf-trans` package [3] by Paweł Jackowski, is written in low-level TEX so cleanly that it can be directly incorporated

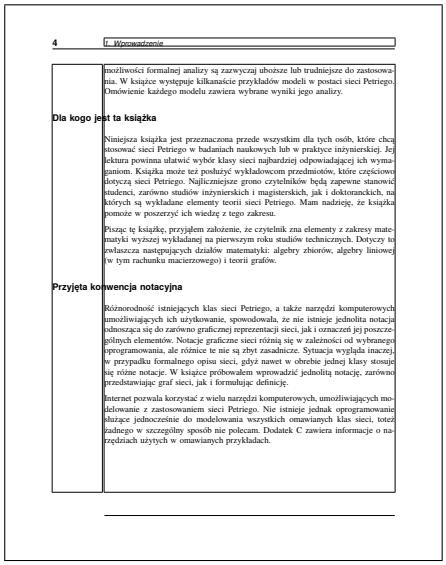


Figure 1: See with your own eyes what the composed page looks like

Figure 2: `\shb` depicts the boxes ...

into LATEX 2_ε. So far, I have not come across any conflict of those macros with any LATEX macros. The `\boxsh` definition, or its abbreviation `\shb` (*showbox*) placed directly before boxes like `\vbox` and `\hbox` will, when typeset, show their contours and baselines (fig. 2). We need only remember to exclude the "helpers" from the final typesetting, e.g.,

```
\let\shb\relax
```

Now we can move on to the next activity, i.e., setting of the page parameters as seen in the printout. The default page sizes defined in LATEX classes usually do not conform to the Polish typography tradition or the requirements of our publishing houses. Nothing stands in the way of adjusting them to our needs. Already in the standard format, there are about 15 parameters which define the characteristic page dimensions, and to which we have direct access. Not everything needs to be "touched" — if we do not use marginal notes in our document, then there is no need to manipulate their dimensions.

Let us for example do the following:

```
\setlength{\paperwidth} {165mm}
\setlength{\paperheight}{238mm}
\special{papersize=\the\paperwidth,
                   \the\paperheight}
```

It doesn't seem like much ... but if at a later stage the document will be further processed at prepress or converted to the PDF format, its pages should have proper outer dimensions (including margins, etc.). Without that magic "spell" the paper size information will be missing from the DVI file and most drivers (e.g., `dvips`) will in such a case assume the default letter page size (11 in × 8.1 in or 279 mm × 216 mm). Those who like minimizing chances for trouble in the future may readily remember and appreciate this little hint.

The modern production cycle of compiling documents directly to PDF from TEX sources, with pdfTEX, also requires these sizes to be set, this time like so:

```
\pdfpagewidth  165mm
\pdfpageheight 238mm
```

Other default size values will surely require corrections. For example:

```
\setlength{\textheight}    {190mm}
\setlength{\textwidth}     {130mm}
\setlength{\hoffset}       {  0mm}
\setlength{\voffset}       {-11mm}
\setlength{\oddsidemargin} { -5mm}
\setlength{\evensidemargin}{ -5mm}

\newlength\g@parindent
\setlength\g@parindent{20bp}

\setlength{\parskip}    {0pt}
\setlength{\parindent}  {\g@parindent}
\setlength{\leftmargini}{\g@parindent}
\setlength{\topskip}    {8bp}
```

The auxiliary variable `\g@parindent` we define above (something like a *global-parindent*) helps other macros to come, where we will define distances being multiples of the paragraph indent. Therefore, it seems sensible to define that particular length unit in one place. This "fixed" variable allows us to not to worry, for example, about some lengths changing with a change of the current font size.

What alternatives exist? One is the `geometry` package — calling it with appropriate parameters allows setting of all necessary dimensions of the typeset page [6]:

```
\RequirePackage[pdftex,
     papersize={185mm,235mm},
     textwidth= 123mm,
    textheight= 181mm,
         inner=  42mm,
    headheight=   9pt,
       headsep=   8mm,
          top=  14mm,
      footskip=  10mm,
marginparwidth=  21mm,
marginparsep=    1mm,
 includehead,
  asymmetric,
]%
{geometry}
```

(Note: the above examples are for two different publications).

We should also mention:

```
\sloppy
\widowpenalty   10000
\clubpenalty    10000
\flushbottom
```

The `\sloppy` parameter forces LATEX to set quite liberal demerit values thus allowing for sub-optimal line breaks and page breaks. The following two parameters, `\widowpenalty` and `\clubpenalty`, set for the maximum demerit value of 10000, forbid page breaks leaving "widows" or "orphans". The `\flushbottom` parameter (a possibly overzealous setting) asks for the following pages to be set to exactly the same height. Some packages may confuse the typesetting with respect to page height.

5 Not only Computer Modern

There is nothing to prevent replacing the "core" TEX fonts with fonts of the typographer's choosing. We will not go here into the problems related to using outline fonts with TEX as they have been discussed many times elsewhere. However, when typesetting math texts, especially with "tough" or "heavy" math, one should bear in mind that only the CM fonts offer the most comprehensive set of sophisticated math signs and symbols.

Recently the free fonts of high TEXnical quality were joined by the TEX Gyre family of fonts. They complement the collection of other freely available computer fonts of Polish origin (Antykwa Półtawskiego, Antykwa Toruńska, Kurier, Iwona, Cyklop) [10].

Lets assume that our model document will use Pagella (previously QuasiPalatino) as the base font (the current, serif font), the sans serif Heros font (in some installations it might lie around as QuasiSwiss) — for typesetting of headings or sometimes to emphasize portions of text, and Cursor (ex-QuasiCourier) as the typewriter font (e.g., for program code snippets). To achieve that one need only give the following commands:

```
\RequirePackage{pxfonts}
\RequirePackage{tgheros}
\RequirePackage{tgpagella}
\RequirePackage[QX]{fontenc}
```

The `pxfonts.sty` macro package enables the use of math symbols available with the Pagella font (the `pxfonts` package in TEX distributions). Inclusion

of `fontenc.sty` with the QX option allows for easy access to the complete set of glyphs available in the font — for more on the (Polish) QX encoding see [9].

All those packages can also be included "wholesale", with the following single command:

```
\RequirePackage{pxfonts,tgheros,tgpagella}
\RequirePackage[QX]{fontenc}
```

6 Typesetting of headings

The fundamental LaTeX commands `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph` are responsible for the formatting of the titles of the sectioning units of the document. Of course, the commands do much more but for now we are only going to discuss the formatting of their parameter text.

First, to make it easier to set the size of the font and line spacing, let's define some font manipulation utility macros:

```
\def\h@fam{qhv}
\def\font@def #1#2{%
    \usefont{\encodingdefault}%
    {\h@fam}{#1}{#2}}
\def\font@set #1#2#3#4{%
    \font@def{#1}{#2}%
    \fontsize{#3pt}{#4pt}\selectfont}

\def\foo@chp@num{\font@set{bx}{n} {32}{16}}
\def\foo@chp@txt{\font@set{b} {n} {18}{16}}
\def\foo@sec    {\font@set{b} {n} {14}{16}}
\def\foo@sse    {\font@set{m} {n} {12}{14}}
\def\foo@sss    {\font@set{b} {n} {11}{13}}
\def\foo@par    {\font@set{m} {n} {11}{13}}
\def\foo@spa    {\font@set{m} {it}{10}{12}}
```

Why these fancy macro names? Well, indeed, the naming is the writer's choice ... we tried to make them intuitive (font for the headings, chapter text, chapter number and so on) — there is no ideal, unambiguous recipe. Here we only suggest a solution — one can define macros with almost any name. What one should look out for, however, are names previously used inside the format, classes or packages used with our document.

Now we will move on to redefining the original LaTeX macros — the use of `\renewcommand` or `\def` is crucial. The `\def` command is to some degree dangerous: as a purely Plain TeX construct, it does not control name conflicts, thus one may unintentionally create quite some havoc. The `\newcommand` command detects possible name conflicts and so if we really want to redefine an already existing macro, we should use the `\renewcommand` command. Let's start with `\chapter`:

```
\renewcommand\chapter{%
  \if@openright\clearemptydoublepage
```


Przedmowa

Pomysł napisania tej książki zrodził się kilka lat temu, a konkretnie w 2000 roku, kiedy obroniłem pracę doktorską, której tematyka była związana z kolorowanymi sieciami Petriego. Rok wcześniej ukazała się książka [74], której jestem współautorem. Zawierała ona przegląd klas sieci Petriego niskiego poziomu. Brakowało jednak w literaturze książki, w której przedstawiono by wybrane klasy sieci Petriego niskiego i wysokiego poziomu z zachowaniem jednolitych oznaczeń i uwypukleniem różnic między tymi klasami. Postanowiłem tę lukę wypełnić, choć realizacja tych ambitnych planów się odsuwała, głównie z powodu różnych obowiązków naukowych i dydaktycznych. W tym czasie prowadziłem badania dotyczące systemów czasu rzeczywistego. Interesowały mnie możliwości zastosowania sieci Petriego do modelowania i analizy systemów wbudowanych, a zwłaszcza własności czasowych takich systemów.

Udało mi się w końcu doprowadzić do skutku moje zamierzenia. Książka ta nie zawiera przeglądu wszystkich klas sieci Petriego, nie jest też wyczerpującym studium na temat klas w niej opisanych. Zależało mi bowiem na przedstawieniu zarysu teorii sieci Petriego przez ukazanie różnorodności klas tych sieci, metod ich formalnej analizy oraz możliwości ich zastosowania do modelowania systemów współbieżnych i systemów czasu rzeczywistego. Miała to być specyficzna *podróż* przez różne klasy sieci, począwszy od najprostszych, skończywszy na bardzo rozbudowanych z możliwościami hierarchizacji opisu sieci.

Wszystkie omawiane w książce klasy sieci Petriego zostały opisane z podaniem ich formalnej definicji, z przedstawieniem ich najważniejszych własności oraz metod ich analizy. Główny nacisk położyłem na zaprezentowanie przykładów zastosowania poszczególnych klas sieci, zwracając uwagę zwłaszcza na formalną analizę ich własności. W kilku wypadkach ten sam system rozważam przy okazji omawiania różnych klas, tak aby zwrócić uwagę na różne możliwości rozmaitych klas sieci Petriego.

1

Figure 3: The beginning of an unnumbered chapter

```
  \else\clearpage\fi
  \thispagestyle{open}%
  \global\@topnum\z@
  \@afterindentfalse
  \secdef\@chapter\@schapter}
```

— here we call the `\clearemptydoublepage` command with the objective to output, if necessary, an empty left (even numbered) page (with head and foot empty), just before the chapter's title page. This command does not exist in the canonical set of LaTeX macros, so we have to define it ourselves [2]:

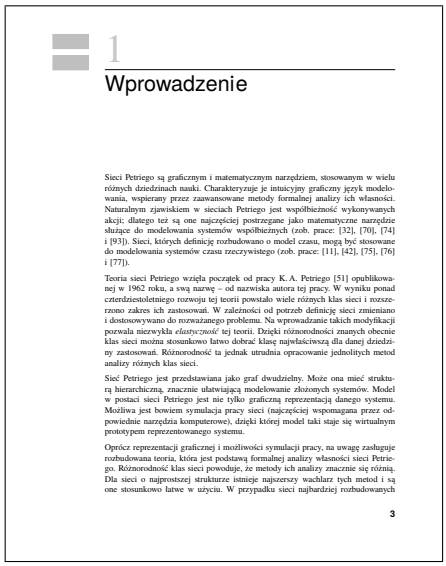
```
\newcommand{\clearemptydoublepage}{%
  \newpage{\thispagestyle{empty}}%
  \cleardoublepage}
```

The use of a nonstandard style for the current page `\thispagestyle{open}` may also have caught the reader's attention — we will discuss this in section 7.

The following changes reach deeper into the source code — they cannot be explained without a detailed analysis of the original LaTeX code — those interested are referred to the documentation. One might also see this as an antidote for sleeplessness. We will cut corners and go directly to the the macro code. The macros modify the headings in the way presented with figures 3 and 4.

Let us begin with some utility definitions (only those which have been used to change the look of the chapters' titles):

```
\definecolor {xxv@gray} {cmyk}{0,0,0,.25}
\newlength\begin@skip
\setlength\begin@skip {46mm}
\newsavebox{\chpt@box}
\def\@stempel{\vphantom
     {\foo@chp@num 0123456789}}
```


1
Wprowadzenie

Sieci Petriego są graficznym i matematycznym narzędziem, stosowanym w wielu różnych dziedzinach nauki. Charakteryzuje je intuicyjny graficzny język modelowania, wspierany przez zaawansowane metody formalnej analizy ich własności. Naturalnym zjawiskiem w sieciach Petriego jest współbieżność wykonywanych akcji; dlatego też są one najczęściej postrzegane jako matematyczne narzędzie służące do modelowania systemów współbieżnych (zob. prace: [32], [70], [74] i [93]). Sieci, których definicję rozbudowano o model czasu, mogą być stosowane do modelowania systemów czasu rzeczywistego (zob. prace: [11], [42], [75], [76] i [77]).

Teoria sieci Petriego wzięła początek od pracy K. A. Petriego [51] opublikowanej w 1962 roku, a swą nazwę – od nazwiska autora tej pracy. W wyniku ponad czterdziestoletniego rozwoju tej teorii powstało wiele różnych klas sieci i rozszerzono zakres ich zastosowań. W zależności od potrzeb definicję sieci zmieniano i dostosowywano do rozważanego problemu. Na wprowadzanie takich modyfikacji pozwala niezwykła *elastyczność* tej teorii. Dzięki różnorodności znanych obecnie klas sieci można stosunkowo łatwo dobrać klasę najwłaściwszą dla danej dziedziny zastosowań. Różnorodność ta jednak utrudnia opracowanie jednolitych metod analizy różnych klas sieci.

Sieć Petriego jest przedstawiana jako graf dwudzielny. Może ona mieć strukturą hierarchiczną, znacznie ułatwiającą modelowanie złożonych systemów. Model w postaci sieci Petriego jest nie tylko graficzną reprezentacją danego systemu. Możliwa jest bowiem symulacja pracy sieci (najczęściej wspomagana przez odpowiednie narzędzia komputerowe), dzięki której model taki staje się wirtualnym prototypem reprezentowanego systemu.

Oprócz reprezentacji graficznej i możliwości symulacji pracy, na uwagę zasługuje rozbudowana teoria, która jest podstawą formalnej analizy własności sieci Petriego. Różnorodność klas sieci powoduje, że metody ich analizy znacznie się różnią. Dla sieci o najprostszej strukturze istnieje najszerszy wachlarz tych metod i są one stosunkowo łatwe w użyciu. W przypadku sieci najbardziej rozbudowanych

3

Figure 4: The beginning of a numbered chapter

```
\def\foo@chp@num{%
    \font@set{qtm}{m}{n}{48}{48}}
\def\foo@chp@txt{%
    \font@set{qhv}{m}{n}{26}{28}}
```

And now to "more serious" code snippets:

```
\newcommand\@make@chapterhead[2]{%
  \nointerlineskip
  \vspace*{-\topskip}%
  \shb\vbox to\begin@skip{%
  \vspace*{-.5mm}%
  \parindent \z@
  \language 255
  \raggedright
  \color{xxv@gray}%
   \ifx|#2|
    \sbox{\chpt@box}%
      {\normalfont\foo@chp@txt
       \textcolor{black}{#1}}%
   \else
    \sbox{\chpt@box}%
      {\normalfont\foo@chp@num#1\@stempel}%
   \fi
  \chap@dinks
  \rlap{\raisebox{10pt}[0pt][0pt]%
        {\usebox{\chpt@box}}}%
  \color{black}%
  \rule{\linewidth}{.5pt}%
  \hfill\endgraf
  \par\vspace{-8pt}%
  \interlinepenalty\@M
  \foo@chp@txt#2\par
  \vspace{\stretch{1}}
  }\nointerlineskip \vskip-6pt}
```

That parameterized macro deals with two cases — numbered and unnumbered chapters. Doing it that way seems to facilitate future changes. Usage of this definition is trivial:

```
% for numbered:
\renewcommand\@makechapterhead[1]{%
  \@make@chapterhead
    {\thechapter}%
    {#1}}%
% for unnumbered:
\renewcommand\@makeschapterhead[1]{%
  \@make@chapterhead
    {#1}%
    {}}
```

Any questions? `\@stempel`? What are the digits 0–9 for? They are not needed in the presented case. The font we use does not require it because all digits have the same height.But let's imagine that the designer employed in that place an unusual, e.g., handwriting font — each digit has its own height and depth ... and the headings must be aligned "in one line". This definition is the easiest way to obtain the maximal dimensions.

Now, let's remind ourselves of how to modify the appearance of the lower level headings — there is little of pure plain but nonetheless:

```
\renewcommand\section{\@startsection
  {section}%            name
  {1}%                  level
  {\z@}%                indent
  {-3.5ex \@plus -1ex \@minus -.2ex}% above skip
  {2.3ex \@plus.2ex}%   below skip
  {\normalfont\foo@sec}% font
  }
```

Similarly for the titles of the lower levels down to `\subparagraph`, if need be. What is left is to insert the period sign after the section number in the numbered titles. The most convenient way is to redefine the proper macro:

```
\def\@seccntformat#1{
  \csname the#1\endcsname.\quad}
```

Low-level, as in ... plain!

7 Changing headers and footers

The default page headers are usually not satisfying. The same applies to footers, where page numbers are usually placed. However, nothing stands in our way of changing that appearance! The manipulations of the content of these page elements can be done according to the following pattern:

```
\def\ps@myheadings{%
  \def\@evenhead{ ...left  head... }%
  \def\@oddhead { ...right head... }%
  \let\@oddfoot { ...left  foot... }%
  \let\@evenfoot{ ...right foot... }%
  \let\@mkboth\markboth
  }
```

Figure 5: Page headers

For example, the headers presented in figure 5 were defined as follows:

```
\newlength\full@wd
\setlength\full@wd{145mm}
\def\foo@hed@pag{%
    \font@set{qhv}{b}{n}{11}{10}}
\def\foo@hed@odd{%
    \font@set{qhv}{m}{n} {9}{10}}
\let\foo@hed@eve = \foo@hed@odd
```

The `\foo@hed@pag` command defines the font to be used when typesetting the page numbers in the header, the `\foo@hed@odd` and `\foo@hed@eve` commands may be used to define different fonts for the left and right running headings. In the sample case illustrated with figure 5, the same font and size were used for both page numbers.

The style definition of the sample page looks as follows:

```
\def\ps@MYheadings{%
  \def\@evenhead{\@x@line
    {\makebox[22mm][l]{\foo@hed@pag\thepage}%
     \foo@hed@eve\leftmark
     \hfill}}%
  \def\@oddhead {\@x@line
    {\hspace*{22mm}\foo@hed@odd\rightmark
     \hfill
     \makebox[22mm][r]{\foo@hed@pag\thepage}}}%
  \let\@oddfoot\@empty
  \let\@evenfoot\@empty
  \let\@mkboth\markboth
}
\def\ps@open{
  \let\@evenhead\@empty
  \let\@oddhead\@empty
  \def\@oddfoot {\makebox[\textwidth][r]{%
        \foo@hed@pag\thepage}}%
  \def\@evenfoot{\makebox[\textwidth][l]{%
        \foo@hed@pag\thepage}}%
}
```

Here we see the previously mentioned, private page style `\thispagestyle{open}`, applied to the initial pages of chapters (figure 3 and 4). In a similar way other needed page styles might be defined.

And now the remaining definitions:

```
\def\@x@line#1{%
    \makebox[\textwidth][r]%
     {\shb\hb@xt@\full@wd{#1}\ul}}
\newcommand\ul{\unskip
  \llap{\rule[-4bp]{\full@wd}{.5bp}}}
```

As we see, LaTeX and plain commands can be used alongside. One only needs to remember their functionality — parametrisation, expandability and redefinitions.

The periods are still missing after chapter and section numbers:

```
\renewcommand\chaptermark[1]{%
  \markboth {\thechapter.\space#1}%
            {\thechapter.\space#1}}

\renewcommand\sectionmark[1]{%
  \markright{\thesection.\space{#1}}}
```

The default page style declaration is simple:

```
\pagestyle{MYheadings}
```

8 Redefining enumeration environments

Our customization zeal will not stop short of standard enumerations: we feel that some things should be done about them.

LaTeX enumeration environments have several variants and applications. However, a set of instructions similar for all of them determines their appearance (hanging indents, line spacing, indents). Unfortunately, we have to consult "the source", retype appropriate parts of the code and modify the relevant parameters. Let's do it:

```
\setlength\leftmargini  {\g@parindent}
\leftmargin\leftmargini
\setlength\leftmarginii {\g@parindent}
\setlength\leftmarginiii{\g@parindent}
\setlength\leftmarginiv {\g@parindent}
\setlength\leftmarginv  {1em}
\setlength\leftmarginvi {1em}
\setlength\labelsep     {.5em}
\setlength\labelwidth   {\leftmargini}
\addtolength\labelwidth {-\labelsep}
\setlength\partopsep{0\p@}
```

`\g@parindent`, defined close to the beginning of our code, has now found its application — now, independent of their placement within the document, enumerations will always be indented by the same amount. Unless we spoil this with a strange macro or environment.

In our next move we will change the formatting parameters for the consecutive enumeration nesting levels of which standard LaTeX allows up to six! Such deep nesting is not used in practice. One should think of re-writing the text rather than using very deeply nested enumerations, which might hinder comprehension. Anyway, let's now deal with the definitions:

```
\def\@listi{\leftmargin\leftmargini
  \parsep  \z@
  \topsep  .5\baselineskip
     plus .25\baselineskip
    minus .15\baselineskip
  \itemsep \z@ plus .5pt}
\let\@listI\@listi
\@listi
```

And the deeper levels:

```
\def\@listii {\leftmargin\leftmarginii
    \labelwidth\leftmarginii
    \advance\labelwidth-\labelsep
    \topsep  \z@
    \parsep  \z@
    \itemsep \z@}
\def\@listiii{\leftmargin\leftmarginiii
    \labelwidth\leftmarginiii
    \advance\labelwidth-\labelsep
    \itemsep   \z@}
```

. . . enough! Those who grasped the rules of the game will easily tackle even deeper levels if need be. and, as demonstrated, that kind of activity requires reading the source code of classes and styles.

Let's change the bullets for the consecutive nesting levels (we will skip the last two levels):

```
\renewcommand\labelitemi  {\sq@black}
\renewcommand\labelitemii {\sq@white}
\renewcommand\labelitemiii{\textemdash}
\renewcommand\labelitemiv {$\ast$}

\def\sq@black{\rule[.1ex]{1ex}{1ex}}
\def\sq@white{\mbox{%
    \fboxsep=0pt
    \fboxrule=.5pt
    \raisebox{.15ex}{\fbox{%
      \phantom{\rule{.8ex}{.8ex}}}}}}
```

The following command might turn out to be very useful:

```
\def\keepitem{\@beginparpenalty\@M}
```

— we forbid page breaks before the first short item. It suffices to give it just before that item — this is the equivalent of eliminating "orphans" in running text.

For those liking pure LaTeX solutions, the package `enumitem` is worth recommending. It allows for individual tailoring of each enumeration through proper parameters, or might be applied globally, in the preamble:

```
\setenumerate{%
    labelsep = 6pt,
  leftmargin = \leftmargini,
     itemsep = 1pt,
      topsep = 6pt,
   partopsep = 0pt,
      parsep = 0pt
 }
\setitemize{%
       label = \textsquare\hfill,
    labelsep = 6pt,
  leftmargin = *,
     itemsep = 1pt,
      topsep = 6pt,
   partopsep = 0pt,
      parsep = 0pt
  }
```

The meanings of the parameters are intuitive. Their names correspond to the various lengths used to construct enumerations. A more complete description of the package might be found in its documentation [1].

9 Modifying footnotes

Another typesetting element we might want to modify is footnotes. Let's assume we'd like the rule separating the footnote from the text column to be of full column width:

```
\renewcommand\footnoterule{%
  \kern-3\p@
  \hrule width\linewidth height.5pt
  \kern2.5\p@}
```

That's the way to gain direct access to the separating rule: we can modify its length, thickness, position and even colour!

With the replacement of the default font, need may arise to correct the positioning of footnote numbers both in the running text and the footnotes. Changes might be needed in the font and size of the indexes, or the footnote marks.

```
\def\@makefnmark{\hbox{%
 \@textsuperscript{\normalfont\@thefnmark}}}

\renewcommand\@makefntext[1]{%
 \parindent\g@parindent%
 \noindent
 \hb@xt@\parindent{\hss\@makefnmark\space}#1}
```

10 Positioning floating objects

Figures and tables are amongst the most often used floating objects. One may also encounter other objects with distinct typographic properties (algorithms, screen shots) and separate numbering. The placement of such typesetting elements is controlled in LaTeX through several parameters which, unfortunately, have their defaults set to rather restrictive

values. For publications with a large number of tables and figures, it might be difficult to obtain aesthetically pleasing page breaks and proper placements of the elements with respect to the running text.

However, again nothing stands in our way to slightly relax those parameters. For example:

```
\setcounter  {topnumber}            {3}  %%{2}
\renewcommand{\topfraction}         {.9} %%{.7}
\setcounter  {bottomnumber}         {0}  %%{1}
\renewcommand{\bottomfraction}      {.2} %%{.3}
\setcounter  {totalnumber}          {3}  %%{3}
\renewcommand{\textfraction}        {.1} %%{.2}
\renewcommand{\floatpagefraction}   {.85}%%{.5}
\setcounter  {dbltopnumber}         {2}  %%{2}
\renewcommand{\dbltopfraction}      {.8} %%{.7}
\renewcommand{\dblfloatpagefraction}{.8} %%{.5}
```

The meaning of the particular counters and parameters can be found in the LaTeX documentation. The default values offered by the format and standard LaTeX classes are given after the double percent sign. The modified values will surely cause a better placement of our floating objects.

11 Smaller fonts for tables et al.

Instead of placing a command changing the current font size in each and every `table` environment, we may achieve the desired result globally:

```
\def\font@caption{\font@set{qtm}{m}{n}{8}{10}}
\def\@floatboxreset {%
    \reset@font
    \font@caption
    \centering
    \@setminipage
 }
```

Isn't it simple? In a similar way the e.g., `verbatim` environment might be modified — a slight decrease of the typewriter font size without reducing the readability of these fragments. This may be achieved with a small change to the canonical LaTeX format:

```
\def\verbatim@font{%
    \normalfont\ttfamily\small}
```

However, one should bear in mind that some specialized packages dealing with verbatim-like environments will be immune to such machinations. Such packages might have internal font mechanisms and offer considerable flexibility.

12 Summary

The topic has by no means been exhausted — but still, I hope that the examples given here will convince those less versed in LaTeX that even that format may be modified to achieve one's own ends.

Turning such code snippets into styles or classes is a separate subject and, often, requires more in-depth studies of the subject, e.g., compatibility with other macro packages. There will be no way around reading the documentation or, in many cases, analysing the TeX code.

Some examples in this article were taken from this book by Marcin Szpyrka: *Sieci Petriego w modelowaniu i analizie systemów współbieżnych*, Warsaw 2007. The book was typeset for Wydawnictwa Naukowo-Techniczne using the techniques presented here.

References

[1] Javier Bezos: *Customizing lists with the enumitem package*, `mirror.ctan.org/macros/latex/contrib/enumitem/`

[2] Michel Goossens et al.: *The LaTeX Companion*, Addison-Wesley, 2nd edition (2004)

[3] Paweł Jackowski, *Box Transformations in pdf-trans.tex*, `mirror.ctan.org/macros/generic/pdf-trans/`

[4] Helmut Kopka, Patrick W. Daly: *Guide to LaTeX 2ε, Document Preparation for Beginners and Advanced Users*, Addison-Wesley, 4th edition (2003)

[5] Leslie Lamport: *LaTeX System opracowywania dokumentów Podręcznik i przewodnik użytkownika* [LaTeX: A Document Preparation System], Wydawnictwa Naukowo-Techniczne (2004)

[6] Hideo Umeki: *The geometry package*, `mirror.ctan.org/macros/latex/contrib/geometry/`

[7] Marcin Woliński: *Moje własne klasy dokumentów* [My own document classes], `www.math.upenn.edu/tex_docs/latex/mwcls/mwclsdoc.pdf`

[8] Marcin Woliński: *Ku polskim klasom dokumentów dla LaTeX-a*, Biuletyn GUST, nr 15, 2000

[9] *Kodowanie QX* [The QX encoding], `www.gust.org.pl/doc/fonts/qx/`

[10] *Projekty fontowe* [Font projects], `www.gust.org.pl/doc/fonts/projects`

◇ Jacek Kmiecik
AGH University of Science and Technology
University Computer Centre
al. Mickiewicza 30, 30-059 Kraków
`jk (at) agh dot edu dot pl`

Some misunderstood or unknown LaTeX 2ε tricks

Luca Merciadri

1 Introduction

Some simple things to do in LaTeX 2ε are often misunderstood (or simply unknown) to — even skilled — authors. We here present some useful coding for:

1. *pedagogical uses*: write a matrix with borders, show simplifications of terms in mathematical developments;
2. *presenting data*: make an accolade in a table;
3. *solving Springer's* `tocdepth` *problem in* `svmono`;
4. *writing European envelopes in an efficient way*;
5. *improving typography*: the `microtype` package.

These examples come from (Merciadri, 2009).

2 Pedagogical uses

It is always interesting to give students (whatever the institution) some clues about presented concepts. Sometimes, the LaTeX skills of lecturers are not sufficient. Despite their great ideas, these ideas sometimes fall into oblivion. Why? All their material is illustrated using LaTeX. If they decide to show something they cannot code in LaTeX, they will need to write it in another form (a non-LaTeX form). That makes it directly "less serious", or at least less competent than they want to appear. Facing this situation, they may choose to put aside their idea rather than show anything.

That is a bad thing, as LaTeX can do many things, even if it is sometimes tricky. A typesetting program should not limit the writer.

2.1 Matrix with borders

Let's say you want to explain an adjacency matrix to students. You first draw the triangle in Figure 1. (By the way, to make the drawing I used LaTeXDraw (`http://latexdraw.sourceforge.net`), which can generate graphics in PSTricks.)

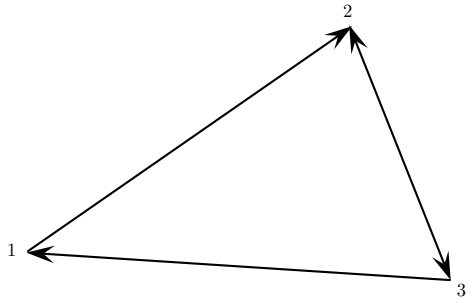


Figure 1: Graph example.

Next, you need to typeset its adjacency matrix. How? One of your colleagues has created this:

$$\mathcal{M}_S = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

He's away now, so you can't ask him how he did it. So here's how:

```
\mathcal M_{S} =
  \bordermatrix{
            & 1 & 2 & 3 \cr
          1 & 0 & 1 & 0 \cr
          2 & 0 & 0 & 1 \cr
          3 & 1 & 1 & 0 \cr
  }
```

Side note: `\bordermatrix` uses `\tabular` underneath, so, as with every `tabular`, you can use `\hline` to draw a horizontal line.

2.2 Show simplifications

It is sometimes helpful, particularly when writing long mathematical developments, to show the simplifications which can be done, especially when the developments become tricky. (The opposite is also true: it sometimes happens in proofs that you choose to write, for example,

$$(x+1)^2$$

as

$$(x+1+(1-1))(x+1+(1-1))$$

as a step on the way to the result you want to prove.)

For cancelling terms, you might use `\not`, but it is not very efficient, and was not implemented for this. A better alternative is to use the `cancel` package, thus:

```
\usepackage{cancel}
```

in your preamble, and then

```
\dfrac{\cancel{(b-a)} (a^2+ab+b^2)}
      {\cancel{(b-a)}(b+a)}
```

in the body. The result is:

$$\boxed{\frac{\cancel{(b-a)}(a^2+ab+b^2)}{\cancel{(b-a)}(b+a)}}.$$

Typographically, cancelling terms like this is ugly, but, pedagogically, it can be very useful for students.

3 Presenting data

Tables constitute a natural way to display data without taking too much space. They can be very useful with judicious choice of data.

3.1 With braces

Let's say that you want to make a table. Everything is set up: you have the data, the bibliography references, and you have written the whole table. But you are puzzled, because you want to put a connecting brace at some place to help the reader. You can use:

```
\usepackage{multirow}
\usepackage{bigdelim}
```

in the preamble, and

```
\begin{tabular}{l|l|r}
\hline
$a$ & \rule{0pt}{10pt} $b$ &\\
\hline
$c$ & $d$ & \rule{0pt}{12pt}
            \multirow{2}{*}{}
            \rdelim\}{2}{1cm}[text]\\
$e$ & $f$ &
\end{tabular}
```

in the body. The (invisible) `\rule` commands stop the b and the brace from touching the horizontal rule above. The result:

a	b	
c	d	} text
e	f	

4 Springer's `tocdepth` problem

Springer is one of the largest mathematical publishers. When writing a book with their `svmono` class, you may realize that it does not respect `tocdepth`. As a result, subparagraphs cannot be included in the table of contents — it can include chapters, sections, subsections, subsubsections, and paragraphs ... but not subparagraphs. The following code solves this:

```
\makeatletter
\renewcommand\subparagraph{%
  \@startsection{subparagraph}{4}{\z@}%
   {-18\p@}% \p@lus -4\p@ \@minus -4\p@}%
   {6\p@}% \p@lus 4\p@ \@minus 4\p@}%
   {\normalfont\normalsize\itshape
    \rightskip=\z@ \@plus 8em%
    \pretolerance=10000}%
}
\makeatother
```

5 Envelopes

You'd like to write envelopes with LaTeX but do not know how. Here is one global solution. What could be better looking than a LaTeX-generated envelope?

5.1 C6 standard and adaptations

Here is an example of an envelope document (using the C6 standard size).

```
\documentclass[12pt]{letter}
\usepackage{geometry}
\geometry{paperheight=162mm,paperwidth=114mm}
\usepackage{graphics}
\usepackage[c6envelope,noprintbarcodes,
            rightenvelopes,printreturnaddress]
           {envlab}
\makelabels
\begin{document}
\startlabels
\mlabel{%
% Sender's info:
 LastName FirstName\\
 Street No\\
 ZIPcode City (Country)}{
% Receiver's info:
 LastName FirstName\\
 Street No\\
 ZIPcode City (Country)}
\end{document}
```

You can directly print the output of such a code on a C6 envelope. If your envelopes have height M mm and width N mm, just replace the third line in this code with:

```
\geometry{paperheight=Mmm,paperwidth=Nmm}
```

5.2 In the printer

You need to put your C6 envelope in your printer as illustrated at Figure 2.

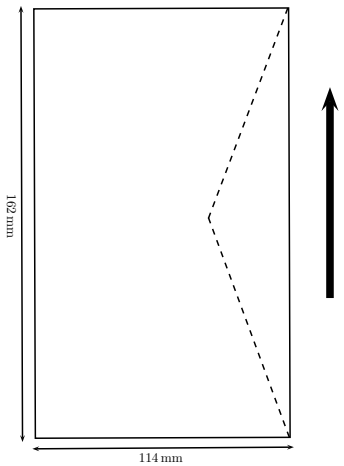


Figure 2: How to put your C6 envelope in the printer.

5.3 Sample output

An example of the envelope's output which will be produced is given in Figure 3.

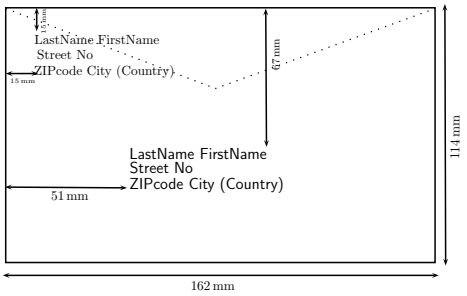


Figure 3: Generated C6 envelope.

5.4 Standard envelope sizes

Here are some standard sizes for envelopes:

1. C4 format: 22,9 cm × 32,4 cm, to transport an A4 sheet,
2. C5 format: 16,2 cm × 22,9 cm, to transport an A4 sheet folded one time,
3. C6 format: 11,4 cm × 16,2 cm, to transport an A4 sheet folded two times.

You can directly modify the envelope code in Section 5.1 to match the dimensions of your envelope.

6 Improving typography

According to Schlicht (2009), the `microtype` package provides a LaTeX interface to the micro-typographic extensions of pdfTeX: most prominently, character protrusion and font expansion. Furthermore, it provides for the adjustment of interword spacing and additional kerning, as well as hyphenatable letterspacing (tracking) and the possibility of disabling all or selected ligatures.

That is, `microtype` basically helps you produce better output when used in conjunction with pdfTeX. You do not have to worry over the details; just use:

```
\usepackage{microtype}
```

in the preamble of your document. If you are using `babel`, for further refinement you may write

```
\usepackage[babel=true]{microtype}
```

An example of its interest is illustrated by the following phenomenon. This document has been typeset using `ltugboat` class and, among others, the `microtype` package, with the options which were given before. If this package is not added to the preamble, a black box appears in the first sentence of Section 2: the period was too far into the margin. With `microtype` loaded, all is well.

Many options can be selected with `microtype`. Reading its nice manual (Schlicht, 2009) is a fruitful way to learn more about typesetting.

◇ Luca Merciadri
University of Liège
Luca.Merciadri (at) student dot ulg dot ac dot be
http://www.student.montefiore.ulg.ac.be/~merciadri/

References

Downes, Michael. *Short Math Guide for LaTeX*, 2002. ftp://ftp.ams.org/pub/tex/doc/amsmath/short-math-guide.pdf.

Merciadri, Luca. "A Practical Guide to LaTeX Tips and Tricks". 2009.

Schlicht, R. *The microtype package: An interface to the micro-typographic extensions of pdfTeX*, 2009. http://mirror.ctan.org/macros/latex/contrib/microtype/microtype.pdf.

LaTeX3 News

Issue 3, January 2010

Happy New Year

Welcome to the holiday season edition of 'news of our activities' for the LaTeX3 team.

Recent developments

The last six months has seen two significant releases in the LaTeX3 code. In the CTAN repository for the xpackages,[1] you'll find two items of interest:

- A revised version of xparse; and
- The new package xtemplate, a re-implementation of template with a new syntax.

Special thanks to Joseph Wright who handled the implementations above almost single-handedly (with lots of input and feedback from other members of the team and members of the LATEX-L mailing list).

These two packages are designed for the LaTeX package author who wishes to define document commands and designer interfaces in a high-level manner.

xparse This package allows complex document commands to be constructed with all sorts of optional arguments and flags. Think of how `\newcommand` allows you to create a command with a single optional argument and xparse is a generalisation of that idea.

xtemplate This package requires more explanation. Xtemplate is designed to separate the logical information in a document from its visual representation. 'Templates' are constructed to fulfil individual typesetting requirements for each set of arguments; to change the look of a certain part of a document, instantiations of templates can be swapped out for another without (a) having to change the markup of the source document, or (b) having to edit some internal LaTeX macro.

LaTeX 2$_\varepsilon$ packages, such as geometry or titlesec, already provide parameterized interfaces to specific document elements. For example, one may use titlesec to change the layout of a `\section`: one modifies its layout parameters via `\titleformat` and `\titlespacing`. In a way, such packages define a template for a specific document element and some manipulation commands to instantiate it. However, the moment the intended layout is not achievable with one package you are on your own: either you have to resort to low-level programming or find some other high-level package which, of course, comes with its own set of conventions and manipulation commands.

The xtemplate package can be thought of a generalization of such ideas. It provides a uniform interface for defining and managing templates for any kind of document element and most importantly provides a uniform interface for instantiating the layout.

Thus the designer activity of defining or modifying a document class is limited to selecting the document elements that should be provided by the class (e.g., `\chapter`, `\section` `\footnote`, lists, ...), selecting appropriate "named" templates for each of them, and instantiating these templates by specifying values for their layout parameters. If a desired layout can't be achieved with a given template a different template for the same document element can be selected.

Programming is only necessary if no suitable template for the intended layout is available. It is then that a LaTeX programmer has to build a new template that supports the layout requirements. Once this task is complete, the template may be added to the selection of templates that designers and users may choose from to define or adjust document layouts seamlessly.

This is a slight gloss over the complexities of the package itself, which you can read about in the documentation. We've tried to document xtemplate clearly but we'd love feedback on whether the ideas make sense to you.

As an addendum to the introduction of xtemplate, the older template package will be retired in the near future. To our knowledge there is only a single package on CTAN that uses template, namely xfrac, and members of the LaTeX team are in the process of switching this package over to xtemplate. If you have any private code that uses template, please let us know!

Upcoming plans

Having announced the updated xparse and the new xtemplate, the next stage of development will revolve around using these two systems in the other components of the xpackages, feeding back our experience in practise with the original ideas behind the designs and evaluating if the packages are meeting our expectations.

[1] http://mirror.ctan.org/tex-archive/macros/latex/contrib/xpackages/

Beyond \newcommand with xparse

Joseph Wright

1 Introduction

The LaTeX 2ε \newcommand macro is most LaTeX users' first choice for creating macros. As well as the 'sanity checks' it carries out, the ability to define macros with an optional argument is very useful. However, to go beyond using a single optional argument, or to create more complex input syntaxes, LaTeX 2ε users have to do things 'by hand' using \def or load one of the packages which extend \newcommand (for example twoopt (Oberdiek, 2008)).

As part of the wider efforts to develop LaTeX3, the xparse package (LaTeX3 Project, 2009) aims to replace \newcommand with a much more flexible set of tools. This means looking again at the way that commands are defined, and so xparse uses different syntax conventions to \newcommand. In this article, I will be looking at some of the highlights of xparse.

2 Creating document commands

LaTeX 2ε provides not only \newcommand, but also \renewcommand and \providecommand, all sharing a common syntax. xparse also provides a family of related commands following the same pattern:

- \NewDocumentCommand For defining a macro not already defined, giving an error message if it is.
- \RenewDocumentCommand For changing a definition, issuing an error message if the macro does not already exist.
- \ProvideDocumentCommand Creates a macro if it does not exist, and otherwise does nothing: i.e., will not change an existing definition.
- \DeclareDocumentCommand Does no checks for an existing definition: simply defines the macro using the expansion given.

As \DeclareDocumentCommand always creates an updated definition, it is most convenient for the examples in the rest of this article.

The \DeclareDocumentCommand function takes three mandatory arguments:

1. The name of the function to define;
2. An 'argument specification';
3. The code which the function expands to.

```
\DeclareDocumentCommand \foo { m } {%
  % Code here
}
```

The first and third arguments are essentially the same as the equivalents for \newcommand: it is the argument specification that marks out an xparse definition. As you might guess from the above example, it is enclosed in braces, and spaces are ignored.

3 Argument specifications

The basic idea of an argument specification ('arg spec') is that each argument is listed as a single letter. This means that the number of letters tells you how many arguments a function takes, while the letters themselves determine the type of argument. As the argument specification is a mandatory argument, a function with no arguments still needs an arg spec.

```
\DeclareDocumentCommand \foo { } {%
  % Code with no arguments
}
```

xparse provides a range of argument specifier letters, some of which are somewhat specialised. The following is therefore only covers the most generally useful variants in detail.

Mandatory arguments are created using the letter m. So

```
\DeclareDocumentCommand \foo { m m } {%
  % Code with 2 arguments
}
```

is nearly equivalent to

```
\newcommand*\foo[2]{%
  % Code with 2 arguments
}
```

The 'nearly' is an important point: in contrast to \newcommand, xparse functions are not \long by default. In xparse, we can decide for each argument whether to allow paragraph tokens or not. This is done by preceding the arg spec letter by +:

```
\DeclareDocumentCommand \foo { m +m } {%
  % #1 No \par tokens allowed
  % #2 \par tokens permitted
}
\DeclareDocumentCommand \foo { +m +m } {%
  % Both arguments allow \par
}
```

LaTeX optional arguments with no default value are given the letter o, while those with a default value are given the letter O. The latter also requires the default itself, of course!

```
\DeclareDocumentCommand \foo { o m } {%
  % First argument optional, no default
  % Second argument mandatory
}
\DeclareDocumentCommand \foo
  { O{bar} m } {%
  % First argument optional, default "bar"
  % Second argument mandatory
}
```

The use of two separate letters here illustrates another LaTeX3 concept: functions used for setting up a document should have a fixed number of mandatory arguments. So while `o` is given with no additional information, `O` must always be given along with the default (as shown).

Thus far, the xparse method does not go significantly beyond what is possible using `\newcommand`. However, as well as recognising more types of argument, xparse also allows free mixing of optional and mandatory arguments. For example, it is easy to create a function with two optional and two long mandatory arguments in one step.

```
\DeclareDocumentCommand \foo
  { o +m o +m } {%
  % Four args, #1, #2, #3 and #4
  % Only #2 and #4 can include \par tokens
}
```

Creating this type of behaviour is far from trivial without xparse.

Generalising the idea of a LaTeX 2_ε optional argument, which is always enclosed in square brackets, xparse can create optional arguments delimited by any pair of tokens. This is done using the letters `d` (no default value) and `D` (with default value): 'd' stands for 'delimited'. So we can easily add an argument in parentheses or angle brackets, for example.

```
\DeclareDocumentCommand \foo
  { d() D<>{text} m } {%
  % Optional #1 inside ( ... )
  % Optional #2 inside < ... >
  %   with default "text"
  % Mandatory #3
}
```

A standard LaTeX 2_ε method to indicate a special variant of a macro is to add a star to its name. xparse uses the letter `s` to indicate this type of argument. There is then a need to indicate if a star has been seen. This done by returned one of two special values (`\BooleanTrue` or `\BooleanFalse`), which can be checked using the function `\IfBooleanTF`:

```
\DeclareDocumentCommand \foo { s m } {%
  \IfBooleanTF #1 {%
    % Starred stuff using #2
  }{%
    % Non-starred stuff using #2
  }%
}
```

A generalised version of the `s` specifier, with the letter `t` for 'token'. This works in exactly the same way, but for an arbitrary token, which is given following the 't'.

```
\DeclareDocumentCommand \foo { t/ m } {%
  \IfBooleanTF #1 {%
    % Code if a slash was seen
  }{%
    % Code if no slash was seen
  }%
}
```

Of the more specialised specifier letters, perhaps the most interesting is `u`, to read 'up to' some specified value.

```
\DeclareDocumentCommand \foo
  { u{stop} } {%
  % Code here
}
\foo text stop here
```

Here, the code will parse '`text␣`' as `#1`. Following standard TeX behaviour, the space between 'text' and 'stop' will be picked up as part of the argument.

4 Optional arguments

`\newcommand` does not differentiate between an optional argument which has not been given and one which is empty:

```
\newcommand\foo[2][]{%
  % Code
}
\foo{bar}
\foo[]{bar}
```

In both cases, `#1` is empty: not entirely helpful. It is possible to get around this using a suitable default value, but xparse aims to solve this problem in a general fashion.

When no default is available for an optional argument, xparse will return the special marker `\NoValue` if the argument is not given. It is then possible to check for this marker using the `\IfNoValue` test:

```
\DeclareDocumentCommand \foo { o m } {%
  \IfNoValueTF{#1}{%
    % Stuff just with #2
  }{%
    % Stuff with #1 and #2
  }%
}
```

Following the standard LaTeX3 approach, this test is available with versions which only have a true or false branch:

```
\DeclareDocumentCommand \foo { o m } {%
  \IfNoValueF{#1}{%
    % Stuff with #1
  }%
  % Stuff with #2
}
```

5 Robustness

xparse creates functions which are naturally 'robust'. This means that they can be used in section names and so on without needing to be protected using `\protect`. This makes using functions created using xparse much more reliable than using those created using `\newcommand`, particularly when there are optional arguments.

xparse is also designed so that optional arguments can themselves contain optional material. For example, if you try

```
\newcommand*\foo[2][]{%
  % Code
}
\foo[\baz[arg1]{arg2}]{arg3}
```

you will find that `\foo` will pick up '`\baz[arg1`' as `#1` and '`arg2`' as `#2`: not what is intended. However, the same code with xparse

```
\DeclareDocumentCommand \foo { o m } {%
  % Code
}
\foo[\baz[arg1]{arg2}]{arg3}
```

will parse '`\baz[arg1]{}`' as `#1` and '`arg`' as `#2`, as anticipated.

6 Fully expandable commands

There are a small number of circumstances under which fully expandable detection of optional arguments is desirable. For example, the etextools package (Chervet, 2009) provides a number of utility macros to produce this type of macro.

Rather than require the learning of an entirely new method for creating purely expandable commands, xparse can generate them in an analogous manner to normal (robust) commands.

```
\DeclareExpandableDocumentCommand \foo
  { o m } {%
  % Expandable code
}
```

This process has some limitations, some of which can be detected by xparse at definition time. It is therefore intended for exceptional use when a robust command will not behave suitably.

7 Environments

In analogy to the relationship between `\newcommand` and `\newenvironment`, xparse provides the function `\DeclareDocumentEnvironment` (and variants) for creating environments. The same argument specifications are used for declaring the arguments to `\begin{...}`. The crucial difference to standard LaTeX 2_ε environments is that the arguments are also available in the `\end{...}` code.

```
\DeclareDocumentEnvironment {foo} { o m } {%
  % Begin code using #1 and #2
}{%
  % End code using #1 and #2
}
```

8 Conclusions

By providing a single interface for defining both simple and complex user functions, xparse frees us from needing to worry about the detail of parsing input. Almost all cases can be covered without the need to use low level methods to process input.

Final note: you can use xparse in LaTeX 2_ε. Just:

```
\usepackage{xparse}
```

References

Chervet, Florian. "The etextools package: An ε-TeX package providing useful (purely expandable) tools for LaTeX users and package writers". Available from CTAN, `macros/latex/contrib/etextools`, 2009.

LaTeX3 Project. "The xparse package: Generic document command processor". Available from CTAN, `macros/latex/contrib/xpackages/xparse`, 2009.

Oberdiek, Heiko. "The twoopt package". Part of the oberdiek bundle, available from CTAN, `macros/latex/contrib/oberdiek`, 2008.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph dot wright (at)
morningstar2 dot co dot uk

Programming key–value in expl3

Joseph Wright

1 Introduction

Key–value entry, in which a series of ⟨*key*⟩ = ⟨*value*⟩ statements are given in a comma-separated list, is a powerful method for specifying a flexible range of values in (LA)TEX. For the user, this type of input avoids the need for a very large number of control macros to alter how a LATEX package or class behaves. Using key–value input also allows the programmer to expose a very wide range of internal settings. Used properly, this should avoid the need for users to access or modify the internal code of a LATEX package or class to achieve the desired behaviour.

Programming key–value methods can be confusing, as the link between the user interface and implementation at the code level is not always obvious. Christian Feuersänger and I have given an overview of programming key–value input using most of the LATEX 2ε implementations (Wright and Feuersänger, 2010), covering both the broad concepts and the implementation details.

The new possibilities opened up by the expl3 programming language of LATEX3 (LATEX3 Project, 2010) include providing a new, consistent interface for creating key–value input. Programming using expl3 is somewhat different from traditional (LA)TEX, and I covered the key general concepts for using expl3 in an earlier *TUGboat* article (Wright, 2010). In this article, I will focus on how expl3 implements key–value methods, with an assumption of some familiarity with both expl3 and using key–value with LATEX 2ε.

2 Low-level key–value support: l3keyval

The expl3 bundle of modules includes two with `key` in their name, l3keyval and l3keys. The two are aimed at different parts of the programming process, and most of this article will focus on l3keys. However, some idea of the place of l3keyval in the larger scheme is useful.

The l3keyval module provides low-level parsing of key–value input, used by l3keys but also available for other purposes. Thus, l3keyval does not do anything beyond split a list first at each comma (that is, into key–value pairs), and then into a key and value. It includes the facility to sanitize the category codes of '`,`' and '`=`', and also to ignore spaces if necessary.

The result is that, while l3keyval is crucial for key–value support using expl3, the exact mechanisms used are unimportant. This frees us to focus on the facilities provided by l3keys. (Well, it frees *you* from understanding l3keyval: I wrote most of l3keys, so I have to know what is going on!)

3 The design ideas for l3keys

Perhaps the best package for defining key–value input using LATEX 2ε methods is pgfkeys (Tantau, 2008). It uses key–value input in the definition of keys themselves, and thus uses the power of key–value methods to help the programmers as well as the end user. The approach taken by the l3keys module in expl3 is inspired by pgfkeys, although programmers familiar with the latter will find some important differences.

The main design ideas for l3keys were:

- Use of key–value methods for creating keys at the programming level;
- Separate functions for defining and setting keys (namely, `\keys_define:nn` and `\keys_set:nn`);
- 'Fit' with the LATEX3 syntax and variable conventions;
- Rich set of key types, including clear handling of multiple choices.

Taking these ideas and concepts from pgfkeys, l3keys introduces the idea of 'properties' for keys. Each valid key name must have at least one property, to attach some code to the key. By combining a number of properties, a wide range of effects can be created without an overly-complex interface.

4 Functions for keys

Keys are created using the `\keys_define:nn` function, where the function name follows general expl3 conventions and thus requires two arguments. The first is the module name with which the keys are associated. Typically, this will be the same as the LATEX 2ε package or LATEX3 module name being created, although it can be more complex. The second argument for `\keys_define:nn` is a list of keys, properties and values, which are then used to set up the key–value system.

```
\keys_define:nn { module } {
  key-one .property-a:n = value-one   ,
  key-one .property-b:n = value-two   ,
  key-two .property-a:n = value-three ,
}
```

As illustrated, the 'properties' of a key are indicated starting with a full stop (period) character at the end of the name of the key. (Remember that expl3 code blocks ignore spaces, so there are no significant spaces in the example.) In line with expl3 conventions, each property includes a specification to indicate what arguments it expects.

The second part of using key–value methods is setting keys, and is handled by the `\keys_set:nn`

function. This also takes the module as the first argument and a key–value list as the second:

```
\keys_set:nn { module } {
  key-one = value-one   ,
  key-one = value-two   ,
  key-two = value-three ,
}
```

Here, the key–value list is used along with the key implementation to 'set' the keys.

`\keys_define:nn` will almost always appear inside code blocks, and so does not carry out any sanity checks on category codes of its input. On the other hand, `\keys_set:nn` is likely to handle user input, and so does carry out these checks. It is also worth saying that `\keys_set:nn` will typically be 'wrapped up' in a user-accessible function, say, `\modulesetup`. Using the LaTeX3 xparse module, this might look like:

```
\DeclareDocumentCommand
  \modulesetup { +m } {
    \keys_set:nn { module } {#1}
  }
```

or using traditional LaTeX 2_ε:

```
\newcommand \modulesetup [1] {
  \keys_set:nn { module } {#1}
}
```

5 Key properties

The most general property that can be given for a key is `.code:n`. This associates completely general code with a particular key name; the value given to a key when used is available within the code as `#1`.

```
\keys_define:nn { module } {
  key .code:n = You~gave~input~#1! ,
  ...
}
```

As is generally the case with key–value input, we do not need braces around the code here, as it is delimited by the comma separating key–value pairs. The only exception is if the code itself contains `,` or `=` characters, which of course need to be 'hidden'.

Related to the `.code:n` property is `.code:x`. Following expl3 conventions, the difference here is in the expansion of the code. `.code:n` carries out no expansion, whereas `.code:x` carries out an `\edef`-like procedure:

```
\tl_set:Nn \l_tmp_tl { You~said }
\keys_define:nn { module } {
  key-one .code:x = \l_tmp_tl\~``#1'',
  key-two .code:n = \l_tmp_tl\~``#1'',
}
\tl_set:Nn \l_tmp_tl { You~typed }
```

If inside the document body we then do

```
\keys_set:nn { module } {
  key-one = text      ,
  key-two = more~text ,
}
```

the result will be to print 'You said "text"' followed by 'You typed "more text"'.

5.1 Storing values

One of the most common tasks to carry out using key–value methods is storing values in variables. While this can be done using the `.code:n` property, a series of dedicated properties are available, all of which follow the same general pattern.

```
\keys_define:nn { module } {
  key-one   .dim_set:N  = \l_module_dim  ,
  key-two   .int_set:N  = \l_module_int  ,
  key-three .skip_set:N = \l_module_skip ,
  key-four  .tl_set:N   = \l_module_tl   ,
}
```

As illustrated, each property should 'point' to a variable to store the value given. While the examples here use LaTeX3-style variables, the properties will also work with variables following LaTeX 2_ε naming conventions. Giving the setting instruction:

```
\keys_set:nn { module } {
  key-four = content
}
```

will set token list variable `\l_module_tl` to the text `content`. (As a reminder, a LaTeX3 'token list variable' is a macro which is used as a variable to store tokens, often text.)

Assignments using the `.`⟨*var*⟩`_set:N` properties are local, which is normally what you want. However, global assignments can also be made, using the `.`⟨*var*⟩`_gset:N` properties. These are set up and used in exactly the same way as the local versions:

```
\keys_define:nn { module } {
  key-one .tl_set:N  = \l_module_tl ,
  key-two .tl_gset:N = \g_module_tl ,
}
\keys_set:nn { module } {
  key-one = text, % Locally
  key-two = text  % Globally
}
```

Values can be stored in token list variables either as-given or with full (`\edef`) expansion. As the expansion takes place later it cannot be indicated using an argument specifier. Instead, two 'expand then store' properties are available:

```
\keys_define:nn { module } {
  key-one .tl_set_x:N  = \l_module_tl ,
  key-two .tl_gset_x:N = \g_module_tl ,
```

```
}
\tl_set:Nn \l_tmp_tl { text }
\keys_set:nn { module } {
  key-one = \l_tmp_tl ,
  key-two = \l_tmp_tl ,
}
\tl_set:Nn \l_tmp_tl { changed }
```

Both `\l_module_tl` and `\g_module_tl` will store '`text`', whereas with the normal `.tl_(g)set:N` property they would simply contain '`\l_tmp_tl`'.

One thing to notice is that l3keys will make sure every variable we use actually exists. So there is no need to worry about long lists of declarations along with an equally long list setting up keys.

5.2 Storing Boolean values

Boolean variables can only take true and false values, and so can be viewed as a type of multiple choice. To avoid code duplication, l3keys provides a method to set LaTeX3 Boolean variables using a pre-defined choice, using the `.bool_set:N` property. This works in much the same way as those for setting other variables, except that it will only accept the values `true` and `false`.

```
\keys_define:nn { module } {
  key .bool_set:N = \l_module_bool
}
```

It is important to note that LaTeX3 Boolean variables do *not* work in the same way as TeX or LaTeX 2ε `\if..` statements. Thus, `.bool_set` cannot be used to set the latter: you have to use `.code:n`.

5.3 Values assumed, required, forbidden

Some keys can assume a particular value is meant if only the key name is given. This is often the case with keys which can be set only to `true` or `false`: giving the key name alone is usually the same as given the `true` value. This is referred to by l3keys as a default value, and is set up using the `.default:n` property:

```
\keys_define:nn { module } {
  key .code:n    = Do stuff with #1! ,
  key .default:n = yes
}
```

With the settings above

```
\keys_set:nn { module } { key }
```

and

```
\keys_set:nn { module } { key = yes }
```

are entirely equivalent.

Alternatively, rather than assume a particular value is meant if the key name alone is given, you might wish to always require a value or forbid one entirely. This can be controlled using the `.value_required:` and `.value_forbidden:` properties, both of which act in an obvious way:

```
\keys_define:nn { module } {
  key-one .code:n = Do stuff with #1,
  key-one .value_required:          ,
  key-two .code:n = Do other stuff  ,
  key-two .value_forbidden:         ,
}
```

In both cases, error messages result if the requirement is not met.

5.4 Choices

One very useful thing to do using key–value input is to provide a list of predetermined choices. These can then be used to set up potentially complicated code patterns with a simple interface.

A key is made into a multiple choice by setting the `.choice:` property, but this does not create any valid choices! Each choice is created as a 'subkey':

```
\keys_define:nn { module } {
  key .choice:,
  key / choice-a .code:n = Choice-a code ,
  key / choice-b .code:n = Choice-b code ,
  key / choice-c .code:n = Choice-c code ,
}
```

As shown, each choice is given in the format ⟨*key*⟩ `/` ⟨*choice*⟩: the `/` character marks the boundary between the key and subkey names. It is likely that there will be some similarity between the implementation for different keys, but this is not necessary for the system to work.

To avoid the need to duplicate code between choices with very similar implementations, an automated system is available. First, the shared code is set up using the `.choice_code:n` property. A comma-separated list of choices is then given using the `.generate_choices:n` property.

```
\keys_define:nn { module } {
  key .choice_code:n = {
    Do something using either
    \l_keys_choice_tl or
    \l_keys_choice_int.
  },
  key .generate_choices:n = {
    choice-a, % Choice 0
    choice-b, % Choice 1
    choice-c, % Choice 2
    ...
  }
}
```

As illustrated, within the code `\l_keys_choice_tl` and `\l_keys_choice_int` are available. The name of the current choice (for example `choice-b`) is assigned to `\l_keys_choice_tl`, its numeric position in the list (for example `1` for `choice-b`) is assigned to `\l_keys_choice_int`. Notice that this is indexed from 0!

5.5 Keys setting keys

The final property provide by l3keys is for creating so-called meta keys: keys which themselves set other keys. Using the `.meta:n` property, we can provide a short-cut to set several things in one go.

```
\keys_define:nn { module } {
  key-one   .code:n = Some code  ,
  key-two   .code:n = Other code ,
  key-three .meta:n = {
    key-one = Value ,
    key-two = Value ,
  }
}
```

It is possible to pass on the argument given to a meta-key to its 'children' using `#1`:

```
\keys_define:nn { module } {
  key-one   .code:n = Something with #1 ,
  key-two   .code:n = Other thing #1    ,
  key-three .meta:n = {
    key-one = #1 ,
    key-two = #1 ,
  }
}
```

Almost always, the data for a meta key needs to be wrapped in braces, as it contains `,` and `=` characters.

6 Unknown keys and choices

The ability to handle input which has not been previously defined is important for flexible key–value methods. Each time a key is set using `\keys_set:nn`, after looking for the key itself l3keys checks for a special `unknown` key before issuing an error message. This key is set up in the same way as any other, and can carry out whatever function is appropriate. The name of the unknown key is available within the `unknown` key as `\l_keys_key_tl`, and can therefore be used by the attached code. A simple example would be to issue a customised error message if a key is unknown:

```
\keys_define:nn { module } {
  unknown .code:n = {
    \msg_error:nnx { module }
      { unknown-key } { \l_keys_key_tl }
  }
}
```

More sophisticated use might include creating new keys from this data, storing information in custom variables and so on.

7 LaTeX 2ε package options

As LaTeX3 development is still at the stage of creating low-level structures, the most likely use of l3keys is with LaTeX 2ε packages and classes. To enable the methods described here to be used with LaTeX 2ε package and class options, a support package l3keys2e is available to enable the appropriate processing.

As is true with any key–value package, any options created with l3keys are simply keys that have been defined when option processing takes place. So creating options means first using `\keys_define:nn` for set up, then processing the option list with the `\ProcessKeysOptions` function. This takes a single argument: the name of the module.

```
\keys_defined:nn { module } {
  option-one .code:n = ... ,
  option-two .code:n = ... ,
  ...
}
\ProcessKeysOptions { module }
```

8 An example

Putting everything together can be challenging starting from a bare description of the methods available. In my general key–value article, I included a short example package to illustrate some of the major ideas. I'll use the same scenario here, which also means that readers can compare the l3keys approach directly to keyval- and pgfkeys-based solutions.

Consider the following situation. The inexperienced LaTeX user who asked for a small package for the last article has come back, and wants to be at the cutting edge. So they've asked if you can rewrite your code from before using expl3. What they want is a package to provide one user macro, `\xmph`, which will act as an enhanced version of `\emph`. As well as italic, it should be able to make its argument bold, coloured or a combination. This should be controllable on loading the package, or during the document. Finally, a de-activation setting is requested, so that the `\xmph` macro acts exactly like `\emph`. This latter setting should be available only in the preamble, so that it will apply to the entire document body.

Looking back over your earlier solution, there is not too much to change. You decide to follow LaTeX3 conventions and adjust some of the option names slightly:

- `inactive`, a key with no value, which can be given only in the preamble;

- `use-italic`, a Boolean option for making the text italic;
- `use-bold` and `use-colour`, two more Boolean options with obvious meanings;
- `colour`, a string option to set the colour to use when the `use-colour` option is true.

You also anticipate that US users would prefer the option names `use-color` and `color`, and so you decide to implement them as well.

Things are going to look a bit different from a traditional LaTeX 2ε package, but hopefully things will not be too bad! The first stage is to declare the code as a LaTeX3 package, and to load color for colour support, l3keys2e to do the option processing, and xparse to make user commands the LaTeX3 way.

```
\RequirePackage{color,l3keys2e,xparse}
\ProvidesExplPackage
  {xmph} {2010/01/02}
  {2.0}  {Extended emph}
```

The next stage is to set up the key–value input, and set the default values (red italic text).

```
\keys_define:nn { xmph } {
  colour
    .tl_set:N   = \l_xmph_colour_tl   ,
  color
    .meta:n     = { colour = #1 }     ,
  inactive
    .code:n     =
      \cs_set_eq:NN \xmph \emph       ,
  use-bold
    .bool_set:N = \l_xmph_bold_bool   ,
  use-colour
    .bool_set:N = \l_xmph_colour_bool ,
  use-color
    .bool_set:N = \l_xmph_colour_bool ,
  use-italic
    .bool_set:N = \l_xmph_italic_bool ,
}
\keys_set:nn { xmph } {
  colour = red ,
  use-italic
}
```

With everything set up, any load-time options can be dealt with using the `\ProcessKeysOptions` function.

```
\ProcessKeysOptions { xmph }
```

For the code implementing everything, the pattern here is the same as in the LaTeX 2ε version. The formatting functions are wrapped up one inside another.

```
\NewDocumentCommand \xmph { m } {
  \xmph_emph:n {
    \xmph_bold:n {
      \xmph_colour:n {#1}}}}
\cs_new:Nn \xmph_bold:n {
  \bool_if:NTF \l_xmph_bold_bool {
    \textbf {#1}
  }{#1}}
\cs_new:Nn \xmph_colour:n {
  \bool_if:NTF \l_xmph_colour_bool {
    \textcolor { \l_xmph_colour_tl } {#1}
  }{#1}}
\cs_new:Nn \xmph_emph:n {
  \bool_if:NTF \l_xmph_italic_bool {
    \emph {#1}
  }{#1}}
```

The last job to do is to disable the `inactive` at the end of the preamble. That simply means setting the option to do nothing.

```
\AtBeginDocument {
  \keys_define:nn { xmph } {
    inactive .code:n = { }
  }
}
```

9 Conclusions

Key–value methods are a powerful way to provide users with a clear interface to code internals. expl3 adds the ability to create key–value input to LaTeX, along with the many other programming refinements it provides. By including this in the base layer of LaTeX3, the confusion between LaTeX 2ε implementations is avoided. This should mean that more people can get to grips with using key–value methods in their packages, and do so more reliably.

References

LaTeX3 Project. "The expl3 package". Available from CTAN, `macros/latex/contrib/expl3`, 2010.

Tantau, Till. "pgfkeys". Part of the Ti*k*Z and pgf bundle, available from CTAN, `graphics/pgf`, 2008.

Wright, Joseph. "LaTeX3 programming: External perspectives". *TUGboat* **31**, 2010.

Wright, Joseph, and C. Feuersänger. "Implementing key–value input: an introduction". *TUGboat* **31**, 2010.

◇ Joseph Wright
Morning Star
2, Dowthorpe End
Earls Barton
Northampton NN6 0NH
United Kingdom
joseph dot wright (at)
morningstar2 dot co dot uk

ConTEXt basics for users: Conditional processing

Aditya Mahajan

Abstract

Very often, you want to generate multiple versions of the same document: one version for printing and one for viewing on the screen, one version for students and one version for the instructor, and so on. You can do this in a simple but naive way: create different files set up for the different versions and `\input` the common material, or create some new conditional flags using `\newif` and set them appropriately for conditional processing. Or you could use *modes* — the ConTEXt way of doing conditional processing.

1 Introduction

A mode is similar to a conditional flag, but with a few advantages: new modes need not be explicitly defined (no need for something like `\newif`), multiple modes can be simultaneously enabled or disabled, and the status of multiple modes can be checked easily. Moreover, modes can be set from a command line switch. As a result, multiple versions of a document can be generated without changing the source file.

The name or identifier of a mode can be any combination of letters, digits, or spaces. Names starting with `*` are reserved for system modes.

In this article I explain how to activate a mode and how to check if a mode is active or not.

2 Setting modes

ConTEXt has three commands for setting modes:

- `\enablemode [...]`
- `\disablemode[...]`
- `\preventmode[...]`

The names are self-descriptive. `\enablemode` activates a mode, `\disablemode` deactivates a mode, and `\preventmode` permanently deactivates a mode. All three commands take a list of modes as an argument. For example, you can activate modes named `screen` and `solution` with

```
\enablemode[screen,solution]
```

Modes can also be activated by a command line switch `--modes` to `texexec` or `context`. For example, another way to activate the `screen` and `solution` modes, to run ConTEXt using one of:

```
texexec --mode=screen,solution ...
context --mode=screen,solution ...
```

3 Conditional processing based on modes

You may want to process or ignore a chunk of code if a particular mode is enabled or disabled. Such a chunk of code is specified using `\startmode` and `\startnotmode` environments. Their usage is best explained by an example.

Suppose you want to change the paper size of a document depending on whether it is for print or screen. This can be done in multiple ways. You could set the default paper size for print and change it in screen mode:

```
\setuppapersize[letter][letter]
\startmode[screen]
  \setuppapersize[S6][S6]
\stopmode
```

(S6 is one of the screen-optimized paper sizes in ConTEXt; the paper size has a 4:3 aspect ratio and a width equal to the width of A4 paper.)

Alternatively, you could set a default paper size for the screen and change it if screen mode is not enabled:

```
\setuppapersize[S6][S6]
\startnotmode[screen]
  \setuppapersize[letter][letter]
\stopnotmode
```

`\startmode` and `\startnotmode` can check for multiple modes, by giving a list of modes as their arguments. `\startmode` processes its contents (everything until the next `\stopmode`, thus `\startmode` cannot be nested) if any of the modes are enabled, otherwise (i.e., when all the modes are disabled) `\startmode` ignores its contents. The opposite is `\startnotmode`: it processes its contents (everything until the next `\stopnotmode`) if any of the modes are disabled, otherwise — when all the modes are enabled — the contents are ignored.

`\startmode` and `\startnotmode` are "*or*" environments. They process their contents if any of the modes satisfy the required condition. Their "*and*" counterparts are also available: `\startallmodes` and `\startnotallmodes` process their contents only if all the given modes satisfy the required condition. For example, suppose you want to enable interaction (e.g., hyperlinks) only when both `screen` and `solution` modes are enabled. Then you can use:

```
\startallmodes[screen,solution]
  \setupinteraction[state=start]
\stopallmodes
```

To summarize, the four start-stop environments for checking modes are:

```
\startmode[mode1, mode2, ...]
  % Processed if any of the modes is enabled
\stopmode
```

```
\startnotmode[mode1, mode2, ...]
  % Processed if any of the modes is disabled
\stopnotmode

\startallmodes[mode1, mode2, ...]
  % Processed if all the modes are enabled
\stopallmodes

\startnotallmodes[mode1, mode2, ...]
  % Processed if all the modes are disabled
\stopnotallmodes
```

These environments have `\doif...` alternatives that are useful for short setups. Also, they can be nested.

```
\doifmode        {modes} {content}
\doifnotmode     {modes} {content}
\doifallmodes    {modes} {content}
\doifnotallmodes {modes} {content}
```

The logic for determining when the content is processed is exactly the same as for the `start-stop` commands.

These `\doif` commands each have a variant to process alternative code if the conditions are not satisfied (like the `\else` branch of `\if`).

```
\doifmodeelse       {modes} {content} {alt}
\doifnotmodeelse    {modes} {content} {alt}
\doifallmodeselse   {modes} {content} {alt}
\doifnotallmodeselse{modes} {content} {alt}
```

4 System modes

Besides allowing user-definable modes, ConTEXt provides some system modes. These modes start with a `*` character. Here I will explain only the more commonly used system modes; see the ConTEXt modes manual (`http://pragma-ade.com/general/manuals/mmodes.pdf`) for a complete list.

Perhaps the most useful system modes are `*mkii` and `*mkiv` which determine whether MkII or MkIV is being used. These modes are handy when you want different setups for MkII and MkIV.

Other modes are useful for very specific situations. Some of these are described below.

A document must be run multiple times to get the cross referencing, table of contents, etc. right. However, sometimes you need to do some external processing (e.g., graphic conversion) that only needs to be done once. In such cases, the `*first` mode is handy — it is active only on the first run of the document.

You can use the project-product-component structure for managing large projects like a book series. See the ConTEXt wiki article (`http://wiki.contextgarden.net/Project_structure`) for details of this approach. A product or its components may be compiled separately, and you may want to do something different when a product is compiled or when a component is compiled. To do so, you need to check for modes `*project`, `*product`, `*component`, and `*environment`; these modes are set when the corresponding structure file is processed. For example, the `*product` mode is set whenever a product file is read; more specifically, when `\startproduct` is encountered. Similarly, a mode `*text` is enabled when `\starttext` is encountered, and likewise for the others.

A large document is typically broken down into different section blocks: frontmatter, bodymatter, appendices, and backmatter. Internally, these section blocks are referred to as `frontpart`, `bodypart`, `appendix`, and `backpart`. Each section block sets a system mode with the same name. So, if you want macros that work differently in different section blocks, you can check for modes `*frontpart`, `*bodypart`, and so on.

ConTEXt provides support for multiple languages. Languages are recognized by their IETF language tags, like `en-us` for US English, `en-gb` for British English, `nl` for Dutch, `de` for German, etc. A document has a main language, set with the command `\mainlanguage[...]`, that is used for translated labels like *chapter* and *figure*. You can also switch the current language using `\language[...]` to change the hyphenation rules. Whenever a language is chosen, its identifier is set as a mode. The mode for the main language starts with two `*`. For example, when the main language is US English and the current language is Dutch, the modes `**en-us` and `*nl` are set (notice the extra `*` in `**en-us`).

Other system modes: `*figure` is set when a graphic is found, `*interaction` is set when interaction is enabled, `*grid` is set when grid typesetting is enabled, and `*pdf` and `*dvi` are set when the output is PDF or DVI. Others are too esoteric to describe here. If you are interested, see the modes manual mentioned earlier.

In summary, modes provide generalized conditional processing. A rich set of built-in modes is available.

◇ Aditya Mahajan
`adityam (at) ieee dot org`

Glisterings

Peter Wilson

> His eye, which scornfully glisters like fire,
> Shows his hot courage and his high desire.
>
> *Venus and Adonis*, William Shakespeare

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

Corrections, suggestions, and contributions will always be welcome.

> Words are wise men's counters, they do but reckon with them, but they are the money of fools.
>
> *Leviathan*, Thomas Hobbes

1 Counting

1.1 Number of words

Some publications put word limits on manuscripts, and the question often arises as what is the (best) way to count them. This is answered in detail in the FAQ [1] but my answer is to simply count the number of words on one page of your manuscript and multiply by the number of pages. This is essentially the technique used by book designers and publishers when confronted with a manuscript in the process called *casting off* (see, for example, [3, Chap. 8]). The publishers are not particularly interested in the exact number of words but are very much interested in the number of pages in the final production.

If you are writing a thesis the powers-that-be may specify a word limit, but it is probably safe to assume that they will not actually check the number of words themselves, unless there is an obvious mismatch between your number and the size of the thesis. In any event, what counts as a 'word'? Is 'powers-that-be' one word or three? How many 'words' are the equivalent of a table or a figure? If different sized fonts will be used, are all 'words' equal? What about footnotes, mathematical equations, verse — how many 'words' should be allocated to them?

1.2 Lua

By the time you read this LuaTEX should be available, and perhaps you have used it. At the time of writing it is still being developed and I have not tried it. However, assuming that Lua [2] will be available on all TEX platforms, I thought that I would try and use it for its own sake.

I have been fortunate in being able to use lead type and a hand operated press, much as Gutenberg did in the 15th century. Unlike digital typesetting where you can have an unlimited number of characters of any particular kind, the number of available characters is strictly limited — if the font you are using has only 23 'e' sorts (a *sort* is a single piece of lead type), then in one go you can only set text that has no more than 23 'e' characters. It is therefore important to know how many of each sort is required to set a page of text. For example, I wanted to print a 16th century poem — only 2 verses on one page — in a particular font but I couldn't do so as I was one 'h' short (there were a lot of thee, thou, thine, ... eth, etc., words compared to modern English). I work on a Linux system which provides programs for counting the number of words and the total number of characters in a piece of text; presumably other systems provide the equivalent. But what I wanted was a program to count the numbers of the individual characters — the number of 'A' characters, the number of 'a' characters, and so on.

I managed to extend a Lua program that would do this for me. Here it is, in a file that is called `gwc.lua`:

```
#!/usr/local/bin/lua5.1
-- gwc.lua Lua program to count characters, etc
-- (see Lua Manual p.198)
-- call as: gwc.lua file

local BUFSIZE = 2^13        -- 8k
local f = io.input(arg[1])  -- open input file
local cc = 0       -- count of chars
local lc = 0       -- count of lines
local wc = 0       -- count of words
local ct = {}      -- table of char counts
local k, v         -- table key and value
for i = 32,126 do  -- initialise ASCII slots
  ct[i] = 0
end
local T = 0        -- my total chars
local tc = 0       -- actual total chars
                   -- (no newlines, etc)
while true do
  -- read a chunk of text
  local lines, rest = f:read(BUFSIZE, "*line")
  if not lines then break end
  if rest then
     lines = lines .. rest  .. "\n" end
  cc = cc + #lines
  -- count words in the chunk
  local _, t = string.gsub(lines, "%S+", "")
  wc = wc + t
  -- count newlines in the chunk
  _, t = string.gsub(lines, "\n", "\n")
  lc = lc + t
```

```
-- make a list of character frequencies
  local K
  for i = 1, string.len(lines) do
    K = string.byte(lines,i)
    if K > 32 then
     if K < 126 then
      ct[K] =  ct[K] + 1
      T = T + 1
     end
   end
  end
end

-- strip off input (e.g., fin.ext) file's
-- extension and make output file fin.gwc
base, ext = string.match(arg[1],
                  "(%w+)%.(%w+)")
ofile = base..".gwc"

-- cc includes newlines, so T = (lc + wc)
tc = cc - lc - wc
io.output(ofile)
io.write("Character counts in file ",
         arg[1], "\n")
io.write("", "lines =", lc, "\n",
             "words =", wc, "\n",
             "characters = ", tc, "\n\n")

io.write("Character total\n")
for k,v in pairs(ct) do
  if v > 0 then
    print(string.char(k),v)
    io.write("  ", string.char(k), "       ",
             string.format("%4d",v), "\n")
  end
end
print("Output saved in: ", ofile)
```

That ends the Lua program. In this case a 'word' is a sequence of characters followed by one or more spaces. I was only interested in characters corresponding to the sorts in the fonts that were available to me. Being English this fortunately restricted the characters to the ASCII printable character set. If you need to count other characters then you will have to extend the program. The Lua manual [2, p. 198] describes how the word and line count part of the program works in more detail.

> Change is not made without inconvenience, even from worse to better.
>
> *A Dictionary of the English Language: Preface*, Samuel Johnson

2 Changing the layout

A question that pops up from time to time is 'How do I change the layout for a particular page?', where the 'layout' includes items like the size and location of the textblock, and different headers and footers.

2.1 The shape of the page

You can do many things, but one that you cannot do is to change the textwidth in the middle of a paragraph. For instance if the textblock is 30pc wide on one page and 25pc wide on the following page, then a paragraph that starts on the first page and continues onto the next will be 30pc wide on both pages. This is because TEX internally typesets paragraph by paragraph according to the current textwidth. Having set a paragraph it then decides if there should be a pagebreak in it. If there is it puts the beginning of the already laid out paragraph on the first page and the remainder, which is already set internally, goes on the following page(s) with the *same* textwidth.

The general page layout parameters are diagrammed in Figure 1.

To change the height of the textblock on a particular page, the LATEX `\enlargethispage` macro can be used. This takes a single length argument which is added to the textheight for the page on which it occurs — a positive length increases the textheight and a negative one decreases it. The change is made at the bottom of the textblock; the location of the top of the textblock is unchanged.

The `quote` and `quotation` environments temporarily change the margins and width of the textblock, and you can do the same by using, for example, the `adjustwidth` environment provided by the `changepage` package [5].

> The `adjustwidth` environment takes two length arguments, and increases the left and right margins by the given amounts. For example, I used
> `\begin{adjustwidth}{3em}{1.5em}`
> at the start of this paragraph, and will end `adjustwidth` at the end of the paragraph.

The page layout parameters used are those in effect at the start of a page when the first item (e.g., a character, a box, etc.) is put onto the page. Layout changes after that will not be effective until the start of the next page. You can, though, change the text width *between* pages. The trick here is that when you change from one column to two columns, or vice versa, LATEX recalculates its view of the layout. The general scheme is to clear the page, change the layout parameters, then set the number of columns which starts the same new page again but with the layout changes implemented. Assuming a one column document, the general procedure is:

The circle is at 1 inch from the top and left of the page. Dashed lines represent (`\hoffset + 1 inch`) and (`\voffset + 1 inch`) from the top and left of the page.

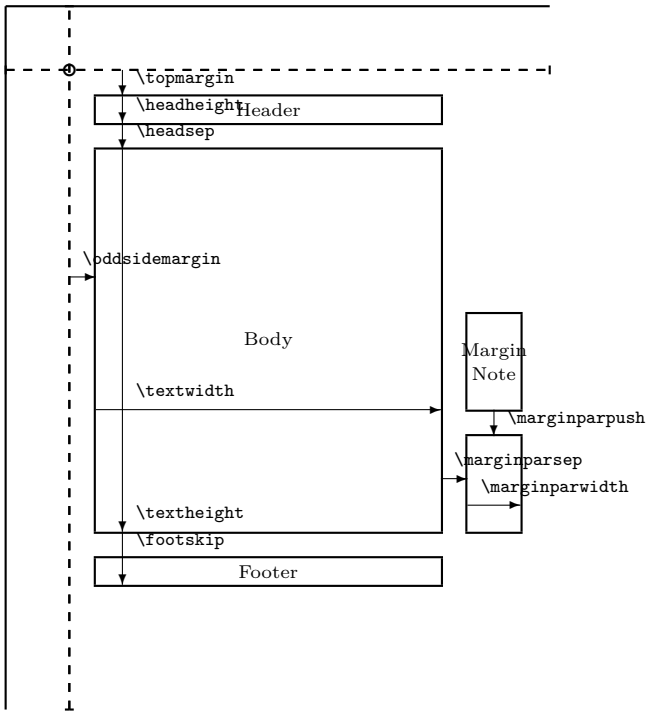


Figure 1: LaTeX page layout parameters for a recto page

```
\clearpage
% change textblock, margins, ...
\onecolumn
```

If you need them, the changepage package provides macros to '`change textblock, margins, ...`'.

Just so you can see what happens, the kernel definition of `\onecolumn` is:

```
\def\onecolumn{%
  \clearpage
  \global\columnwidth\textwidth
  \global\hsize\columnwidth
  \global\linewidth\columnwidth
  \global\@twocolumnfalse
  \col@number \@ne
  \@floatplacement}
```

The code for `\twocolumn` is similar but does a little more, especially as it takes an optional argument although that has no effect on the various width settings.

As an example, if you needed to have different text heights and widths for one set of pages, those in the frontmatter perhaps, than for another set, say the rest of the work, you could define

```
\newcommand*{\addtotextheightwidth}[2]{%
  \clearpage
  \addtolength{\textheight}{#1}
  \addtolength{\textwidth}{#2}
  \onecolumn}
```

and use it when you need to make a change.

2.2 Headers and footers

Another kind of layout change that I have seen requested is to add 'Page' above the page numbers in the Table of Contents or List of Figures, etc. As an example say that the requirement is that for the List of Figures (LoF) the word 'Figure' should be placed flushleft at the start of the column of figure titles and the word 'Page' flushright above the page numbers; if the LoF continues for more than one page, these should be repeated at the start of each page. The page number(s) of the LoF itself should be centered at the bottom of the page (i.e., the plain pagestyle). There are similar requirements for the Table of Contents (ToC) and List of Tables (LoT), but I'll just show how the LoF requirements can be met.

Changing pagestyles can be accomplished with the fancyhdr package [4] but I will assume that the memoir class [6] is being used which includes similar facilities.

The memoir class lets you define as many pagestyles as you want. We need a pagestyle for any

LoF continuation pages (and others for the ToC and LoT). Here's the one for the LoF, which I have called the lof pagestyle. This puts the page number centered in the footer and 'Figure' at the left in the header and 'Page' at the right.

```
\makepagestyle{lof}% a new pagestyle
  \makevenfoot{lof}{}{\thepage}{} % like plain
  \makeoddfoot{lof}{}{\thepage}{} % like plain
  \makeevenhead{lof}{Figure}{}{Page}
  \makeoddhead{lof}{Figure}{}{Page}
```

When we start the LoF we need to make sure that the lof pagestyle will be used for any continuation pages. We can do this by adding the necessary code to the `\listoffigures` command, and memoir provides the `\addtodef` command for doing this. It takes three arguments, the first is the name of a macro, the second is code to be added at the start of the macro's definition and the third is code to be added at the end of the macro's definition.

```
\addtodef{\listoffigures}{%
  \clearpage\pagestyle{lof}}{}
```

Memoir provides a command that is called before setting the title of the LoF and another that is called after the title. You can redefine these to do what you want. In this case we just need to extend what happens after the title.

```
\renewcommand*{\afterloftitle}{%
  \thispagestyle{plain}
  \par\nobreak
  {\normalfont\normalsize Figure \hfill Page}
  \par\nobreak}
```

The above makes the first page of the LoF use the plain pagestyle, and then puts a line containing 'Figure' at the left and 'Page' at the right. The actual listing of titles and page numbers will start after these preliminaries.

Setting up the ToC and LoT is almost identical to the above, but with names changed.

One thing to watch for is that after the LoF has been processed the lof pagestyle is still in effect. After the LoF has finished it will be necessary to revert back to the regular pagestyle which, for the sake of argument, let's say is heads. To be on the safe side the general scheme, then, is:

```
\documentclass[...]{memoir}
%% define heads pagestyle
%% ToC, LoF, ToC changes
%% more preamble
\addtodef{\mainmatter}{}{\pagestyle{heads}}
\pagestyle{heads}
\begin{document}
%% title pages
%% maybe Preface and such
%% \tableofcontents\clearpage\pagestyle{heads}
%% \listoffigures\clearpage\pagestyle{heads}
%% \listoftables\clearpage\pagestyle{heads}
%% other prelims
\mainmatter
...
\end{document}
```

which ensures that at the start of the main matter the regular heads pagestyle is in effect, no matter what games were played beforehand.

References

[1] Robin Fairbairns. The UK TEX FAQ. Available on CTAN in `help/uk-tex-faq`.

[2] Roberto Ierusalimschy. *Programming in Lua, Second Edition*. Lua.org, Rio de Janeiro, 2006. ISBN 85-903798-2-5.

[3] Ruari McLean. *The Thames and Hudson Manual of Typography*. Thames and Hudson, 1980. ISBN 0-500-68022-1.

[4] Piet van Oostrum. Page layout in LATEX, 2004. Available on CTAN in `latex/macros/contrib/fancyhdr`.

[5] Peter Wilson. The changepage package, 2008. Available on CTAN in `latex/macros/contrib/misc/changepage.sty`.

[6] Peter Wilson. The memoir class for configurable typesetting, 2009. Available on CTAN in `latex/macros/contrib/memoir`.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries dot press (at)
earthlink dot net

The exact placement of superscripts and subscripts

Timothy Hall

Introduction

Consider the following segment of mathematics in "uncramped" display (D) style.

$$\theta^{At}\gamma_{jk}$$

It is conceivable that an author would want the At term to be higher on the θ, since the base of the At is apparently all the way down to the midline of the θ. Such higher placement might make room for different levels of superscripts, like those used in tensor notation. This need commonly occurs when the nucleus, in this case θ, has zero depth and a height that is greater than its width. That same author might also want the jk to be lower on the γ, since, on quick inspection, the jk might appear to be on the same level as the γ (making it a product of three terms). This situation commonly occurs when the nucleus has a descender and the subscript has an ascender (and where a significantly large nucleus depth exists compared to the ascender height).

In both these cases, it would be impractical for the author to address these issues by changing to a completely different font that has more appealing font dimensions, or to use a specially designed character, when its presence in other contexts would have just as many problems as before.

As is usually the case with TEX, there is a way to address these issues with a scalpel and not a club. Individual elements in a mathematical expression may be precisely placed to an author's exact requirements through the careful manipulation of local font dimensions, without the need of creating new fonts or using custom character glyphs.

The placement of superscripts and subscripts when generating boxes from formulas follows the rules set forth in Appendix G of *The TEXbook*. In particular, Rules 18a, 18c, and 18d apply to the first situation, where there is a superscript on θ but not a subscript, and Rules 18a and 18b apply to the second situation, where there is a subscript on γ but not a superscript. Other combinations of the rules apply when there is both a superscript and a subscript on the same nucleus.

Superscript placement

Consider the first situation, where an author wishes to raise the placement of the At term slightly higher. The standard placement rules under this circumstance are as follows.

18a. Since the nucleus is a character box, set $u = 0$.

18c. Set box X to the contents of the superscript, here At, in scriptstyle, and add `\scriptspace` to the width of X. Then set

$$u \leftarrow \max\left\{u, \sigma_{13}, d\,(\theta) + \frac{1}{4}\,|\sigma_5|\right\},$$

since we are in display style, where $d\,(\theta)$ is the depth of the θ character, σ_{13} is font dimension 13 of `\textfont2` (the so-called "sup1" dimension), and σ_5 is font dimension 5 of `\textfont2` (the so-called "x-height"). Note that $|\sigma_5|$ is used since the x-height of a font may (conceivably) be negative.

18d. Append box X to the horizontal list, shifting it up by u.

These simple steps show that the only way to influence the placement of the superscript box X (without changing the dimensions of the nucleus itself) is to temporarily change the 5^{th} and/or 13^{th} font dimension of `\textfont2`. The most direct way of doing so is to increase σ_{13} by an amount that exceeds $d\,(\theta) + \frac{1}{4}\,|\sigma_5|$. For Plain TEX, `\textfont2` is `\tensy` (an alias for `cmsy10`), and $d\,(\theta) = 0$, $\sigma_5 = 4.30554$ pt, and $\sigma_{13} = 4.12892$ pt. This means $u = \sigma_{13}$ when θ^{At} is translated under Plain TEX into a horizontal list. However, the height of θ is 6.94444 pt (which shows that the superscript is normally placed $\frac{4.12892}{6.94444} \approx 59.456\%$ of the way up θ from the baseline), so that $\sigma_{13} = 7$ pt would place the baseline of At less than $\frac{1}{10}$ pt above the top of the θ. Comparing these two options, we have

$$\theta^{At} \quad \text{versus} \quad \theta^{At}$$

which was produced by the following TEX code.

```
\tmp=\the\fontdimen14\tensy
\centerline{$\theta^{At}$\quad
   {\rm versus}\quad
   $\fontdimen14\tensy=7pt
   \theta^{At}$}
% See note below about the
% use of font dimension 14.
\fontdimen14\tensy=\tmp
```

Since font dimension changes are, by default, global, any such changes must be explicitly reversed. This is the purpose of the `\tmp` dimension register in the previous example. Note also that it is the font dimensions of `\textfont2` that are changed to produce the desired results, and not the dimensions

of \scriptfont2, even though the placement of the "script" parts are affected.

For expressions that have only superscripts, in regular math mode, i.e., not in display style, the 14^{th} font dimension (sup2) of \textfont2 would be changed (instead of the 13^{th} one as before), and in "cramped" display style, the 15^{th} font dimension (sup3) of \textfont2 would be changed.

Subscript placement

The procedure for subscript placement is similar to that of superscript placement, except that (a) it is slightly more straightforward, except (b) it uses a "cramped" style. To wit:

18a. Since the nucleus is a character box, set $v = 0$.

18b. Set box X to the contents of the subscript, here jk, in "cramped" scriptstyle, and add \scriptspace to the width of X. Append box X to the horizontal list, shifting it down by

$$\max\left\{v, \sigma_{16}, h\left(\gamma\right) - \frac{4}{5}\left|\sigma_5\right|\right\},$$

since we are in display style, where $h\left(\gamma\right)$ is the height of the γ character, σ_{16} is font dimension 16 of \textfont2 (the so-called "sub1" dimension), and σ_5 is font dimension 5 of \textfont2 (the so-called "x-height"). Note that $|\sigma_5|$ is used since the x-height of a font may (conceivably) be negative.

We have $h\left(\gamma\right) = 4.30554$ pt, $\sigma_5 = 4.30554$ pt, and $\sigma_{16} = 1.49998$ pt. This means the subscript is lowered by $\sigma_{16} = 1.49998$ pt under Plain TEX. However, the depth of γ is 1.94444 pt, and the height of k in scriptstyle is 4.8611 pt. This means $\sigma_{16} = (1.94444 + 4.8611)$ pt $= 6.8055$ pt would place the top of jk at the bottom of the γ. Comparing these two options, we have

$$\gamma_{jk} \quad \text{versus} \quad \gamma_{\,jk}$$

which was produced by the following TEX code.

```
\tmp=\the\fontdimen16\tensy
\centerline{$\gamma_{jk}$\quad
   {\rm versus}\quad
   $\fontdimen16\tensy=6.8055pt
   \gamma_{jk}$}
\fontdimen16\tensy=\tmp
```

Note that for expressions that have only subscripts, the 16^{th} font dimension of \textfont2 would be changed regardless of the style where the expression occurs.

Optional changes

Additional aesthetic changes for the second example might include moving the subscripts slightly closer to the γ due to the shape of the j. This may be accomplished by a small kern in the subscript, such as

```
\gamma_{\mkern-2.7mu jk}
```

However, considering how small the differences may be relative to the font dimensions, it may be difficult to distinguish the horizontally unadjusted version with the adjusted version.

$$\gamma_{jk} \quad \text{versus} \quad \gamma_{\,jk}$$

Both types of scripts together

Rule 18a contains more complicated instructions when the nucleus of the expression is not simply a character, and Rules 18d, 18e, and 18f contain even more complicated instructions when there is a superscript *and* a subscript in the same expression. Taken collectively, the effect of this latter set of rules is not the same as simply placing the superscript independently of the subscript, one after the other, in either order. These two more challenging circumstances are not covered here; everything an author needs to know, and a lot more, to make arbitrary changes in superscript and subscript placements, is in Appendix G of *The TEXbook*. It suffices to state that no matter how exotic the expression becomes, the placement of superscripts and subscripts in TEX is enormously flexible due to the ability to change a few font dimensions at strategic positions.

◇ Timothy Hall
PQI Consulting
P. O. Box 425616
Cambridge, MA 02142-0012
tgh (at) pqic dot com

The Treasure Chest

This is a list of selected new packages posted to CTAN (`http://ctan.org`) from July 2009 through April 2010, with descriptions based on the announcements and edited for brevity.

Entries are listed alphabetically within CTAN directories. A few entries which the editors subjectively believed to be of especially wide interest or otherwise notable are starred; of course, this is not intended to slight the other contributions.

We hope this column and its companions will help to make CTAN a more accessible resource to the TEX community. Comments are welcome, as always.

◇ Karl Berry
`http://tug.org/ctan.html`

biblio

`biber` in `biblio`
BibTEX replacement for `biblatex` users, in Perl.

`persian-bib` in `biblio/bibtex/contrib`
Persian translations of standard BibTEX styles.

fonts

`ccicons` in `fonts`
LATEX support for Creative Commons icons.

`jablantile` in `fonts`
New Metafont font from Don Knuth to implement the modular tiles described by Slavik Jablan.

`mathgifg` in `fonts`
Text and math LATEX support for Microsoft Georgia and ITC Franklin Gothic.

`oldlatin` in `fonts`
Computer Modern with "long s" used in old texts.

`softmaker-freefont` in `fonts`
Support for SoftMaker fonts from `freefont.de`.

`txfontsb` in `fonts`
Small caps and oldstyle numerals for `txfonts`, and Greek support through Babel.

`zhmetrics` in `fonts`
TFM subfonts to support Chinese in 8-bit TEX.

graphics

`drv` in `graphics/metapost/contrib/macros`
Drawing derivation trees in MetaPost.

`fig4latex` in `graphics`
Makefile support for `xfig` graphics exported to LATEX.

`garrigues` in `graphics/metapost/contrib/macros`
Drawing Garrigues' Easter nomogram.

`jflap2tikz` in `graphics`
Convert finite automata from JFlap to Ti*k*Z.

`pgf-umlsd` in `graphics/pgf/contrib`
UML sequence diagrams.

`pst-am` in `graphics/pstricks/contrib`
(De)modulation of radio waves.

`pst-abspos` in `graphics/pstricks/contrib`
Absolute or relative page positioning.

`pst-knot` in `graphics/pstricks/contrib`
Drawing knots.

`pst-mirror` in `graphics/pstricks/contrib`
Three-dimensional objects on a spherical mirror.

`pst-node` in `graphics/pstricks/contrib`
Drawing nodes and their connections.

`pst-platon` in `graphics/pstricks/contrib`
Drawing the five Platonic solids in a 3D view.

`pst-plot` in `graphics/pstricks/contrib`
Drawing functions and data records.

`pst-thick` in `graphics/pstricks/contrib`
Drawing and filling of very thick lines and curves.

`suanpan` in `graphics/metapost/contrib/macros`
Drawing Chinese and Japanese abaci.

`tikz-3dplot` in `graphics/pgf/contrib`
Coordinate transformations for 3D plots.

`tikz-qtree` in `graphics/pgf/contrib`
Simpler syntax and better layout for trees in Ti*k*Z.

`tkz-orm` in `graphics/pgf/contrib`
Drawing Object-Role Model (ORM) diagrams.

info

`apprends-latex` in `index`
French manual for LATEX beginners.

`asy-faq/zh-cn` in `info`
Chinese translation of the Asymptote FAQ.

`asymptote-by-example-zh-cn` in `info`
An Asymptote tutorial in Simplified Chinese.

`asymptote-manual-zh-cn` in `info`
Chinese translation of the Asymptote manual.

`context-notes-zh-cn` in `info`
ConTEXt tutorial in Chinese.

`context-top-ten` in `info`
Most common ConTEXt commands.

`ctex-faq` in `info`
LATEX FAQ for Chinese TEX users.

`latexcheat-es_MX` in `info`
Spanish translation of `latexcheat`.

`lshort-persian` in `info`
Persian translation of `lshort`.

`memdesign` in `info`
A few notes on book design, by Peter Wilson.

`Presentations_de` in `info/examples`
Examples from the book *Presentationen mit LATEX*.

language

`bardi` in `language`
Hyphenation and language support for Bardi (BCJ).

`cjkpunct` in `language/chinese`
Adjust location and kerning of CJK punctuation.

`ctex` in `language/chinese`
Classes and macros for Chinese typesetting.

`kmrhyph` in `language/hyphenation`
Hyphenation patterns for Kurmanji (T1-encoded).

`turkmen` in `language`
Turkmen definitions for Babel.

macros/context

`context-ruby` in `macros/context/contrib`
W3C's "ruby", short runs by base text.

macros/latex/contrib

* `background` in `macros/latex/contrib`
Background material on pages of a document.

`bigints` in `macros/latex/contrib`
Producing big integrals.

`boolexpr` in `macros/latex/contrib`
Expandable evaluation of boolean expressions.

`bracketkey` in `macros/latex/contrib`
Bracket keys for genealogies.

`cachepic` in `macros/latex/contrib`
Lua script and package to convert document fragments to graphics.

`censor` in `macros/latex/contrib`
Support for redacting material.

`changelayout` in `macros/latex/contrib`
Change layout of individual pages.

`combelow` in `macros/latex/contrib`
Typeset letters with a comma-below accent.

`combinedgraphics` in `macros/latex/contrib`
Extended options for including EPS or PDF.

`cookybooky` in `macros/latex/contrib`
Typeset recipes.

`currfile` in `macros/latex/contrib`
Macros for file name and path of input files.

`dashundergaps` in `macros/latex/contrib`
Underline some specified, possibly invisible, text.

`docmute` in `macros/latex/contrib`
Use standalone documents with `\input` or `\include`.

`dox` in `macros/latex/contrib`
Extensions to the `doc` package.

`elbioimp` in `macros/latex/contrib`
Journal of Electrical Bioimpedance support.

`engtlc` in `macros/latex/contrib`
Telecommunications engineering support.

`eqexam` in `macros/latex/contrib`
Exam generation, with additional features for `AeB`.

`erdc` in `macros/latex/contrib`
Technical information reports of the US Army Engineer Research and Development Center.

`estcpmm` in `macros/latex/contrib`
Munitions management reports.

* `etextools` in `macros/latex/contrib`
ε-TEX tools for LATEX.

`fancypar` in `macros/latex/contrib`
Decorative styles for individual text paragraphs.

`fc_arith` in `macros/latex/contrib`
Create an arithmetic flash card.

`filehook` in `macros/latex/contrib`
Hooks for input files.

`flashmovie` in `macros/latex/contrib`
Embed flash movies in PDF output.

`ftnxtra` in `macros/latex/contrib`
Make `\footnote` work in `\caption`, `\chapter`, etc.

`germkorr` in `macros/latex/contrib`
Change kerning for German quotation marks.

`hvindex` in `macros/latex/contrib`
Facilitate index typesetting.

`idxlayout` in `macros/latex/contrib`
Key–value interface to configure index layout, supporting KOMA-script and Memoir.

`iftex` in `macros/latex/contrib`
Am I running under pdfTEX or XETEX or LuaTEX?

`keycommand` in `macros/latex/contrib`
Natural way to define commands with optional keys.

`knitting` in `macros/latex/contrib`
Fonts and (LA)TEX macros for knitting charts.

`knittingpattern` in `macros/latex/contrib`
Formatting knitting patterns.

`librarian` in `macros/latex/contrib`
Extract material from `.bib` files, for typesetting in plain, LATEX, and ConTEXt.

`listings-ext` in `macros/latex/contrib`
Shell script and macros to input parts of programs into a document.

`ltxnew` in `macros/latex/contrib`
Generalized `\new`, `\renew`, and `\provide` prefixes.

`mailmerge` in `macros/latex/contrib`
Instantiate a template from successive actual texts.

`mdframed` in `macros/latex/contrib`
Automatically split framed environments.

`minted` in `macros/latex/contrib`
Highlight LATEX sources using Pygments.

`mylatexformat` in `macros/latex/contrib`
Construct `.fmt` for fast loading from any preamble.

`newverbs` in `macros/latex/contrib`
`\verb` variants that can add TEX code before and after the verbatim text.

`ocgtools` in `macros/latex/contrib`
Manipulate OCG layers in PDF, that is, hide and reveal material via links or buttons.

onrannual in macros/latex/contrib
Office of Naval Research report class.

ot-tableau in macros/latex/contrib
Produce optimality theory tableaux.

pagerange in macros/latex/contrib
Expand page ranges.

plantslabels in macros/latex/contrib
Labels for plants.

popupmenus in macros/latex/contrib
Create popup menus via links or buttons.

properties in macros/latex/contrib
Load (key,value) properties from a file.

rangen in macros/latex/contrib
Random integers, rational, and decimal numbers.

renditions in macros/latex/contrib
Classes of comments environments.

seuthesis in macros/latex/contrib
Theses at the Southeast University, Nanjing, China.

skeyval in macros/latex/contrib
Extensions to xkeyval.

soton in macros/latex/contrib
Beamer-friendly University of Southampton palette.

spreadtab in macros/latex/contrib
Spreadsheet features for LaTeX table environments.

*spverbatim in macros/latex/contrib
Allow line breaks within \verb and verbatim.

standalone in macros/latex/contrib
Compile TeX pictures or code by themselves or included in a main document.

subsupscripts in macros/latex/contrib
Additional features for superscripts and subscripts.

tablenotes in macros/latex/contrib
Notes for tables, à la footnotes and endnotes.

tabularborder in macros/latex/contrib
Make \hline have the width of tabular text, taking account of the outer \tabcolsep.

tex-label in macros/latex/contrib
Label (classify) parts of a document for proofs.

texilikechaps in macros/latex/contrib/misc
Customizable Texinfo-like chapter headers.

threeparttablex in macros/latex/contrib
Support table notes in longtable.

thumby in macros/latex/contrib
Creating thumb indexes.

titlepic in macros/latex/contrib
Display a picture on the title page.

trimspaces in macros/latex/contrib
Remove spaces from token lists and macros.

widetable in macros/latex/contrib
Typeset tables of specified width.

*xpackages in macros/latex/contrib
High-level parts of LaTeX3 — still experimental.

xypdf in macros/latex/contrib
Improve PDF output of xypic.

ydoc in macros/latex/contrib
An alternative for documenting LaTeX packages.

macros/latex/exptl

cfr-lm in macros/latex/exptl
Enhanced support for GUST's Latin Modern fonts.

keys3 in macros/latex/exptl
Key management for LaTeX3.

macros/plain

font-change in macros/plain/contrib
Change text and math fonts in plain TeX with one command; supports all major free font families.

macros/xetex

itrans in macros/xetex/generic
ITRANS mappings for XeTeX for Devanagari and Kannada.

xecjk in macros/xetex/latex
Typesetting CJK documents in XeTeX.

xeindex in macros/xetex/generic
Automatically index specified strings in XeLaTeX.

xesearch in macros/xetex/generic
Manipulate and apply macros to (sub)strings.

zhspacing in macros/xetex/generic
Simpler typesetting of CJK documents in XeTeX.

support

lua-alt-getopt in support/lua
Lua implementation of GNU *getopt_long*.

chklref in support
Report unused labels in LaTeX.

csv2latex in support
Ruby script + Applescript for copying spreadsheet cells to LaTeX.

installfont in support
Shell script for installing a Type 1 font family.

latex-make in support
Easy compilation via GNU make with automatic dependency tracking and xfig support.

match_parens in support
Perl script for general parenthesis balancing.

*pdfjam in support
Shell script interface to pdfpages: concatenating PDF files, selecting pages, *n*-up formatting, etc.

rake4latex in support
A Ruby script to compile LaTeX projects.

ratexdb in support
Preprocessor in Ruby to query a database and create LaTeX; compatible with latexdb.

style_showcase in support
Build web page to compare LaTeX styles.

texdiff in support
Merge two LaTeX documents, for change tracking.

ArsTEXnica #8 (October 2009)

ArsTEXnica is the journal of GUIT, the Italian TEX user group (`http://www.guit.sasup.it/`).

Massimiliano Dominici and Gianluca Pignalberi, Editoriale [From the editor]; pp. 5–6

A short overview of the present issue.

Enrico Gregorio, Simboli matematici in TEX e LATEX [Mathematical symbols in TEX and LATEX]; pp. 7–24

An introduction to the primitive commands of TEX for the typesetting of mathematical formulas and to the corresponding LATEX commands, with examples and suggestions for defining new symbols in a suitable way in order to exploit the automatic spacing provided by TEX.

Agostino De Marco, Produrre grafica vettoriale di alta qualità programmando Asymptote [Introducing the high quality vector graphics programming Asymptote)]; pp. 25–39

Asymptote is a powerful programmable graphics system, distributed under the GNU GPL. It provides a high level descriptive programming language, which is based on advanced mathematical functions. Asymptote is particularly suited to produce technical drawings. It allows users to compose labels and more complicated textual objects with LATEX and this feature guarantees a high-quality typographical performance. This article does not give a complete overview of Asymptote, rather it has the aim at introducing gradually, with appropriate examples, the main elements of its programming language emphasizing the aspects which are of interest for the LATEX user.

Given the breadth of topics related to a high level programming language, the reader interested in the details and in the potential of this graphics system is advised to read carefully the references cited and to study the source code of the many predefined functions.

Kaveh Bazargan, TEX as an ebook reader; pp. 40–41

Published in *TUGboat* 30:2.

Claudio Beccari, La composizione di tabelle con larghezza specificata [Composition of specified-width tables]; pp. 42–47

This tutorial examines the LATEX kernel's basic commands necessary to typeset tabular material with specified width; pros and cons are discussed and, as an exercise, this tutorial suggests correction of some glitches with suitable macros.

Lorenzo Pantieri, L'arte di gestire la bibliografia con `biblatex` [The art of bibliography handling with `biblatex`]; pp. 48–60

The purpose of this work is to describe the basic concepts of the `biblatex` package, which offers a general solution for managing and customizing the bibliography in a LATEX document. The article requires a basic knowledge of BibTEX.

Massimiliano Dominici, LATEX e CSV [LATEX and CSV]; pp. 61–69

In this paper we will present some techniques and a few examples about handling data in comma separated value format. We will focus mainly on two packages specifically aimed at this purpose: `datatool` and `pgfplots`.

Gianluca Pignalberi, `combelow`: abbasso i segni diacritici di serie B [`combelow`: Down with second-class diacritic marks]; pp. 70–75

Should Romanian and Latvian be considered second class languages in the TEX world? Though up to now they may have been, this small package tries to raise them to the right level, by using the correct diacritic mark. No more cedilla instead of comma below.

Claudio Beccari, Uso del comando `\write18` per comporre l'indice analitico in modo sincrono [Using `\write18` command to typeset the index in a synchronous way]; pp. 76–78

Published in *TUGboat* 30:2.

Luigi Scarso, Una estensione di `luatex`: `luatex` lunatic [A `luatex` extension: `luatex` lunatic]; pp. 79–91

Published in *TUGboat* 30:3.

Emmanuele Somma, Il respawn di Infomedia (LATEX-based) [The rebirth of Infomedia (LATEX-based)]; pp. 92–101

Infomedia, famous Italian publisher of programming magazines, has been born again thanks to free software. To typeset its magazines it relies on LATEX and some other tools discussed in the paper.

Jean-Michel Hufflen, Processing "computed" texts; pp. 102–110

This article is a comparison among methods that may be used to derive texts to be typeset by a word processor. By 'derive', we mean that such texts are extracted from a larger structure. The present standard for such a structure uses XML-like format, and we give an overview of the available tools for this derivation task.

[Received from Gianluca Pignalberi.]

Die TEXnische Komödie 2009/4–2010/2

Die TEXnische Komödie is the journal of DANTE e.V., the German-language TEX user group (`http://www.dante.de`).

DTK 2009/4

Jürgen Gilg, Exercises, tests and exams

Using the `eqexam` package one can easily typeset exercises, tests and exams to include the solutions in the text. This article introduces the environment `problem` as well as its starred version, and shows how to deal with the solutions of the exercises.

Uwe Ziegenhagen, Dynamic hiding of text and creating gap texts

There are many applications where hiding certain parts of the text can be useful: a cv is to be set with or without final grades; for test sheets, exercises with and without solutions are needed. Especially useful in school are texts with gaps where the students have to fill certain words or phrases. Using LATEX it is quite easy to fulfill theses requirements, dealing with them also provides a chance to work with the definition of new commands.

Herbert Möller, Creating LATEX figures using OpenOffice.org 3 Draw

The powerful, freely available drawing tool OpenOffice.org 3 Draw (OOoDraw) and a Perl-based filter program are used to create — even complicated — editable figures for the LATEX picture environment. The filter program `OOopict.pl` transforms the PostScript code exported by OOoDraw and directly creates LATEX code. Since the PostScript code is the same on all platforms `OOopict.pl` can be used with any operating system. Particularly useful is the fact that the curves generated by OOoDraw are implemented as cubic Bezier splines in the PostScript code since the `pict2e` package (which is supported by `OOopict.pl`) can represent these curves in a very efficient way.

Herbert Möller, From pageref to `\hyperpage`

The jump targets created from links defined by `\pageref` are often faulty when the `hyperref` package is used with pdfTEX. This article shows how the problem and systematic corrections while creating an index can be solved by Perl filter programs.

Adelheid Grob, Typesetting crossword puzzles with LATEX

There are two packages available for typesetting crossword puzzles with LATEX: The older `crossword` package by B. Hamilton Kelly from 1996, the newer `cwpuzzle` by Gerd Neugebauer, its latest version from 2009. While the first only allows typesetting of quadratic puzzles, the latter allows all kinds of crossword puzzles and even Sudoku or Kakuro riddles. Both packages allow the output of the solution as well. This article introduces both packages and their features.

Alexander Willand, LATEX going business: Law firms

LATEX was designed for science. Is it possible to employ LATEX in companies doing real business? To what extent does LATEX meet the requirements for software to write commercial texts? What will be better compared to Word & Co., and where are the difficulties?

DTK 2010/1

EuroTEX 2009 proceedings, a.k.a. *TUGboat* 30:3.

DTK 2010/2

Christian Justen, Hebrew typesetting for theologians

This article offers a brief overview of the peculiarities in Hebrew typesetting and tries to show a LATEX approach.

Wilfried Ehrenfeld, The LATEX template for IWH green papers

The article discusses the LATEX template for German and English green papers used by the Institute for Economic Research in Halle. Requirements, implementation, and first experience with the template are described.

Arno Trautmann, Dennis Heidsiek, Christian Kluge, and Stefan Mayer, Neo & XHLATEX — Ergonomics and variety of characters

Since the spread of XHTEX and the advancements of LuaTEX, the TEX world has certainly arrived at Unicode text encoding and modern font technologies. The main input device of most users, the keyboard, still remains essentially a mechanical typewriter. This article presents the modern Neo keyboard layout, which intends permitting up to date work with text processing and possibly eases the use of LATEX.

Christian Faulhammer, Commercial LATEX: Canoe Rental

Even with outdoor activities TEX can provide good service. Reading on the balcony in summer is not intended, but rather its application in canoe rental. Here the wide range and flexibility is demonstrated in various possible scenarios.

[Received from Herbert Voß.]

The Asian Journal of TEX, Volume 3 (2009)

The Asian Journal of TEX is the publication of the Korean TEX Society (`http://ktug.kr`).

AJT Volume 3, Number 1

Kangsoo Kim, Hangul TEX: Past, present, and future; pp. 1–26

This article looks back upon the past and the current status of Hangul TEX system, and tries to give a view on the future of Hangul TEX. Specifically, we will look into a set of required features of the Hangul TEX system by describing the tasks that ko.TEX has faced and tackled. Our focus will be on the issues regarding implementing proper Hangul typography as well as basic typesetting of Hangul characters.

Kihwang Lee, Installing TEX Live 2008 and ko.TEX under Ubuntu Linux; pp. 27–40

This article provides practical guides for installing TEX Live 2008 and ko.TEX under Ubuntu Linux, a popular Linux distribution. We also look into issues regarding installing other TEX-related tools including Kile and LyX, and additional TrueType fonts.

Eung-Shin Lee, Practical presentations using TEX; pp. 41–50

To achieve effective communication of ideas and thoughts, it is vital to choose appropriate tools and medium. This article offers some general principles for better presentations. It also introduces the beamer class, a LaTeX macro packages for creating beautiful and effective presentation materials. We will concentrate on the key features of beamer that distinguish it from other presentation tools.

Juho Lee, Applications of TEX in the publishing world; pp. 51–79

In this paper, we look into the definition and the components of a book which is the final product of publishing. We also introduce the roles that TEX can play in the various stages of producing a book, and describe the strengths and weaknesses of TEX as a typesetting system compared to other systems. The methods of implementing essential typographical elements including book size, page layout, font selection, line and character spacing settings, and paragraph justification are also shown together with practical examples.

Hans Hagen and Taco Hoekwater, Halfway, the LuaTEX Project; pp. 81–87

Published in *TUGboat* 30:2.

AJT Volume 3, Number 2

Shinsaku Fujita, Articles, books, and Internet documents with structural formulas drawn by XyMTEX — Writing, submission, publication, and Internet communication in chemistry; pp. 89–108

Preparation methods of chemical documents containing chemical structural formulas are surveyed, referring to the author's experiences of publishing books, emphasizing differences before and after the adoption of (La)TEX-typesetting as well as before and after the development of XyMTEX. The recognition of XyMTEX commands as linear notation has led to the concept of the XyM notation, which has further grown into XyMML (XyM Markup Language) as a markup language for characterizing chemical structural formulas. XML (Extensible Markup Language) documents with XyMML are converted into HTML (Hypertext Markup Language) documents with XyM notations, which are able to display chemical structural formulas in the Internet by means of the XyMJava system developed as a Java applet for Internet browsers. On the other hand, the same XML documents with XyMML are converted into LaTeX documents with XyMTEX commands (the same as XyM notations), which are able to print out chemical structural formulas of high quality. Functions added by the latest version (4.04) of XyMTEX have enhanced abilities of drawing complicated structures such as steroids. LaTeX documents with XyMTEX formulas can be converted into PDF (Portable Document Format) documents directly or via PostScript document. Applications of such PDF documents in online or semi-online submission to scientific journals have been discussed.

Shin-ichi Todoroki, Beyond standard slideware: Audience-oriented slide preparation using LaTeX and a scripting language; pp. 109–118

Many people start to prepare their slides before identifying their core messages, which should be extracted from what they want to talk about. Thus the resulting presentations fail to attract much attention. To avoid this mistake, I apply the "Rule of Three" to all my slides, in each of which I place certain key phrases including my three core messages. These additional editing tasks are performed semi-automatically with the aid of the programming functions of LaTeX and a scripting language. My motivation for developing this system is to acquire a sincere attitude towards my audience through *Kata*, an essential concept in the process by which traditional Japanese culture is passed on.

Yoshihisa Nagata, Overcoming limited access issues with LaTeX: Online reprints of old books; pp. 119–123

The online publishing potential of LaTeX offers a possible solution to the problem of access to old and rare books. This paper demonstrates how LaTeX could be applied to rare editions of the nineteenth century *Grimm's Fairy Tales*. Attention is drawn to the ability of LaTeX to accommodate Old German scripts, and by extension, other archaic typefaces in its font selection scheme. A khm package that I developed myself by integrating developments of Daniel Taupin, Walter Schmidt and Torsten Bronger is introduced, outlining its ease of use and range of option selections.

Satoshi Hagihira, Tool for customizing BibTeX style files; pp. 125–131

BibTeX is a powerful tool for building reference lists from a bibliography database. Because bibliography styling varies so widely among journals, a bibliography style file, capable of creating a list that exactly meets the requirements of a target journal, may not always be available. Since manually editing a BibTeX style file to ensure compatibility is troublesome and prone to error, I developed cbst, a tool that employs shell scripting and Gawk scripts to customize BibTeX style files. Using cbst, it is possible to easily generate bibliographies that conform with the style of most target journals.

Tomohiko Morioka, Typesetting of multilingual bibliography for Oriental studies using upLaTeX; pp. 133–139

This paper describes the typesetting of "Annual Bibliography of Oriental Studies" (ABOS) as a case study of multilingual typesetting. ABOS is a multilingual bibliography of oriental studies, including various languages and scripts such as Japanese, Chinese, Korean, English, French, German, Russian, other European languages, Vietnamese, Thai, Latin transcriptions of Sanskrit, Tibetan, Arabic, etc., IPA phonetic symbols, Ancient Chinese scripts such as Oracle-Bone inscriptions, Bronze inscriptions, Chu bamboo scripts and their modern transcriptions, etc. Most of the characters included in ABOS are included in UCS, however, some characters/scripts are missing, for example Oracle-Bone script, other ancient Chinese scripts and their modern transcriptions. This paper briefly describes the current typesetting system based on upLaTeX.

[Received from Jin-Hwan Cho.]

Les Cahiers GUTenberg
Contents of issues 48–53 (2006–2009)

Les Cahiers GUTenberg is the journal of GUT, the French-language TeX user group (`http://www.gutenberg.eu.org`).

Cahiers 48, 2006

Thierry Bouche and Michel Bovani, Éditorial; pp. 3–6

Denis Roegel, Sphères, grands cercles et parallèles [Spheres, great circles, and parallels]; pp. 7–22

[Translation published in *TUGboat* 30:1.]

Till Tantau, Tutoriel Ti*k*Z [Ti*k*Z tutorial]; pp. 23–92

Karl is a math and chemistry high-school teacher. He used to create the graphics in his writings using LaTeX's `{picture}` environment. While the results were acceptable, creating the graphics often turned out to be a lengthy process. His son advises him to try out another tool, named Ti*k*Z. We follow him along his rapid learning curve.

Hagen must give a talk about his favorite formalism for distributed systems: Petri nets. He discovers the power of the tools available with Ti*k*Z in order to set-up this kind of structure.

At the end of the day, both of them seem rather convinced: Ti*k*Z is quite a piece of software!

***Cahiers* 49, 2007**

Thierry Bouche, Éditorial; p. 3

Sébastien Mengin, LaTeX en édition littéraire et dans un contexte professionnel [LaTeX in the professional context of a literary edition]; pp. 5–18

This is the tale of the author's experience while implementing LaTeX as a typesetting tool at an alternative publisher's house.

Jacques André and Jean-Côme Charpentier, Lexique anglo-français du *Companion* [English-French glossary of the *Companion*]; pp. 19–45

The *LaTeX Companion, Second Edition*, has been translated into French. During editing, problems happened due, on one hand, to the fact that prepress process was done by people who were at the same time translators, composers and proof readers, and on the other hand to some difficulties in translating technical terms especially in the context of TeX. Typical examples of these problems are exhibited. Then the English to French lexicon built for this translation is given.

Christian Rossi, De la diffusion à la conservation des documents numériques [From dissemination to preservation of digital documents]; pp. 47–61

[Translation published in *TUGboat* 30:2.]

Cahiers 50, 2008

Thierry Bouche, Éditorial; pp. 3–4

Yves Soulet, Manuel de prise en main pour Ti*k*Z [Hands-on manual for Ti*k*Z]; pp. 5–87

This is a concise manual for getting aquainted with the Ti*k*Z drawing system by Till Tantau. Special attention is given to applications from the real world.

Cahiers 51, 2008

Thierry Bouche, Éditorial; pp. 3–6

Heinrich Stamerjohanns, Deyan Ginev, Catalin David, Dimitar Misev, Vladimir Zamdzhiev and Michael Kohlhase, Conversion d'articles en LaTeX vers XML avec MathML: une étude comparative [Conversion of articles in LaTeX to XML with MathML: A comparative study]; pp. 7–28

Publishing in Mathematics and theoretical areas in Computer Science and Physics has been predominantly using (La)TeX as a formatting language in the last two decades. This large corpus of born-digital material is both a boon — LaTeX is a semi-semantic format where the source often contains indications of the author's intentions — and a problem — TeX is Turing-complete and authors use this freedom to use thousands of styles and millions of user macros.

Several tools have been developed to convert (La)TeX documents to XML-based documents. Different DML projects use different tools, and the selection seems largely accidental. To put the choice of converters for DML projects onto a more solid footing and to encourage competition and feature convergence we survey the market. In this paper we investigate and compare five LaTeX-to-XML transformers along three dimensions: *a*) ergonomic factors like documentation, ease of installation, *b*) coverage, and *c*) quality of the resulting documents (in particular the MathML parts).

José Grimm, Convertir du LaTeX en HTML en passant par XML: Deux exemples d'utilisation de Tralics [From LaTeX to HTML via XML]; pp. 29–59

This paper demonstrates on two examples how a LaTeX document can be converted to HTML using an XML intermediate document. The first example is INRIA's Activity Report, for which the printed reference (the PDF version) is obtained from the XML. The second example concerns a Ph.D. thesis, whose translation to HTML was undertaken after the defence, and needed some adaptations.

Thierry Bouche, Production de métadonnées MathML pour des articles de recherche en mathématiques : l'expérience du CEDRAM [Producing MathML metadata for mathematical research articles: The CEDRAM experience]; pp. 61–76

We describe CEDRICS, a general purpose system for automated journal production entirely based on a LaTeX input format. We show how the very basic ideas that initiated the whole effort turned into an efficient system because of the ability of LaTeX markup to parametrise simultaneously, and without compromising high typographical quality, for the PDF output as well as accurate XML metadata with (presentation) MathML formulas. This was made possible by the availability of two entirely independent LaTeX source processors each with its own specific focus but with full TeX-macro language support: pdfLaTeX by Hàn Thế Thành, and Tralics by José Grimm.

Jean-Michel Hufflen, Passer de LaTeX à XSL-FO [Introducing LaTeX users to XSL-FO]; pp. 77–99

[Published in *TUGboat* 29:1.]

Cahiers 52–53, 2009

Thierry Bouche, Éditorial; pp. 3–4

Yves Soulet, METAPOST raconté aux piétons [METAPOST for pedestrians]; pp. 5–117

This is a manual for getting started with the powerful graphic language METAPOST. It is written in a most accessible manner for those not so familiar with computer programming. It comes with a load of exercises and illustrations, which are each carefully explained. The example files are available for download on the *Cahiers*' website.

Eutypon 22–23, October 2009

Eutypon is the journal of the Greek TEX Friends (`http://www.eutypon.gr`).

Natasha Raissaki, Sans serif and serif Greek fonts; pp. 1–10

Sans serif and serif fonts differ in the width of the stroke in the same character: in the second the stroke is almost uniform throughout the letter, while in the first ones the stroke changes imitating the pen. The Greek small letters do not have serifs like the Latin ones; still many fonts have come out in the last years with that erroneous design. In this article, the Greek fonts are presented from a historic perspective, starting from the first sans serif designs of the Renaissance years up to some more recent efforts to correct the design mistakes of the photocomposition era. (*Article in Greek with English abstract.*)

Alexey Kryukov, Old Standard — a free font family for scholars and classicists; pp. 11–23

This paper takes its origin from the documentation accompanying version 2.0 of the Old Standard font family. However it goes into deeper details describing various problems that someone faces when reconstructing a historical typeface and extending it into a multilingual font family. In particular, the main character metrics (such as x-height and the length of ascenders and descenders) should be harmonized across all supported alphabets (e.g., Latin and Greek). The paper also concentrates on some aspects of OpenType programming which should be taken into account when preparing a font intended for Greek scholars. (*Article in English.*)

Dimitrios Filippou, Searching for the first Greek publications in the New World; pp. 25–43

The first Greek books published in the New World (America and Oceania) remain mostly unknown and very little studied. From some preliminary research, it seems that the first Greek book published in America is Lucian's *Selected Dialogues*, which was printed in Philadelphia, USA, in 1789. The publication of Greek classics in North America continued at an increasing pace throughout the 19th century. At the same time, the publication of some other kinds of Greek books commenced in the United States such as a Greek primer (*Haploun Alphabētarion*), grammars, etc. Later, at the twilight of the 19th century and the dawn of the 20th century, newspapers, journals and books came out for the Greek immigrants in the New World. With regards to æsthetics, these publications were similar to Greek books published in Europe or Greece at those times. (*Article in Greek with English abstract.*)

Apostolos Syropoulos, XeLaTeX and Greek; pp. 45–56

The relationship of TEX with the Greek language started about 25 years ago, with the first Greek fonts created by Silvio Levy. Since then, this relationship has continued with various packages, the `greek` option of `babel`, Beccari's fonts, etc., to today's `xgreek` package for XeLaTeX. XeLaTeX offers unique conveniences such as the use of OpenType fonts, automatic change of languages and hyphenation rules, etc. All these advantages of XeLaTeX for the typesetting of Greek are presented briefly in this paper. (This article is an earlier version of a chapter from Apostolos' forthcoming book (in Greek), *Digital Typography with XeLaTeX.*) (*Article in Greek with English abstract.*)

[Received from Dimitrios Filippou.]

MAPS 38–39 (2009)

MAPS is the publication of NTG, the Dutch language TEX user group (`http://www.ntg.nl`).

MAPS 38 (Spring 2009)

Taco Hoekwater, Redactioneel [From the editor]; pp. 1–2
Overview.

Ulrik Vieth, Do we need a 'Cork' math font encoding?; pp. 3–11
[Published in *TUGboat* 29:3.]

Ulrik Vieth, OpenType Math Illuminated; pp. 12–21
[Published in *TUGboat* 30:1.]

Taco Hoekwater, Math in LuaTEX 0.40; pp. 22–31
The math machinery in LuaTEX has been completely overhauled in version 0.40. The handling of mathematics in LuaTEX has been extended quite a bit compared to how TEX82 (and therefore pdfTEX) handles math. First, LuaTEX adds primitives and extends some others so that Unicode input can be used easily. Second, all of TEX82's internal special values (for example for operator spacing) have been made accessible and changeable via control sequences. Third, there are extensions that make it easier to use OpenType math fonts. And finally, there are some extensions that have been proposed in the past that are now added to the engine.

Hans Hagen, Unicode math in ConTEXt; pp. 32–46
This article is complementary to Taco Hoekwater's article about the upgrade of the math subsystem in LuaTEX. In parallel (also because we needed a testbed) the math subsystem of ConTEXt has been upgraded. In this article I will describe how we deal with Unicode math using the regular Latin Modern and TEX Gyre fonts and how we were able to clean up some of the more nasty aspects of math.

Hans Hagen and Taco Hoekwater and Hartmut Henkel, LuaTEX—Halfway; pp. 47–50
[Published in *TUGboat* 30:2.]

Aditya Mahajan, TEX programming: The past, the present, and the future; pp. 51–56
This article summarizes a recent thread on the ConTEXt mailing list regarding table typesetting. To make the article interesting, I have changed the question and correspondingly modified the solutions.

Paweł Jackowski, TEX beauties and oddities; pp. 57–62
The BachoTEX 2009 conference continued the Pearls of TEX Programming open session, introduced in 2005, during which volunteers present TEX-related tricks and short macros.

Siep Kroonenberg, Doe-het-zelf presentaties [Do-it-yourself presentations]; pp. 63–65
This article shows how one can produce presentations in one's own style, without making use of dedicated presentation packagaes.

MAPS 39

EuroTEX 2009 proceedings, a.k.a. *TUGboat* 30:3.

[Received from Wybo Dekker.]

Baskerville 10.2, October 2009

Baskerville is the journal of the UK-TEX Users' Group (`http://uk.tug.org`).

Jonathan Webley, Editorial; Events; News; pp. 2–3
Editor's introduction; announcement of TUG 2010; an update of the TEX FAQ, and note on the MathTran web site.

Jonathan Webley, The Hound; p. 3
A "somewhat easy", cryptic crossword (solution later in the issue).

Joseph Wright, siunitx: A LATEX Swiss army knife for units; pp. 3–6
Reprinted in this issue of *TUGboat*.

Joseph Wright, LATEX for chemists: filling in the gaps; pp. 6–9
New and existing packages for chemists.

Jonathan Webley, An introduction to the Greek alphabet; pp. 9–12
Background of and typesetting with the Greek alphabet.

[Received from Jonathan Webley.]

***Zpravodaj* 16(1), 2006, 19(3)–19(4), 2009**

Editor's note: *Zpravodaj* is the journal of CSTUG, the TEX user group oriented mainly but not entirely to the Czech and Slovak languages (`http://www.cstug.cz`).

***Zpravodaj* 16(1), 2006**

Jaromír Kuben, Úvodníček [Opening letter from the CSTUG president]; p. 1

Ondřej Jakubčík, Sazba chemických vzorců v TEXu [Typesetting chemical formulas in TEX]; pp. 2–10

The typesetting program TEX is used worldwide for formatting various kinds of documents. In this article, a short presentation of its abilities for typesetting various chemical structures is given.

Petr Březina, Hrstka tipů pro TEXovskou sazbu [Some tips and tricks for TEX typesetting]; pp. 10–25

The article presents an original solution of three challenging typographic tasks by means of TEX's macro language. The reader can find here not only a detailed description of the sophisticated macros but also a number of recommendations concerning typography.

1. In Czech typography, the interval dash cannot stand at the end of a line nor can it be moved to the beginning of the following line. If the split between lines is inevitable, the dash is replaced by a word. While writing a macro for the interval dash, it is necessary to overcome a certain disadvantage of the command `\discretionary`, namely that it does not allow appending glue to the current list, as the word which replaces the dash has to be separated from the text by a space, while the dash is not separated.

2. The setting of the whole first line of a paragraph in capitals (or possibly in a different font) can be applied at the beginning of individual chapters.

3. For hanging hyphenation, *The TEXbook* recommends using a special font with a zero-width `\hyphenchar`. The solution to this typographic task can, however, be achieved using an ordinary font with real-width `\hyphenchar`.

The article is also available on the author's home page, `http://www.volny.cz/petr-brezina/`.

Petr Olšák, Makro pro konverzi textů, PDF záložky [Macro for conversion of texts, PDF outlines]; pp. 26–32

A TEX macro programmer sometimes needs to convert a string of tokens to another string of tokens according to defined rules. For example, we may need to remove accents from a Czech text or to recode this text to another encoding or to transform some special characters to something else or ... For this purpose, the `cnv.tex` macro was designed. The last but not least motivation was the problem of my TEX colleague Jaromír Kuben: he needed to convert Czech text to PDF outlines (conversion to Unicode) using `hyperref.sty` but without activating Czech characters by `inputenc.sty`.

This article presents the `cnv.tex` macro for string conversion by a user defined table. The conversion process is outside of normal expansion and each token (independent of category code) can be converted to a single token or a group of tokens. The macro is available, with documentation, at `ftp://math.feld.cvut.cz/pub/olsak/makra/` in the files `cnv.tex`, `cnv-pu.tex`, and `cnv-word.tex`.

Ondřej Jakubčík, TrueType fonty, TEX a čeština [TrueType fonts, TEX and Czech language]; pp. 33–40

TrueType and OpenType fonts have several advantages compared to the well-known Type 1 fonts, which have been used for years by TEX users. In this article a way to use TrueType fonts with TEX for typesetting in Czech and Slovak language is presented. While instructions are specific for encoding XL2 and XT2 (which are compatible with ISO 8859-2), the encoding vectors can be easily modified to support other encodings, such as Cork.

Zdeněk Wagner, Zpracování pomocných TEXových souborů pomocí XSLT 2.0 [Processing auxiliary TEX files with XSLT 2.0]; pp. 40–53

The article shows possibilities for processing auxiliary TEX files with XSLT 2.0. The idea is demonstrated with a reimplementation of MakeIndex in XSLT. Some thoughts concerning the possibilities of reimplementation of BibTEX purely in XSLT are also presented.

Schválené grantové projekty [Grant projects accepted by CSTUG]; pp. 53–56

***Zpravodaj* 19(3), 2009**

Pavel Stříž, Pozdravy z TEXperience 2009 [Greetings from TEXperience 2009]; p. 117

Subjekty dlouhodobě podporující Zpravodaj [Parties supporting the journal on a long-term basis]; p. 118

Program TEXperience 2009 [The scientific and social programmes of the TEXperience 2009 conference]; pp. 119–120

Miroslava Dolejšová, Malé povídání o Valašsku [About the venue of the conference: Southern Moravia, Wallachia]; pp. 121–123

Jan Šustek, Sazba odstavců do textových oblastí [Typesetting paragraphs into text regions]; pp. 124–137

When typesetting paragraphs of text into text regions some problems arise in cases where the regions have different widths and rubber vertical spaces are used. One possible solution is described. The source code is available from `http://www1.osu.cz/~sustek/TeX/oblasti3.tex`.

Tomáš Kotouč & Martin Kvoch, TEX v informačním systému studijní agendy [TEX used and implemented in the university student agency information systems]; pp. 138–143

The article summarises the use of TEX in the information system of the university student agency (IS/STAG) developed over 15 years at the University of West Bohemia in Plzeň, the Czech Republic. TEX is mainly used to prepare all major official university documents in cooperation with Oracle databases.

A user can use TEX and pre-defined commands directly. The additional tool, FormsEd editor, allows the user to visually select mathematical and other special symbols, such as letters from Greek and Cyrillic alphabets. The style-sheets are customisable at three levels.

Educators from Plzeň deliver the basics of typography and LATEXing with regular training in IS/STAG for secretaries of departments who are usually non-typographers and non-TEXists.

The official website of the IS/STAG is `http://www.stag.zcu.cz`.

Jiří Rybička, Podpora přípravy závěrečných prací [Support for a thesis preparation]; pp. 144–159

Support for a formal submission of a final thesis for the Faculty of Business and Economy, Mendel University of Agriculture and Forestry in Brno, contains different components — courses, texts and technical elements (packages and templates). This article describes the `dipp.sty` package for final thesis preparation. The `dipp` package is supported in the TEXonWeb system (`http://tex.mendelu.cz`) and offered for any user.

Pavel Stříž & Michal Polášek, Šablony pro vysokoškolské kvalifikační práce [Templates at Tomas Bata University in Zlín]; pp. 160–172

This article briefly introduces the concept and effectiveness of creating style sheets. It gives an overview of links with several major templates over the world and templates for thesis used over the world with an emphasis to those found on web sites of the universities in the Czech Republic and Slovakia. The templates are available from `http://web.utb.cz/?id=0_0_12_3&lang=cs` under "Směrnice rektora č. 12/2009" [The Rector's Regulation No. 12/2009].

The second part of the article focuses on style sheets for Microsoft Word, OpenOffice.org Writer and LATEX created at Tomas Bata University in Zlín. The full source code of the LATEX style sheet is also presented as an appendix of the article.

Michal Mádr & Pavel Stříž, Nové a staronové knihy [New and older books]; pp. 173–180

Frank Mittelbach, et al.: *The LATEX Companion*, `http://www.informit.com`.

Michel Goossens, et al.: *The LATEX Web Companion: Integrating TEX, HTML, and XML*, `http://www.informit.com`.

Apostolos Syropoulos, et al.: *Digital Typography Using LATEX*, `http://ocean1.ee.duth.gr/LaTeXBook`.

Robert Bringhurst: *The Elements of Typographic Style*, `http://typebooks.org/r-elements.htm`; see also `http://webtypography.net`.

Florian Coulmas: *The Blackwell Encyclopedia of Writing Systems*, `http://blackwellreference.com`.

Peter T. Daniels, William Bright (editors): *The World's Writing Systems*, `http://ukcatalogue.oup.com`, cited in `http://www.omniglot.com`.

Zdeněk Wagner: *Chromozon 46,YB* (in Czech), `http://icebearsoft.euweb.cz/chromozom46yb`. Please meet an adventure book written in TEX. Congratulations to Zdeněk who holds the title "The winning book of the TEXperience 2009 conference".

Zpravodaj 19(4), 2009

Pavel Stříž, Šťastné a veselé . . . [Merry Christmas and Happy New Year 2010 from the Editorial Board]; pp. 181–183

Dave Walden & Hàn Thế Thành, Rozhovor s Hàn Thế Thànhem, tvůrcem a správcem pdfTEXu [Interview with Hàn Thế Thành, the creator and maintainer of pdfTEX, Czech version; translation and corrections by Michal Mádr and Pavel Stříž]; pp. 184–190

Translation of an interview conducted by Dave Walden with Hàn Thế Thành, the creator and maintainer of the pdfTEX program. Topics included the origins of the program pdfTEX, its early development, introduction to the wider TEX community and the organization of current development and maintenance. The original English interview can be found at

http://tug.org/interviews/interview-files/han-the-thanh.html.

Michal Mádr & Pavel Stříž, Představení LuaTEXu [Introduction to LuaTEX]; pp. 191–200

This article introduces the program LuaTEX — an ambitious successor of pdfTEX. The article discusses LuaTEX's position among other TEX follow-ups and new features of this system. Its official web site is http://www.luatex.org.

Jan Šustek, Zašifrování zdrojového textu při zachování jeho funkčnosti [On Enciphering a Source Code and Preserving Its Functionality]; pp. 201–211

It is possible to write a document in many different ways using TEX. Similarly, one can write a macro package in many different ways. Some ways are readable and some are not. If we want another user to use our macros, we have to send him the source code of those macros. But if we don't want him to see the source code, we have to encipher it and send only the ciphertext to the user.

This paper shows one way to achieve this goal. The original file is enciphered using the \uppercase primitive. The first line of the ciphered file contains macros for deciphering the other lines. This ensures that the ciphered file has the same functionality as the original file. The source code is available from the author's website at http://www1.osu.cz/~sustek/TeX/zasifruj.tex.

Luís Nobre Gonçalves, Michal Mádr & Pavel Stříž, Drawing the Shree Yantra Core with METAPOST [Vykreslení středové části obrazce Shri Yantra pomocí METAPOSTu]; pp. 212–221

The Shree Yantra Core is a figure composed of nine interlocking isosceles triangles, all inside a circle. This figure is found in some of the oldest Hindu temples and contains, in itself, a series of interesting mathematical problems. This article describes an attempt to solve the problem of drawing the Shree Yantra Core with METAPOST. The sources can be downloaded from http://matagalatlante.org/nobre/MetaPost/PlainMetaPostProgs/, in the files sriyantra.mp and sriyantraquest.mp.

André Simon, The Highlight programme: Code & syntax highlighting [Program Highlight a jeho užití]; pp. 222–239

The article presents features and options of the Highlight programme which is capable of highlighting source codes of different programming languages. The experimental approach of nested syntax configuration is tested on files with HTML+CSS+PHP+JavaScript, Perl+TEX, LATEX+Sweave+R and Lua+TEX. The official web site is http://andre-simon.de.

Jiří Demel, Pár postřehů z TEXperience 2009 [Impressions from the TEXperience 2009 conference]; pp. 240–241

PF 2010! from CSTUG and the Editorial Board [Přání do nového roku od výboru a redakce]; p. 241

Michal Mádr & Pavel Stříž, Nové a staronové knihy [New and older books]; pp. 242–244

Karl Berry, David Walden (editors): *TEX People: Interviews from the World of TEX*, http://tug.org/store/texpeople.

Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes: "The Evolution of Lua" (an article), http://www.tecgraf.puc-rio.br/~lhf/ftp/doc/hopl.pdf.

Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes: *Lua 5.1 Reference Manual*, http://www.lua.org/manual/5.1.

Roberto Ierusalimschy: *Programming in Lua*, http://www.inf.puc-rio.br/~roberto/pil2.

Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes: Lua Programming Gems, http://www.lua.org/gems.

Zdeněk Wagner: *Sedmero dračích srdcí*, in Czech. Please meet a book with fairy tales written in TEX. The book was presented for the first time at the TEXperience 2008 conference. http://icebearsoft.euweb.cz/sedmero.dracich.srdci .

Pozvánka na TUG 2010: TEX slaví 2^5 [Invitation to the TUG 2010 conference, Czech version; translated by Michal Mádr]; the back cover of the issue.

[Received from Pavel Stříž.]

TUG financial statements for 2009

David Walden, TUG treasurer

The financial statements for 2009 have been reviewed by the TUG board but have not been audited. They may change slightly when the final 2009 tax return is filed. As a US tax-exempt organization, TUG's annual information returns are publicly available on our web site: `http://www.tug.org/tax-exempt`.

Revenue (income) highlights

Membership dues revenue were down from 2008 to 2009 (at the end of December 2009 we had 1,522 paid members); conference income was substantially up; and interest income was substantially down. Product sales income was down; and contributions income was up about $2,200. Altogether, revenue increased 3 percent (about $4,000) from 2008 to 2009.

Cost of Goods Sold and Expenses highlights, and the bottom line

Payroll, office expenses, and *TUGboat* production and mailing continue to be the major expense items.

Because Cost of Goods Sold and Expenses did not change very materially from 2008 to 2009 and revenue increased, year-to-year gross loss was down about $3,500.

Often we have a prior year adjustment that takes place early in the year to compensate for something that had to be estimated at the time the books were closed at year end; in 2009 we had two such adjustments, for a total of minus $175.

Balance sheet highlights

TUG's end-of-year asset level is up from 2008 to 2009 partly because the last 2009 issue of TUGboat was billed but not yet paid.

The 'Committed Funds' come to TUG specifically for designated projects: the LaTeX project, the TeX development fund, and so forth. They have been allocated accordingly and are disbursed as the projects progress. TUG charges no overhead for administering these funds.

'Prepaid Member Income' is member dues that were paid in 2009 for 2010 and beyond. Most of this liability (the 2010 portion) was converted to 'Membership Dues' for 2010 in January 2010.

The payroll liabilities are for 2009 state and federal taxes due January 15, 2010.

Because of the 2009 loss of $3,829, the Total Equity is down essentially the same amount from 2008 to 2009.

Summary

TUG remained financially solid as we entered 2009. However, we did have to increase the fee rates (after holding them steady for three years) to cover slowly inflating expenses. Hopefully the fee increase will not cause too many people to drop their TUG membership.

TUG continues to work closely with the other TeX user groups and ad hoc committees on many activities to benefit the TeX community.

TUG 12/31/2009 (versus 2008) Balance Sheet

	Dec 31, 09	Dec 31, 08
ASSETS		
Current Assets		
Total Checking/Savings	181,596	162,709
Accounts Receivable	55	95
Other Current Assets	2,029	1,314
Total Current Assets	183,680	164,119
Fixed Assets	1,068	2,396
TOTAL ASSETS	**184,748**	**166,515**
LIABILITIES & EQUITY		
Liabilities		
Accounts Payable	11,000	0
Committed Funds	43,417	33,569
Deferred conference income	830	0
Prepaid member income	3,305	2,905
Payroll Liabilities	1,079	1,096
Total Current Liabilities	59,631	37,570
Total Liabilities	59,631	37,570
Equity		
Unrestricted	128,945	136,230
Net Income	-3,828	-7,285
Total Equity	125,117	128,945
TOTAL LIABILITIES & EQUITY	**184,748**	**166,515**

TUG 2009 (versus 2008) Revenue and Expenses

	Jan - Dec 09	Jan - Dec 08
Ordinary Income/Expense		
Income		
Membership Dues	98,815	103,171
Product Sales	5,095	5,809
Contributions Income	9,253	6,987
Annual Conference	7,640	-1,339
Interest Income	3,163	5,341
Advertising Income	315	405
Total Income	124,281	120,374
Cost of Goods Sold		
TUGboat Prod/Mailing	31,045	31,401
Software Production/Mailing	4,112	3,911
Postage/Delivery - Members	2,331	3,164
Conf Expense, office + overhead	1,840	1,036
JMM supplies/shipping	370	829
Member Renewal	434	408
Copy/Printing for members	234	30
Total COGS	40,366	40,779
Gross Profit	83,915	79,595
Expense		
Contributions made by TUG	5,000	10,525
Office Overhead	16,560	12,595
Payroll Exp	64,451	62,200
Professional Fees	230	230
Depreciation Expense	1,328	1,330
Total Expense	87,569	86,880
Net Ordinary Income	-3,654	-7,285
Other Income/Expense		
Other Income		
Prior year adjust	-175	
Total Other Income	-175	
Net Other Income	-175	
Net Income	**-3,829**	**-7,285**

TUG Institutional Members

American Mathematical Society,
Providence, *Rhode Island*

Aware Software, Inc.,
Midland Park, *New Jersey*

Banca d'Italia,
Roma, *Italy*

Center for Computing Sciences,
Bowie, *Maryland*

Certicom Corp.,
Mississauga, *Ontario*, *Canada*

CSTUG, *Praha*, *Czech Republic*

Florida State University,
School of Computational Science
and Information Technology,
Tallahassee, *Florida*

IBM Corporation,
T J Watson Research Center,
Yorktown, *New York*

Institute for Defense Analyses,
Center for Communications
Research, *Princeton*, *New Jersey*

Konica Minolta Systems Lab Inc,
Boulder, *Colorado*

MacKichan Software, Inc.,
Washington/New Mexico, *USA*

Marquette University,
Department of Mathematics,
Statistics and Computer Science,
Milwaukee, *Wisconsin*

Masaryk University,
Faculty of Informatics,
Brno, *Czech Republic*

MOSEK ApS,
Copenhagen, *Denmark*

New York University,
Academic Computing Facility,
New York, *New York*

Springer-Verlag Heidelberg,
Heidelberg, *Germany*

Stanford University,
Computer Science Department,
Stanford, *California*

Stockholm University,
Department of Mathematics,
Stockholm, *Sweden*

University College, Cork,
Computer Centre,
Cork, *Ireland*

University of Delaware,
Computing and Network Services,
Newark, *Delaware*

Université Laval,
Ste-Foy, *Québec*, *Canada*

University of Oslo,
Institute of Informatics,
Blindern, *Oslo*, *Norway*

Calendar

2010

Apr 30 – May 4 BachoTEX 2010: 18th BachoTEX Conference, "Typographers and programmers: mutual inspirations", Bachotek, Poland. For information, visit www.gust.org.pl/bachotex/2010

May 3 – Jun 25 "Marking Time": A traveling juried exhibition of books by members of the Guild of Book Workers. Public Library of Cincinnati and Hamilton County, Ohio. Sites and dates are listed at palimpsest.stanford.edu/byorg/gbw

May 26 NTG 45th meeting, Dordrecht, Netherlands. www.ntg.nl/bijeen/bijeen45.html

May 27 – 28 "DIY Design", Ninth annual St Bride Library conference, London, England. stbride.org/events

Jun 15 – 19 The 4th International Conference on Typography and Visual Communication (ICTVC), "Lending Grace to Language", University of Nicosia, Cyprus. www.ictvc.org

---

TUG 2010
San Francisco, California.

Jun 28 – 30 The 31st annual meeting of the TEX Users Group — TEX's 2^5 Anniversary. tug.org/tug2010

Jun 29 – Jul 1 75 Years of Penguin Books: An International Multidisciplinary Conference, University of Bristol, Bristol, UK. www.bristol.ac.uk/penguinarchiveproject

Jul 5 – Aug 26 "Marking Time": A traveling juried exhibition of books by members of the Guild of Book Workers. Lafayette College, Easton, Pennsylvania. Sites and dates are listed at palimpsest.stanford.edu/byorg/gbw

Jul 7 – 10 Digital Humanities 2010, Alliance of Digital Humanities Organizations, King's College, London, UK. www.cch.kcl.ac.uk/dh2010

Jul 25 – 29 SIGGRAPH 2010, Los Angeles, California. www.siggraph.org/s2010

Aug 2 – 6 Balisage: The Markup Conference, Montréal, Québec. www.balisage.net

Aug 17 – 22 TypeCon 2010: "Babel", Los Angeles, California. www.typecon.com

Aug 17 – 20 SHARP 2010, "Book Culture from Below", Society for the History of Authorship, Reading & Publishing, Helsinki, Finland. www.helsinki.fi/sharp2010

Aug 25 – 29 EuroTEX 2010, Pisa, Italy. www.guit.sssup.it/eurotex2010

Sep 5 – Oct 25 "Marking Time": A traveling juried exhibition of books by members of the Guild of Book Workers. Dartmouth College, Hanover, New Hampshire. Sites and dates are listed at palimpsest.stanford.edu/byorg/gbw

Sep 8 – 12 Association Typographique Internationale (ATypI) annual conference, Dublin, Ireland. www.atypi.org

Sep 13 – 18 Fourth International ConTEXt User Meeting, "ConTEXt typesetting documentation, teach as we preach", Brejlov (Prague), Czech Republic. meeting.contextgarden.net

Sep 24 – 26 DANTE Herbsttagung and 43rd meeting, Trier, Germany. www.dante.de/events/mv43.html

Oct 2 – 3 Oak Knoll Fest XVI, New Castle, Delaware. www.oakknoll.com/fest

Nov 6 – 8 The Eighth International Conference on the Book, University of St. Gallen, St. Gallen, Switzerland. booksandpublishing.com/conference-2010

Status as of 1 May 2010

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

A combined calendar for all user groups is online at texcalendar.dante.de.

Other calendars of typographic interest are linked from tug.org/calendar.html.

TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `http://tug.org/consultants.html`. If you'd like to be listed, please see that web page.

Dangerous Curve
PO Box 532281
Los Angeles, CA 90053
+1 213-617-8483
Email: `typesetting (at) dangerouscurve.org`
Web: `http://dangerouscurve.org/tex.html`
We are your macro specialists for TEX or LATEX fine typography specs beyond those of the average LATEX macro package. If you use XETEX, we are your microtypography specialists. We take special care to typeset mathematics well.

Not that picky? We also handle most of your typical TEX and LATEX typesetting needs.

We have been typesetting in the commercial and academic worlds since 1979.

Our team includes Masters-level computer scientists, journeyman typographers, graphic designers, letterform/font designers, artists, and a co-author of a TEX book.

Martinez, Mercè Aicart
Tarragona 102 4^o 2^a
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) ono.com`
Web: `http://www.edilatex.com`
We provide, at reasonable low cost, TEX and LATEX typesetting services to authors or publishers worldwide. We have been in business since the beginning of 1990. For more information visit our web site.

Peter, Steve
New Jersey, USA
+1 732 287-5392
Email: `speter (at) mac.com`
Specializing in foreign language, linguistic, and technical typesetting using TEX, LATEX, and ConTEXt, I have typeset books for Pragmatic Programmers, Oxford University Press, Routledge, and Kluwer, among others, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

Shanmugam, R.
No. 38/1 (New No. 65), Veerapandian Nagar, Ist St. Choolaimedu, Chennai-600094, Tamilnadu, India
+91 9841061058
Email: `rshanmugam92 (at) yahoo.com`
As a Consultant, I provide consultation, training, and full service support to individuals, authors, typesetters, publishers, organizations, institutions, etc. I support leading BPO/KPO/ITES/Publishing companies in implementing latest technologies with high level automation in the field of Typesetting/Prepress, ePublishing, XML2PAGE, WEBTechnology, DataConversion, Digitization, Cross-media publishing, etc., with highly competitive prices.
I provide consultation in building business models & technology to develop your customer base and community, streamlining processes in getting ROI on our workflow, New business opportunities through improved workflow, Developing eMarketing/E-Business Strategy, etc. I have been in the field BPO/KPO/ITES, Typesetting, and ePublishing for 16 years, handled various projects. I am a software consultant with Master's Degree. I have sound knowledge in TEX, LATEX2ε, XMLTEX, Quark, InDesign, XML, MathML, DTD, XSLT, XSL-FO, Schema, ebooks, OeB, etc.

Sievers, Martin
Im Treff 8, 54296 Trier, Germany
+49 651 49 651 4936567-0
Email: `info (at) schoenerpublizieren.com`
Web: `http://www.schoenerpublizieren.com`
As a mathematician with ten years of typesetting experience I offer TEX and LATEX services and consulting for the whole academic sector (individuals, universities, publishers) and everybody looking for a high-quality output of his documents.

From setting up entire book projects to last-minute help, from creating individual templates, packages and citation styles (BIBTEX, biblatex) to typesetting your math, tables or graphics — just contact me with information on your project.

Veytsman, Boris
46871 Antioch Pl.
Sterling, VA 20164
+1 703 915-2406
Email: `borisv (at) lk.net`
Web: `http://www.borisv.lk.net`
TEX and LATEX consulting, training and seminars. Integration with databases, automated document preparation, custom LATEX packages, conversions and much more. I have about fifteen years of experience in TEX and twenty-eight years of experience in teaching & training. I have authored several packages on CTAN, published papers in TEX related journals, and conducted several workshops on TEX and related subjects.