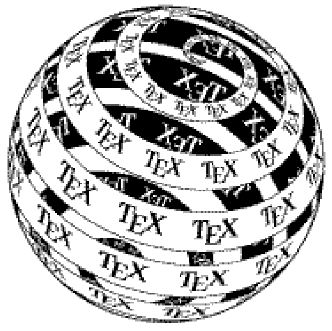
TUG'98: "Integrating TEX with the surrounding world"

Uniwersytet Mikołaja Kopernika, Toruń, Poland
17–21 August, 1998



The 1998 TEX Users Group Conference will be organised and hosted by the Polish TEX Users Group GUST. Planning is now well under way, and enquiries may be addressed to `tug-98@mail.tug.org`.

Call for Papers

Proposals for papers are now being solicited: preference will be given to papers which deal most directly with the theme of the conference, but papers on related topics (e.g., DSSSL,HTML, PDF, SGML, XML, etc.) are not excluded. Proposals (which should ideally be written in English, but which may be written in another language by prior arrangement) should be sent to the TUG'98 Programme Committee (`tug-98-papers@mail.tug.org`).

Each proposal should include the title, name, address, e-mail address and affiliation (where appropriate) of the proposer, together with an extended abstract (the latter should correspond to approximately one side of A4. In addition, there should be an estimate of the necessary time for verbal presentation (excluding questions: a further five minutes will be allowed for questions after each talk), and a further estimate of the number of printed pages which will be required to reproduce the full text of the article using the `[l]tugproc` macros. Any special needs to process the paper should also be stated (for example, colour pages in the preprints/proceedings; e-TEX, pdfTEX, Omega, etc.).

Deadlines

30 January 1998:
Deadline for receipt of proposals

20 February 1998:
Deadline for notification of acceptance/rejection

10 April 1998:
Deadline for receipt of first drafts

29 May 1998:
Deadline for receipt of final versions

For more extensive information, consult the TUG'98 website at `http://www.tug.org/tug-98/`, which also includes a list of related conferences.

Getting there

Delegates arriving from overseas will probably fly into Warszawa ("Warsaw") Airport: the schedule for LOT, the national carrier, is on-line at `http://www.lot.com/schedule/`. Many other carriers (e.g., British Airways, KLM, Sabena) also fly into Warszawa.

From the airport, delegates should take the Airport City bus to Warszawa Centralna ("Warsaw Central") station: this bus journey costs 4 zl (the current exchange rate is about 5 zl to 1 pound sterling). The number 175 bus is a cheaper option (cost: 1.6 zl), but not advisable if traveling with large quantities of luggage, exposed wallets, documents, etc. ... Taxis can prove very expensive unless booked by telephone in advance from a reliable company such as Wawa or Wolfra (a useful list of taxi companies with telephone numbers can be found at `http://www.inter.com.pl/warsaw/wttt.htm`.

From the station, trains run direct to Toruń; the PKP (Polish Railways) timetable is on-line at (`http://bahn.hafas.de/bin/db.w97/query.exe/en`).[1] Train travel within Poland is not expensive, and delegates should book first-class accommodation with reserved seats if possible: couchettes and sleeping cars are available on some overnight trains. Beware of pickpockets, and never allow yourself to be forced to pass between two strangers in the train corridor.

If you prefer to drive to Toruń, and are coming from outside Poland, expect considerable delays at such well-known border crossings such as Frankfurt am Oder. Far better is to seek out the little-used local crossings such as that at Kostrzyn. Formalities are minimal, and you need not complete a currency declaration unless you are carrying more than 2000 ECU. Once in Poland, pay particular attention to speed limits, which are strictly enforced (60 kph in towns/villages, 90 kph outside, 110 kph on expressways and motorways: any/all of these may be overridden by signs). The alcohol limit for drivers is zero.

[1] Specify *Warszawa Centralna* as 'From:' and *Torun Glowny* as the destination 'To:'; in fact, any European station can be the starting and/or finishing point.

Message from the President

Mimi Lafrenz Jett
ETP Harrison, 1466 NW Front Avenue, Suite 200, Portland OR 97209-2820 USA
mimi@etp.com

Greetings TUG Members,

The 18th Annual Meeting and Conference held this summer in San Francisco proved to be an exciting week of TEX-nology and transition. The passing of responsibilities from the outgoing board and officers to the new was effected with tremendous professionalism and grace, under the leadership of Michel Goossens, our diplomatic past-President. It is with sincere appreciation for countless volunteer hours and selfless determination that we, the entire member-body, thank President Goossens; Vice-President Yannis Haralambous; Secretary Sebastian Rahtz; Treasurer Mimi Burbank; and board members Robin Fairbairns, George Greenwade, Alan Hoenig, and Jon Radel for their service to TUG and the TEX community. Only through the efforts of these individuals, and many other enthusiastic volunteers have we made progress and evolved into the forward thinking TUG of the next century.

An obvious indication of our progress is the restructuring of the office and support team. In an effort to reduce costs and improve service to our members, TUG no longer employs a staff of personnel, but instead employs the services of three contractors — with specific responsibilities and duties — eliminating the overhead expenses traditionally associated with a staff. Thanks to the efforts of Art Ogawa and his incredible wife, Marian Goldeen, with the cooperation of our outgoing Executive Director Patricia Monohon, the business of the TUG office was attended to; analyzed; and organized into the three key-result areas we now utilize as our structural base: membership support and office administration; bookkeeping; and email. With great good luck, all three positions have been filled by professionals with character and experience. Lena Mohajerin, Administrative Assistant, is an entrepreneur in the field of training seminars for international trade, when she is not busy answering membership queries or filling orders for the ever-popular TEX Live 2 CD (which, of course, is a benefit of membership). Lena keeps regular hours in the office on Tuesdays and Fridays, with supplemental work from her home office. (Watch our WWW site for scheduled telephone times and other important news from Lena.) Cindy Hansen of ABC Solutions now handles our bookkeeping, as she does for a handful of companies in the Portland area, including ETP Harrison. Justin Winkler (aka Wink) is a TEX programmer with valuable knowledge and experience in a variety of computer operating systems and languages; he reads and forwards all email to the proper support team and works closely with Art Ogawa and Karl Berry to continually improve our database and responsiveness. Lena, Cindy, and Wink are available to support our members at the addresses listed inside the front cover. Please do not hesitate to contact the office with any issues or suggestions you have. If you do not get the help you need, or feel we could have done better, please contact me directly at `president@mail.tug.org`. I will personally respond to any mail at that address, and I appreciate your feedback.

The best laid plans of mice and men, often go asunder

Relocating the physical office has also been an evolutionary process. Amid the upheaval of a change in situations, Art and Marian moved swiftly to keep us operational by moving the essential office tools to their office in Three Rivers, California. Without such heroic actions and open communication of the board and conference committee, I am afraid the meeting in San Francisco may not have been such a success, not to mention the development of our current plan of action. During the week of the conference, dozens of volunteers helped pack and move the remaining contents of our San Francisco office out of the Flood building and into temporary storage. It was quickly apparent that the term 'virtual office' is an oxymoron — there are no bookshelves in a virtual office to stack *TUGboat*s on, no desk for the computer, no cabinets for our decades' accumulation of files. So now, what to do? We had $>$ 60 boxes of important stuff, no staff, and an uncertain direction. Once the election results were finalized, it made sense to reconstitute the office in Portland, Oregon, where two of the five members of the newly formed Business Committee live. Of course, the fact that we were offered space for an office, free of charge, was a major determining factor. We are now housed in the spare room of ETP Harrison, our small publishing service business working in TeX. As long as ETP Harrison holds the lease on this building, the TUG office is guaranteed a rent-free location.

The savings we are experiencing in overhead have allowed us to purchase a new computer for membership support, including email, web server, and database management. A special thanks to Karl Berry for his constant support of our electronic communications and another enthusiastic round of applause for our friends at DANTE for their contribution of a Sparc-10 for our CTAN server.

During the chaos of two moves, new people and the annual meeting, communications and support have been less than adequate. We apologize for any concern or inconvenience this has caused our members, and promise improvement in the near term. Your requests and applications are being processed now, and you should have answers by the time this issue hits your mailbox. The financial condition is also being sorted out, and the treasurer's report is expected to be ready for the next issue. Thank you for your patience during our difficult times. The future is bright, and has finally arrived!

It is with continuing awe of the unexpected, that I sign the President's Message for this edition of our journal *TUGboat*. Although I have hoped to serve in this capacity for sometime, it did not seem like a realizable goal until this year. And my Mother used to tell me: "Be careful what you wish for, you may get it!"

Here's wishing happiness and hope for all of us.

Production Notes

We apologize for the delay of this issue. The production team members first assumed responsibility for the issue in mid December. An article by Hàn Thế Thành, will appear in the next issue of *TUGboat*, as will an article by LaTeX3 project members, David Carlisle, Chris Rowley, and Frank Mittelbach.

Output This issue was prepared using the beta version of the *TeX Live* CD-ROM, version 3 and I am pleased to report that everything runs smoothly!

The final camera copy was prepared at SCRI on a UNIX platform, running AIX v3.2.5, using the *TeX Live* setup (Version 3), which is based on the *Web2c* TeX implementation version 7.2 by Karl Berry and Olaf Weber. PostScript output at 600dpi was produced using Radical Eye Software's dvipsk 5.76a and printed on a QMS 860 printer. Because of time and software requirements, Werner Lemberg very kindly provided the final PostScript files for his two articles.

TeX Live **CD-ROM, version 3**

TUG and the UKTUG are preparing a new *TeX Live* CD-ROM, with an anticipated release date of mid-March 1998.

TeX Live 3 will include:

- updated macro and font packages;
- December '97 LaTeX;
- bugs fixed in install scripts;
- UNIX and Win32 binaries based on web2c 7.2;
- WIN95/NT system runnable directly from CD-ROM;
- Omega, e-TeX and pdfTeX for UNIX and Win32;
- latest CMacTeX, OzTeX, and MikTeX distributions;
- lacheck, dvidvi, dtl, psutils, t1utils, and dviselect/concat for all UNIX and Win32;
- full CONTEXT macro package; and
- Joliet CD-ROM support, so that win32 users see long/mixed-case filenames.

◇ Mimi Burbank
SCRI, Florida State University,
Tallahassee, FL 32306 – 4130
mimi@scri.fsu.edu

TUG'97 Program

Saturday July 26, 1997	**1:00 – 4:00 Board of Directors Meeting**
Sunday July 27, 1997	**Tutorials** **Registration 1:00 – 5:00**
1:00 – 5:00	Aspects of Omega; from everyday use to development and extension of multilingual tools / *Yannis Haralambous & John Plaice*
1:00 – 4:00	LaTeX2HTML / *Ross Moore*
4:00 – 6:00	Moving from LaTeX 2.09 to LaTeX 2ε/ *Anita Hoover*
	Welcome Reception 6:00 – 8:00 LMCC Room 100
Monday July 28, 1997	**Pictures and TeX** **8:00 a.m. – 4:00 p.m. Registration**
9:00 – 9:30	Opening Convocation /*Michel Goossens, Outgoing President*
	Picture this: the TeXxie approach to graphical illustration *Session Chair: Hans Hagen*
9:30 – 10:00	"Xy-pic as a tool for VHL2G and how this made TeX into an animation tool" / *Kristoffer H. Rose*
10:00 – 10:30	"A tutorial on MetaPost graphs" / *Sebastian Rahtz*
10:30 – 11:00	*Break*
11:00 – 11:30	"Drawing with DraTeX" / *Eitan Gurari*
11:30 – 12:00	"High-quality labels on included graphics, using Xy-pic" / Ross Moore
12:00 – 12:30	"CIRC: a package to draw flow-sheets of all types" / *Sebastian Tannert*
12:30 – 1:30	Lunch
	Tooling up: where are we with TeX? *Session Chair: Taco Hoekwater*
2:00 – 2:30	"The state of ε-TeX" / *ε-TeX member*
2:30 – 3:00	"Omega, the full release" / *John Plaice & Yannis Haralambous*
3:00 – 3:30	"TeX Live 2 – towards a fully flexible TeX on CD-ROM" / *The TeX Live Team*
3:30 – 4:00	Break
4:00 – 4:30	"New font tools for TeX" / *Werner Lemberg*
4:30 – 5:00	"Production of complicated and highly interactive documents" / *Hans Hagen*
Tuesday July 29, 1997	**The Web and SGML** **8:00 a.m. – 4:00 p.m. Registration**
8:30 – 9:45	**TUG Business Meeting**
	TeX and scientific publishing on the Internet *Session Chair: Chris Rowley*
10:00 – 10:30	"A new TeX math font family for Elsevier" / *Yannis Haralambous*
10:30 – 11:00	"DVIPDF and graphics" / *Sergey Lesenko*
11:00 – 11:30	"TeX to PDF direct" / *Han The Thanh*
11:30 – 12:00	"Developments in PDF, and LaTeX" / *Steve Zilles*
12:00 – 1:30	Lunch
1:30 – 2:00	"techexplorer: Interactive scientific electronic publishing for the Internet" / *R.S. Sutor, A.L. Diaz and S.S. Dooley*
2:00 – 2:30	"Translating SGML to HTML, with help from TeX" / *Chris Hamlin*

– – – Continued – – –

– – – TUG'97 Program *continued* **– – –**

	TEX behind the scenes: what is our relationship to SGML? *Session Chair: Michel Goossens*
2:30 – 3:00	"The DSSSL style sheet language" / *Jon Bosak*
3:00 – 3:30	"The TEX backend for Jade" /*Sebastian Rahtz*
3:30 – 4:00	Break
4:00 – 4:30	"DSSSL, TEX and math" / *Chris Rowley*
4:30 – 5:00	"LATEX2`HTML` – past, present and future" / *Ross Moore*
Wednesday July 30, 1997	**Publishing and TEX** **8:00 – 4:00 Daily Registration**
	TEX and the real world
9:00 – 9:30	"The advantages of LATEX in producing electronic courseware" / *Mimi Jett*
10:00 – 10:30	"Custom legal documents for the Auto Loan Exchange" / *Douglas Lovell*
10:30 – 11:00	"What LATEX needs to make it useful to publishers" / *Fred Bartlett*
11:00 – 12:00	Panel discussion of LATEX and publishers
12:00 – 1:30	Lunch
1:30 – 2:30	Vendor presentations
	LATEX – state of the art? *Session Chair: Mimi Jett*
2:30 – 3:00	"LATEX project overview" / *Chris Rowley*
3:00 – 3:30	Break
3:30 – 4:00	"LATEX3 Programming / *David Carlisle*
4:00 – 4:30	"Breaking equations" / *Michael Downes*
4:30 – 5:30	Panel discussion on the future development of LATEX in relation to the new TEX variants becoming available
	Hornblower Dinner Cruise 7:30 – 10:30
Thursday July 31, 1997	**LATEX, Fonts and Languages**
	Real Work
8:30 – 9:00	"Using color in TEX: an anecdotal journey" / *Dan Olson*
9:00 – 9:30	"TEX meets watermark" / *Kazuhiro Kitagawa*
10:00 – 12:00	**Birds of a Feather (BOFs)** ***concurrent***
12:00 – 1:30	Lunch
	Multilingual typography without boundaries *Session Chair: Erik Frambach*
2:30 – 3:00	"Developments in LATEX for multilingual documents" / *Frank Mittelbach*
3:00 – 3:30	"The multilingual interface of the ConTEXt macro package" / *Hans Hagen*
3:30 – 4:00	Break
4:00 – 5:00	"The CJK package: multilingual support beyond Babel" / *Werner Lemberg*
5:00	**Closing ceremonies**
Friday August 1, 1997	**9:00 – 1:00 Board of Directors Meeting**

Very High Level 2-dimensional Graphics with TEX and XY-pic

Kristoffer Høgsbro Rose
BRICS, University of Aarhus (DAIMI), Ny Munkegade building 540, 8000 Århus C, Denmark
krisrose@brics.dk
URL: http://www.brics.dk/~krisrose/

Abstract

A problem with using pictures in TEX and LATEX documents is that there is no natural universal notation encompassing all possible diagrams, flow-charts, *etc.*In this paper we argue why one should not attempt such generality but rather design *custom embedded graphic languages* for classes of similar pictures.

The main argument is the usual one for markup languages: using a specialised high-level notation means that the source captures the essential properties of the picture. Not only does this make it easier for the user, who can concentrate on contents rather than form, but it also makes it easier to abstract out inessential style issues such that the "picture style" can be varied without changing the source. The main concern is that implementing a large number of such languages is only feasible with access to a versatile and powerful *drawing library* such that the amount of hacking required for each language is minimal.

As an example we survey how one can include a small "directory tree" in a paper, and we design and implement an embedded tree drawing language useful for this purpose. We illustrate the generality of the notation by showing how the same source can be used to generate the tree and even to *grow* it (by animating it) following the structure information. We finally present the implementation in TEX using XY-pic to produce the actual graphics.

Introduction

A common reason why skilled professionals working in technical areas choose TEX (Knuth, 1988) is that TEX makes it is easy to produce high quality drafts and to introduce corrections based on comments into these, making the edit-publish-feedback loop a fast and smooth one. This is further encouraged by the principle of "logical markup" promoted by formats such as LATEX (Lamport, 1994): this makes it possible to work with manuscripts with the focus on contents rather than form. In particular the facility for mathematical typesetting based on the structure of formulae means that authors can work with notions as they think about them in manuscripts, using familiar notations and groupings, deferring fine points of the typesetting until the latest moment without compromising the quality of drafts seriously, and — more importantly — without compromising the quality of the final version at all. Often it is even possible to use the same source for typesetting and other purposes. This is a very safe way of ensuring that the formulae appearing in a technical report are, indeed, exactly the same used in computations.

However, in contrast to this pleasant situation for formulae with textual structures, the treatment of simple, illustrative *pictures* is presently seriously lacking both in convenience and in the quality of the result. Even on the Internet there is no universally useful standard for "logical" specification of illustrative diagrams.[1] The reason for this is probably that it is rather easy to just compose a "quick and dirty little picture" by throwing together some arrows, boxes, lines, *etc.*, using a small visual drawing tool. That the result is usually excessively ugly is largely ignored (that proper composition of pictures was an immensely complicated task in traditional typography is likely to play a role in this matter).

Presently the following statements are representative for the options for 2-dimensional graphics that can be used with "portable TEX source documents" such as submissions, *etc.*; the choices are listed in approximate order of frequency in the author's experience.

[1] The situation is acceptable for complicated 3-dimensional drawing, however, where several successful standards exist; one of these, VRML, is even becoming an Internet standard.

"Graphics is not portable!" A textual approximation will have to do.

"One can do everything with rules!" One can do wonderful things drawing just horizontal and vertical lines using TEX rules.

"PostScript is portable graphics!" PostScript can be used to create the picture as an encapsulated PostScript (Adobe, 1990) file distributed along with the article source.

"METAFONT is portable graphics!" Since TEX is bundled with METAFONT all TEX installations should have a METAFONT engine. So all we have to do is draw the graphics with METAFONT (Knuth, 1986) and generate a font containing the drawing on each platform.

"Use a custom notation!" Designing an embedded language targeted at expressing the desired graphics directly in the TEX source without any constraints as to how the picture is actually drawn, is the only truly portable form of picture. It is easy to provide a macro package that makes actual pictures using the style of the context as far as possible.

Below we will first survey the five options for a particular *example* before we summarise the *design principles* for custom embedded languages and show how the information contained about a drawing in a well-designed language can be used for other purposes such as *animation*.

Survey by Example

Consider the following: the staff of a computing facility writes a monthly article where tradition has it that the "current directory structure" is included. The article is distributed to a number of departemental newsletter editors that are all published with TEX (of course). The various newsletters are published using a variety of styles and fonts and printed on all sorts of equipment, so portability is a crucial issue.

Avoiding graphics. The staff can use the first choice easily, *e.g.*, through the ouput of some standard tool. The UNIX *tree* command, *e.g.*, produces output as shown in figure 1. While this solution is ugly it is certainly completely portable, and in fact used more often than not.

Using standard TEX rules. The second choice is almost as easy to realise and only requires a bit of hacking. Figure 2 shows what one can produce easily by substituting parts of the textual form with appropriate rules and spaces in a TEX `\haligns` construction. Such substitution essentially means that we translate the text directly into an appropriate TEX source representation which means that we need to know how to interpret the text in order to make a meaningful translation.

```
xy-3.4
|-- doc
|   `-- xyguide-html
|-- mfinputs
|-- pkfonts
|   |-- ljfour600
|   |-- ljfour657
|   |-- ljfour720
|   `-- ljfour864
|-- ps
|-- psfonts
|-- src
|-- texfonts
`-- texinputs
```

Figure 1: UNIX *tree* output.

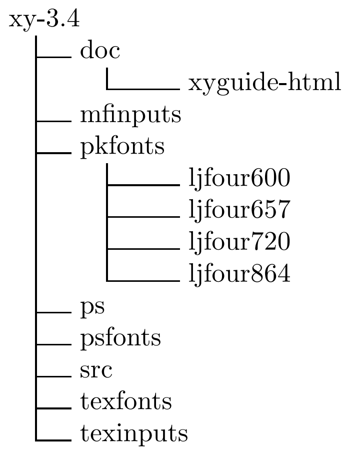


Figure 2: UNIX *tree* output with substitutions.

PostScript. The (encapsulated) PostScript choice is often the most practical, and is almost portable — only a few platforms cannot print PostScript and a few more cannot preview files with PostScript well. With PostScript the example tree might look as shown in figure 3. While the size of the figure can be changed by scaling it remains difficult to change the "Times" (or whatever) look of the figure, however, with an "embedded PostScript" package, such as the very powerful PSTricks (van Zandt, 1996), one can reduce this problem somewhat.

METAFONT. This choice is interesting in that all TEX installations are supposed to have METAFONT and thus compatibility is not a problem in principle.

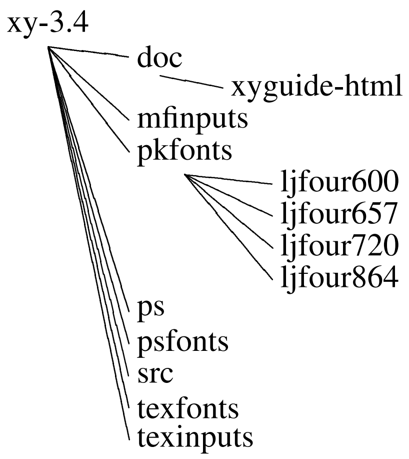


Figure 3: PostScript picture.

However, it turns out that not all TEX installations can handle dynamic changes to the repertoire of fonts gracefully, thus in practice this is less of an option than one would hope. Two alternatives exist: using META O T instead relaxes the dynamic font problem somewhat at the penalty of requiring the use of PostScript, and using the mfpic package (Leathrum and Tobin, 1994) makes it possible to mix METAFONT-generated drawing with TEX-produced text. These are options that need to be further invesitigated. In general TEX would benefit greatly from a *component model* permitting interaction between the various graphic forms — but this is beyond the scope of this presentation.

Designing a custom embedded graphic language. The fifth and last option is obviously ideal — once a dedicated language exists for the kind of graphics in question, that is. The most obvious way to think of the directory structure is as a *tree* with a node for each directory (as hinted at by the UNIX command name). This leaves us with the problem of coming up with a good textual notation for trees. One possible such notation has been used in figure 4 using parentheses to express the directory nesting structure.

In this paper we will argue that designing such very high level drawing languages for *embedding* picture descriptions directly in the manuscript is an option worthwhile pursuing in many cases, even considering the initial cost of designing and implementing the language.

An Embedded Language

We first explain the principal properties of embedded languages and how this is reflected by our toy

```
\tree{
  {xy-3.4} (
    {doc} (
      {xyguide-html} ()
    )
    {mfinputs} ()
    {pkfonts} (
      {ljfour600} ()
      {ljfour657} ()
      {ljfour720} ()
      {ljfour864} ()
    )
    {ps} ()
    {psfonts} ()
    {src} ()
    {texfonts} ()
    {texinputs} ()
  )
}
```

Figure 4: Directory as abstract tree.

directory tree sample language; we then describe the implementation of it in TEX.

Principles. There are the two principal properties that embedded languages should opt for:

Use generic abstract structures. Each custom embedded language is unique. Chances are, however, that your users will see several of your languages. Therefore try to use generic abstract structures as the "glue" of the language: this eases both the design and implementation task, and makes it easier for users to learn and later remember several embedded languages without despairing.

Use conservative notation. A custom embedded language should fit smoothly in with its "host language." Use the host language's notations whenever possible.

The "abstract structure" of a directory tree is a tree. Thus we can use standard prefix notation for trees and write each "branch" as

label (*subtree* ... *subtree*)

where *label* is the text associated to each node (for a directory this will be its name) and each *subtree* is an entire tree rooted below the present node (corresponding to subdirectories and files). Branches with no subtrees are often called "leaves" but we do not need to distinguish: a leaf can be written

label ()

This constitutes the "glue." For the nodes we should try not to extend the TEX notation too much. A nice and conservative approach is to use the TEX argument notation, *i.e.*, write the labels as

`{`*text*`}`

to indicate that *text* should be interpreted as TEX source text. In fact this is the notation we used in the directory tree example in figure 4.

Implementation. We will base our implementation on XY-pic (Rose and Moore, 1997) since this is a generic platform for 2-dimensional graphics that works with TEX and can do what we wish to illustrate without compromising the quality of the typeset drawings. However, the technique used to produce the actual graphics is not essential as long as the "library" of available graphics functions is sufficiently easy to use.

Embedded languages are implemented by writing a small *interpreter* that parses the language and performs the appropriate actions, in this case calls the appropriate XY-pic drawing primitives. In order to write such an interpreter we should write the BNF[2] of the language. This looks as follows:

$$\begin{array}{rcl} \langle\mathit{tree}\rangle & ::= & \texttt{\{}\ \langle\mathit{text}\rangle\ \texttt{\}}\quad \texttt{(}\quad \langle\mathit{subtrees}\rangle\quad \texttt{)} \\ \langle\mathit{subtrees}\rangle & ::= & \langle\mathit{empty}\rangle \\ & | & \langle\mathit{tree}\rangle\quad \langle\mathit{subtrees}\rangle \end{array}$$

The interpreter is then a parser that recognises that this format is followed and performs an appropriate *action* for each recognised symbol.

This implementation will emulate the layout of the *tree* command graphically: each label should be indented relative to its parent and connected to it, furthermore it should be below the previous label. This can be described by actions associated to each symbol as it is encountered: these are shown in figure 5. This is implemented by the small (plain) TEX file `tree.tex` shown in figure 6: the `\parser` macro selects the appropriate action based on the current symbol; each of the `\...action` macros implements the appropriate action from figure 5 using XY-pic with the 'arrow' extension (for details on how to use XY-pic refer to the reference manual, Rose and Moore, 1997). Running this on the source in figure 4 produces the tree shown in figure 7. The macros make use of the general font style of the program, *i.e.*, `\baselineskip` is used for distances to fit with the line skips used. Furthermore, some

[2] BNF is the notation for "meta-linguistic formulae" first used by (Naur et al., 1960) to describe the syntax of the Algol programming language. We use it with the conventions of the TEXbook (Knuth, 1988): "::=" is read "is defined to be", "|" is read "or", and "⟨*empty*⟩" denotes "nothing."

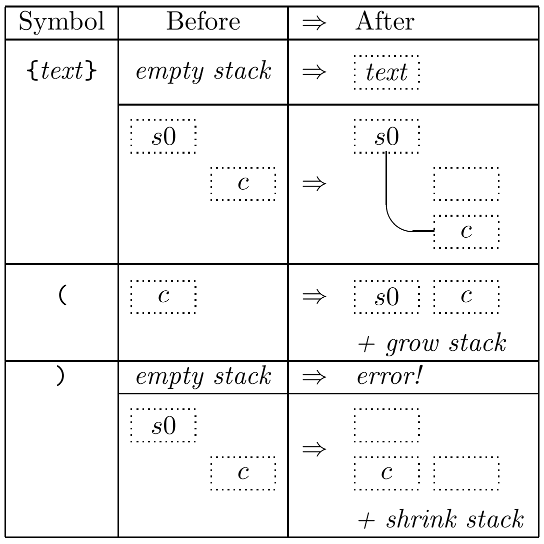


Figure 5: Tree interpreter actions.

components of the state change have been isolated as definitions — something that is possible with a generic macro language as TEX; a production version of the tree language the layout style of the tree should also be extracted into definitions. In fact, the "PostScript" sample of figure 3 was created using the `times` package with the redefinition

```
\def\branch{\save;s0!CD**@{-}\restore}
```

which tells XY-pic to make a plain line from the center bottom of the "parent" to the "child."

Exploiting the structured notation. Being able to vary the style this way is useful, of course. However, a common mistake when implementing packages such as `tree` is, in the author's opinion, to implement an interpreter that is too general. It is better to think of separate tasks as requiring separate embedded languages, implemented by separate interpreters, even if they happen to have the same syntax. One can go even further with *non-standard interpretation* of the information in the embedded language. Say that we wish to interpret the notion of "a tree" in a different way. For example, one could wish to show how a directory tree like our sample can be *grown*. This is slightly more complicated in that it requires our graphic library to include animation. This is possible in a (not yet published) module for XY-pic called `movie`. With this, animations are composed of "scenes" within which something varies from a starting point to an ending point. We can show growth by having a scene with just the root, then one level of branches, *etc.*; at a finer level we can let the branches grow gradually for greater

```
% tree.tex: Print \tree{ <tree> } as directory tree.

% Use Xy-pic, including ‘stack empty’ primitive.
\input xy
{\catcode‘\@=11 \global\let\sempty=\sempty@}
\xyoption{arrow}

% Idioms.
\def\FN{\futurelet\next}
\def\DN{\def\next}
\def\SP.{\futurelet\SP\relax}\SP. %

% <tree> parser.
\def\tree#1{\xy \beginaction \FN\parser#1\relax \endaction \endxy}
\def\parser{%
  \ifx\SP\next \expandafter\DN\space{\FN\parser}%
  \else\ifx\bgroup\next \DN##1{\textaction{##1}\FN\parser}%
  \else\ifx(\next       \DN({\openaction \FN\parser}%
  \else\ifx)\next       \DN){\closeaction\FN\parser}%
  \else\ifx\relax\next  \DN\relax{}%
  \else \DN{%
    \errmessage{<tree> build from (, ), and {text} only: not \meaning\next}}%
  \fi\fi\fi\fi\fi \next}

% Initial action : start fresh stack frame.
\def\beginaction{\POS @( }

% Interpretation action for {text} : typeset node and its branch!
\def\textaction#1{\node{#1}\if\sempty\else \branch \fi}
\def\node#1{\drop+!L\txt{#1}}
\def\branch{\ar @{-} ‘l/\jot s0+DC="s0" "s0" }

% Interpretation action for ( : move left and down!
\def\openaction{\POS @+c +R+/r1em/ +/d\baselineskip/ }

% Interpretation action for ) : move back below parent!
\def\closeaction{\if\sempty \errmessage{too many )s in <tree>}%
  \else \POS {;p+/r/:s0;p+/d/,x}@-c \fi}

% Final action : obliterate stack frame.
\def\endaction{\if\sempty\else \errmessage{missing )s in <tree>}\fi
  \POS @) }
```

Figure 6: The `tree.tex` macros.

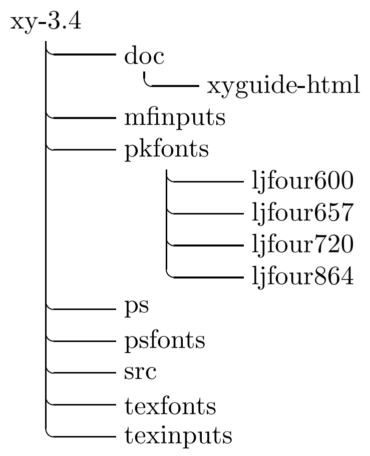


Figure 7: Generated directory tree.

effect. Using the movie class of XY-pic[3] one can produce an animation of the same tree, based on the same source, by modifying the actions for each component to draw it in a manner dependent on the time. The resulting animation can be found in the electronic version of this paper (Rose, 1997); here we can merely reproduce the (also automatically generated) "storyboard" of the animation, shown in figure 8. The source of the movie is shown in figure 9: the actions have been enriched with conditions for hiding leaves until the `\level` counter gets higher than their `\nesting` value, permitting them to appear. Some extra tricks make this happen gradually, using the `\F` construction of the movie class.

Conclusions

We hope to have shown that TEX is quite naturally extended with embedded languages and that this can be a convenient way of

- getting nice pictures and diagrams in papers,
- permitting aesthetic integration of text and diagrams, and
- ensuring that the information in the pictures can be exploited in alternate ways.

Acknowledgements. The author wishes to thank TUG '97 and BRICS for jointly funding my participation in the conference.

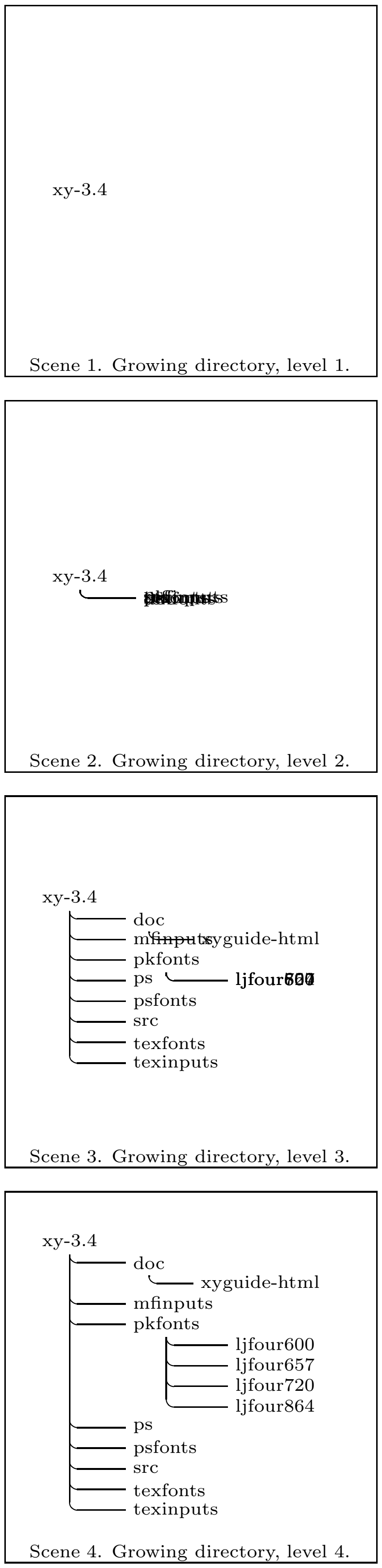


Figure 8: Growing the directory tree.

[3] Available in an experimental version with XY-pic 3.4.

```
\MovieSetup{height=18em,width=18em}% -*-LaTeX-*- animation of directory tree

\newcount\level
\newcount\nesting

\def\node#1{%
  \ifnum \level<\nesting \drop i+!L\txt{#1}%
  \else \drop+!L\txt{#1}\fi}

\def\branch{%
  \ifnum \level<\nesting
  \else \ar @{-} `l/\jot s0+DC="x" "x" \fi \relax}

\def\openaction{%
  \POS @+c +R+/r1em/ \F\down
  \global\advance\nesting by +1 \relax}

\def\down(#1){%
  \ifnum \level>\nesting \POS+/d\baselineskip/*{}%
  \else\ifnum \level=\nesting \POS+/d#1\baselineskip/*{}\fi\fi \relax}

\def\closeaction{\global\advance\nesting by -1 %
  \if\sempty \errmessage{too many )s in <tree>}%
  \else  \POS {;p+/r/:s0;p+/d/,x}@-c \fi \relax}

\level=0 \loop
  \advance\level 1 %
  \nesting=1 %
  \scene{%
    \input{dirtree.tree}%
    \caption{Growing directory, level \the\level.}%
  }%
\ifnum \level<4 \repeat
```

Figure 9: The `dirtree.texmovie` movie.

References

Adobe. *PostScript Language Reference Manual*. Addison-Wesley, second edition, 1990.

Goossens, Michel, S. Rahtz, and F. Mittelbach. *The LATEX Graphics Companion*. Addison-Wesley, 1997.

Knuth, Donald. *The TEXbook*. Addison-Wesley, second edition, 1988.

Knuth, Donald E. *The METAFONTbook*. Addison-Wesley, 1986.

Lamport, Leslie. *LATEX—A Document Preparation System*. Addison-Wesley, second edition, 1994.

Leathrum, Thomas and G. Tobin. "The mfpic package". Available from CTAN: `graphics/mfpic`, 1994.

Naur, Peter et al.. "Report on the Algorithmic Language ALGOL 60". *Communications of the ACM* **3**, 299–314, 1960.

Rose, Kristoffer H. "Very High Level 2-dimensional Graphics with TEX and XY-pic". Available from `http://www.brics.dk/~krisrose/Xy-pic/tug97/`, 1997. Electronic version of TUG97 paper.

Rose, Kristoffer H. and R. R. Moore. "XY-pic release 3.4". Available from CTAN: `macros/generic/diagrams/xypic`, 1997. See also chapter 5 of (Goossens, Rahtz, and Mittelbach).

van Zandt, Timothy. "The PSTricks package". Available from CTAN: `graphics/pstricks`, 1996. See also chapter 4 of (Goossens, Rahtz, and Mittelbach).

High Quality Labels on Included Graphics, using XY-pic

Ross Moore
Mathematics Department
Macquarie University
Sydney, Australia 2109
`ross@mpce.mq.edu.au`
URL: `http://www-math.mpce.mq.edu.au/~ross/`

Abstract

The XY-pic suite of graphics macros, through the `\xyimport` command, offers the ability to easily annotate and label imported graphics using TEX's full capabilities for handling mathematics and special fonts.

Introduction

Many modern software applications exist for constructing elegant graphics to present information of all kinds. This includes scientific data, maps, charts, tables and business graphics. Almost without exception, the handling of text for titles and labelling is rather limited. Typically just a single font may be used, superscript and subscripts are often not supported, and all but the simplest mathematical expressions can be constructed. Questions are continually asked, on Internet newsgroups, about how to include publication-quality typeset labels on the graphics produced using such programs.

A common solution to this problem is to first save the graphic in Encapsulated PostScript[1] (EPS) format, then use Adobe's Illustrator™, or other program, to render the `.eps` file, and edit it to add the annotations and labels. For many purposes this is effective, producing high-quality results. It is a strategy adopted with many scientific journals.

However this technique is not always sufficient (especially for mathematics) and has several drawbacks when the intention is to include the graphic within a TEX or LATEX document, quite apart from the extra expense of a piece of commercial software. It can be quite difficult to achieve the following:

S1 use same or similar fonts in the imported graphic as in the surrounding typeset text, at compatible sizes and styles;

S2 ability to resize the graphic (to fit available space in the typeset document) while retaining font-compatibility as in S1;

S3 inclusion of properly typeset mathematics; e.g. for axis-labels on graphs.

[1] PostScript® is a registered Trademark of Adobe, Inc. (Adobe Systems Incorporated, 1990).

Adjustment of font-sizes or style, and the overall size of the graphic, are decisions of style which can be quite independent of the main content of the graphic. Such decisions should be made late in the overall process of preparing a manuscript for typesetting. Thus to accommodate any changes, re-editing within Illustrator™ is required, and perhaps even regeneration of the original graphic using the specialised software. This may be impractical or even impossible to achieve, once the manuscript has been submitted for publication.

The correct place for such style-decisions to be made is within the TEX (or LATEX) source of the document within which the graphic is to appear. This can be realised effectively using XY-pic. Note that as a first step to achieve font-style compatibility, the original graphic should be created so that *it contains no explicit text or labelling*; for these will be added later during the TEX/LATEX processing. For most graphs it is not even necessary to include the axes, for these also may be added using XY-pic, as the examples show. (Indeed with some programs there are distinct advantages in leaving out the axes and any frame. We comment further about this near the end of the article.)

The `\xyimport` command

To use the `\xyimport` command of XY-pic, first the xyimport feature must have been loaded. This is done by including import among the options given to LATEX's `\usepackage` command when XY-pic is initially loaded, or by explicit use of `\xyoption`.

`\usepackage[...,import,...]{xy}`	(LATEX 2$_\varepsilon$)
`\xyoption{import}`	useable with any format.

The `\xyimport` command is used within an XY-pic picture or diagram in one of the following ways:

```
\begin{xy}                                  using LaTeX
\xyimport(1,1){⟨graphics⟩}
... further Xy-pic commands ...
\end{xy}

\xy\xyimport(1,1){⟨graphics⟩}          in any format
... further Xy-pic commands ...
\endxy
```

Here ⟨*graphics*⟩ denotes the TeX/LaTeX command to include a graphic within the document; e.g. LaTeX's `\includegraphics[...]{...}` command, or use of `\epsbox[...]{...}` from the `epsf` package, or use of `\psfig{...}`. Actually ⟨*graphics*⟩ may be *any* TeX source that creates an `\hbox`. In the examples below we use `\xyimport{\emptybox}` with a definition:

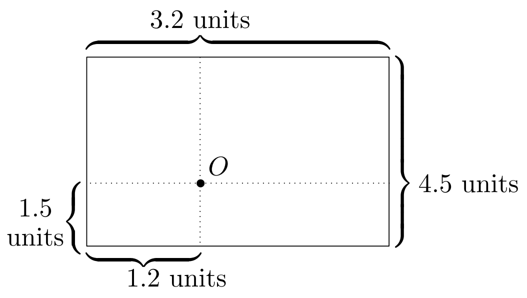
```
\providecommand{\emptybox}{\hbox to4cm{%
 \vrule height2.5cm width0cm\hfill}}
```

The `(1,1)` may be replaced by any pair of positive numbers. Furthermore a second pair of numbers may be optionally given:

```
\xyimport(3.2,4.5)(1.2,1.5){\emptybox}
```

The significance of each number is shown in figure 1. One sees that the first pair establishes the length of coordinate units, independently in horizontal and vertical directions. A second pair gives the offset, in these units, from the bottom-left corner to a point O, which becomes the origin for the coordinate system. When no second pair is given, O is located at the bottom-left corner.

Figure 1: Establishing an origin and coordinates.



With such a coordinate system, any location on the graphic (or even outside of it) can be specified as a pair of numbers (x,y). Using Xy-pic kernel commands, any ⟨*object*⟩ (e.g. text-strings, mathematics or anything typeset using TeX) may be positioned at that location.

In Xy-pic parlance, the `\xyimport` command "drops an ⟨*object*⟩", consisting of the ⟨*graphics*⟩, into a diagram. This specifies a ⟨*pos*⟩; i.e. a rectangle with a distinguished point, usually inside or on its boundary. This information alone is sufficient to drop further ⟨*object*⟩s into the diagram at locations determined using the edges of the ⟨*graphics*⟩ object.

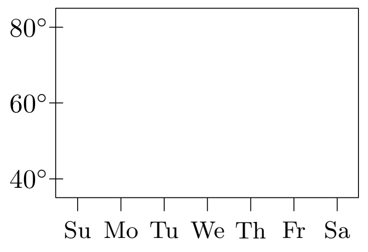
With the extra information provided by the number-pairs, the coordinate basis for the Xy-pic "graphics-state" can be adapted to whatever is most appropriate for the logical meaning of the (visual) content of the ⟨*graphics*⟩.

frames, ticks and labels

As an example, suppose the graphic displays data collected for different days of the week. One wishes to add labels on the axis to mark each day. The following code produces the picture in figure 2.

```
$$\def\degrees{^{\circ}}\def\low{35}
\begin{xy}
\xyimport(7,50)(-.5,-\low){\emptybox}*\frm{-}
,(.5,40)*@{|}*+!R{40\degrees}
,(.5,60)*@{|}*+!R{60\degrees}
,(.5,80)*@{|}*+!R{80\degrees}
,0;/r2pc/**{} % set horiz direction
,(1,\low)*@_{|}*++!U\txt\small{Su}
,(2,\low)*@_{|}*++!U\txt\small{Mo}
,(3,\low)*@_{|}*++!U\txt\small{Tu}
,(4,\low)*@_{|}*++!U\txt\small{We}
,(5,\low)*@_{|}*++!U\txt\small{Th}
,(6,\low)*@_{|}*++!U\txt\small{Fr}
,(7,\low)*@_{|}*++!U\txt\small{Sa}
\end{xy}$$
```

Figure 2: Axis labels on a weekly chart.



Note the following features which are apparent in figure 2 and the preceding code.

- the origin of coordinates can be outside the area of the ⟨*graphics*⟩, by having negative offsets from the bottom-left corner.
- a solid frame is placed using `*\frm{-}`; many alternate styles of frame are possible; see the Xy-pic Reference Manual (Rose, Kristoffer and Moore, Ross, 1997).
- tick-marks on the frame (or axes) may be positioned above or below (inside or outside) or through the edges. That strange combination `;p+/r2pc/**{}` is a technical device to switch the current direction to horizontal, so that the subsequent `*@_{|}` produce vertical tick-marks.

(The initial direction is vertical, so then `*@{|}`s come out as horizontal; i.e. rotated 90°.)

- text-labels are normally given a margin, e.g. as in `*++!U\txt` (add more `+`s for a wider margin), then positioned with an appropriate edge at the coordinate location. `!U` positions the U(pper) edge of the day-name abbreviations, while `!R` positions the R(ight) edge of the temperatures. Similarly L(eft), D(own), C(enter) and 2-letter combinations of these, are allowable. See the XY-pic Reference Manual (Rose, Kristoffer and Moore, Ross, 1997) for more details.

Also note how a coordinate may be specified from expanding a macro rather than typing explicit numbers. Indeed XY-pic kernel commands generally can be given as expansions of macros, provided `\drop` and `\POS` commands are used to ensure that appropriate parsers are activated. So if the repetition of data in the above code is of concern, then the following code produces exactly the same diagram as in figure 2.

```
$$\def\low{35}\def\off{.5}
\def\temp#1{\POS(\off,#1)*@{|}*+!R{#1^{\circ}}}
\def\wkday#1{\ifcase #1{}%
 \or Su\or Mo\or Tu\or We\or Th\or Fr\or Sa\fi}
\def\day#1{\POS
 (#1,\low)*@_{|}*+++!U\txt\small{\wkday{#1}}}
\begin{xy}
\xyimport(7,50)(-\off,-\low){\emptybox}*\frm{-}
\temp{40}\temp{60}\temp{80}\POS 0;/r2pc/**{}
\day1 \day2 \day3 \day4 \day5 \day6 \day7
\end{xy}$$
```

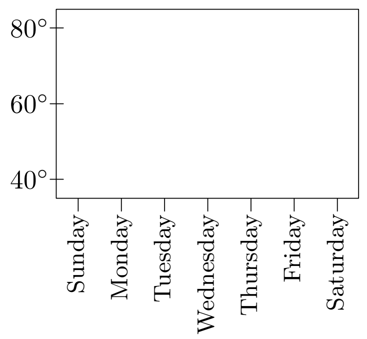
Rotated labels. To rotate text in TeX or LaTeX requires some PostScript® trickery. It can be done using LaTeX's graphics package, provided an appropriate dvi-driver (usually dvips) is loaded. Alternatively XY-pic has a rotate feature, which similarly requires support. Including an `\xyoption` command, as in the first of these lines, causes the required driver-code to be loaded and activated.

```
\xyoption{dvips,rotate}
\usepackage[dvips,frame,rotate,import]{xy}
```

Alternatively multiple features can be loaded with a single `\usepackage` command in LaTeX. For all the diagrams in this article the lower line above is sufficient to request the necessary options.

```
$$\def\low{35}\def\off{.5}
\def\temp#1{\POS(\off,#1)*@{|}*+!R{#1^{\circ}}}
\def\weekday#1{\ifcase#1\or Sun\or Mon\or Tues%
 \or Wednes\or Thurs\or Fri\or Satur\fi day}
\def\day#1{\POS (#1,\low)*@_{|}
 *[@^]++!U\txt\small{\weekday{#1}}}
\begin{xy}
\xyimport(7,50)(-\off,-\low){\emptybox}*\frm{-}
\temp{40}\temp{60}\temp{80}\POS 0;/r2pc/**{}
\day1 \day2 \day3 \day4 \day5 \day6 \day7
\end{xy}$$
```

Figure 3: ...with rotated labels.



The `[@^]` ⟨*modifier*⟩ rotates the text through 90° counter-clockwise; so that that it stands vertical, pointing upwards. Alternatively `[left]` achieves the same effect. Other such modifiers are `[right]` and `[flip]` (180°). After the rotation, adding first the margin and then aligning (`++!U`) gets the text positioned exactly as desired.

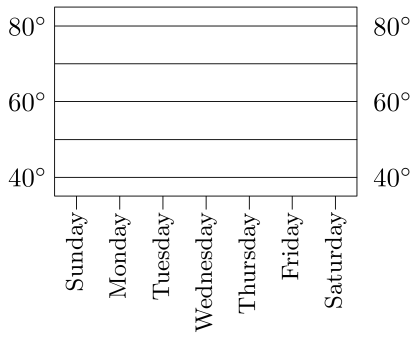
Axes and grid-lines

Grid-lines, stretching across the graph, from one side to the other or from top to bottom, are handled using XY-pic ⟨*connection*⟩s Given the coordinatisation, these are particularly easy. For example, a gridline at 60° could be placed into figure 3 with the code:

```
(.5,60);(7.5,60)**@{-}
```

Earlier code need change only slightly, using a macro `\Grline` to place grid-lines, labelled either side and without ticks. Another macro `\grline` places just lines. (The `\day` macro is as in figure 3.)

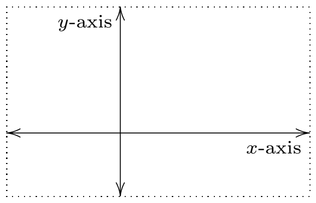
Figure 4: ...with horizontal grid-lines.



```
$$\def\low{35}\def\off{.5}
\def\grline#1{\POS(\off,#1);(7.5,#1)**@{-}}
\def\Grline#1{\POS(\off,#1);p*+!R{#1^{\circ}},
  (7.5,#1)**@{-},*++!L{#1^{\circ}}}
\begin{xy}
\xyimport(7,50)(-\off,-\low){\emptybox}*\frm{-}
\Grline{40}\Grline{60}\Grline{80}
\grline{50}\grline{70}\POS 0;/r2pc/**{}
\day1 \day2 \day3 \day4 \day5 \day6 \day7
\end{xy}$$
```

Coordinate Axes. The axes could be treated as a special form of grid-line. However, as usually they are required to pass through the origin of coordinates, then in Xy-pic they can be specified logically, without reference to coordinates at all.

Figure 5: Axes with arrows and labels.



```
$$\begin{xy}
\xyimport(3.2,4.5)(1.2,1.5){\emptybox}="a"
,*\frm{.} %  dotted outline
,"a"+L;"a"+R,**{}
 ?(0)*@{<};?(1)*@{>}**@{-} % arrow-tips
,?(1)*+!UR\txt\scriptsize{$x$-axis},
,"a"+D;"a"+U,**{}
 ?(0)*@{<};?(1)*@{>}**@{-} % arrow-tips
,?(1)*+!UR\txt\scriptsize{$y$-axis}
\end{xy}$$
```

After naming the ⟨*graphics*⟩ (using `="a"`) the idea is to set an empty ⟨*connection*⟩ (`**{}`) between the left (`"a"+L`) and right (`"a"+R`) edges. This establishes the correct direction for dropping the arrow-tips at either end (`?(0)*@{<}` and `?(1)*@{>}`). Then put a solid ⟨*connection*⟩, using `**@{-}`. Labels are positioned along each axis using the parametrization provided by this latter ⟨*connection*⟩. To shift the arrow-tips just outside the area of the ⟨*graphics*⟩, simply replace the arrow-tips lines with these:

```
?(0)+/l1\jot/*@{<};?(1)+/r1\jot/*@{>}**@{-}
?(0)+/d1\jot/*@{<};?(1)+/u1\jot/*@{>}**@{-}
```

Figure 6: ...with arrow-heads further out.

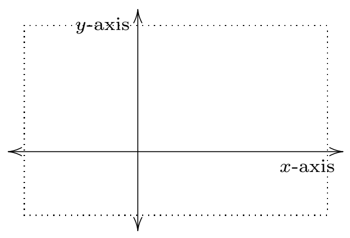
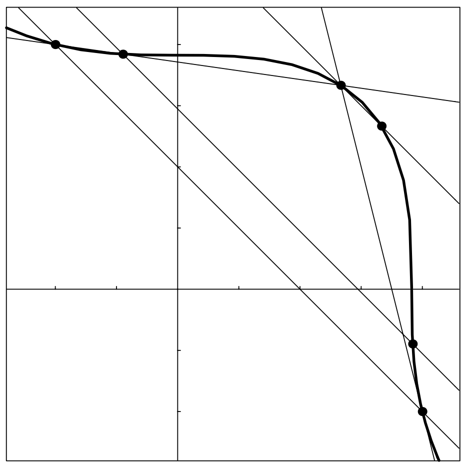
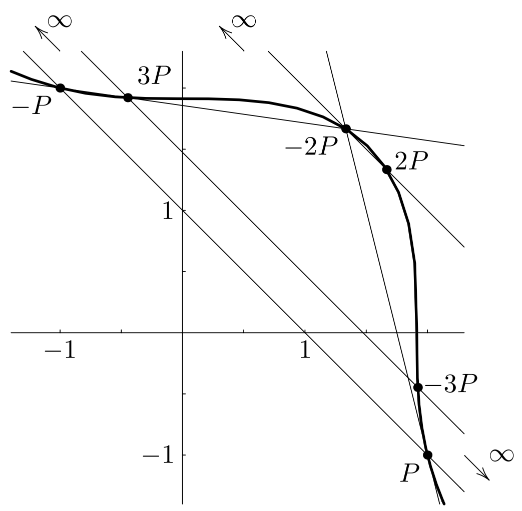


Figure 7: Original `.eps` graphic, with added frame...



framed contents of file:`import1.eps`

Figure 8: ...with mathematical embellishments.



Rational points on the elliptic curve: $x^3 + y^3 = 7$

Real-world example

The graphic in figure 7 was created using the *Mathematica* (Wolfram, 1993) program and appears in the XY-pic Reference Manual (Rose, Kristoffer and Moore, Ross, 1997). It is a variation of a diagram prepared for the book "Lectures on Fermat's Last Theorem"(van der Poorten, 1996)[2], for which I was TEX-nical editor. Many of its diagrams were typeset using `\xyimport`, as discussed here.

```
\xy\xyimport(3.7,3.7)(1.4,1.4){\ellipA}
,!D+<2pc,-1pc>*+!U\txt{Rational points
 on the elliptic curve: $x^3+y^3=7$}
,(1,0)*+!U{1},(-1,0)*+!U{-1}
,(0,1)*+!R{1},(0,-1)*+!R{-1}
,(2,-1)*+!RU{P},(-1,2)*+!RU{-P}
,(1.3333,1.6667)*+!UR{-2P}
,(1.6667,1.3333)*!DL{\;2P}
,(-.5,1.9)*++!DL{3P},(1.9,-.5)*!DL{\;-3P}
,(-1,2.3)*+++!D{\infty}*=0{},{\ar+(-.2,.2)}
,(.5,2.3)*+++!D{\infty}*=0{},{\ar+(-.2,.2)}
,(2.3,-1)*+++!L{\infty}*=0{},{\ar+(.2,-.2)}
\endxy
```

Warning. The appropriate numbers to use for the required argument to `\xyimport` can usually be read directly from the graphic being imported. Another way is to refer to the actual code which was used to generate the graphic, from its "plot range" say. However some software packages add an extra margin to this range, in particular when there are axes or a frame at the edges. Thus it can be necessary to estimate (read 'guess') the best numbers to use, when extreme accuracy is required in the placement of labels. A bit of 'trial-and-error' can be done from TEX or LATEX, without any need to regenerate the graphic.

Code for Figure 1. Here is the XY-pic code that was used to typeset figure 1. Notice how the edges and corners of the imported ⟨*graphics*⟩ are located logically, and saved (e.g. `"a"+R="aR"`) for repeated reuse. The extended braces are placed as a special type of "moustache" frame, on narrow rectangles derived from these corners and edges.

```
\xy\xyimport(3.2,4.5)(1.2,1.5){\emptybox}
 ="a",*\frm{-},*=0@{*},*+!DL{O}
,"a"+L="aL";"a"+R="aR"**@{.}
,"a"+D="aD";"a"+U="aU"**@{.}
,"a"+RD="aRD","a"+RU="aRU"
,"a"+LU="aLU","a"+LD="aLD"
,"aR"+/r5pt/."aRU"."aRD"*\frm{\}}
 ,*++!L\txt{4.5 units}
,"aU"+/u5pt/."aRU"."aLU"*\frm{^\}}
 ,*++!D\txt{3.2 units}
,"aL"+/l5pt/."aLD"!C*\frm{\{}
 ,*++!U(.2)!R\txt{1.5\\units}
,"aD"+/d5pt/."aLD"!C*\frm{_\}}
 ,*++!L(.5)!U\txt{1.2 units}
\endxy
```

[2] This book was awarded the prize of "Outstanding Professional/Scholarly Title in Mathematics, for 1996" by the Association of American Publishers.

References

Adobe Systems Incorporated. *PostScript Language Reference Manual*. Addison–Wesley, second edition, 1990.

Rose, Kristoffer and Moore, Ross. *XY-pic Reference Manual*. DIKU, University of Copenhagen, Universitetsparken 1, DK–2100 København Ø, 3.5 edition, 1997.

van der Poorten, Alf. *Lectures on Fermat's Last Theorem*. Canadian Mathematical Society Series of Monographs and Advanced Texts. Wiley Interscience, 1996.

Wolfram, Stephen. *Mathematica: A System for Doing Mathematics by Computer*. Addison–Wesley, 2nd edition, 1993.

CIRC – A Package to Typeset Block Schematics

Sebastian Tannert
Harbigstraße 14/67.02.01.06, D–14055 Berlin, Germany
`tannert@albert.physik.hu-berlin.de`

Abstract

The CIRC package is a tool for typesetting circuit diagrams. It defines several electrical symbols such as resistors, capacitors, transistors, etc. These symbols can be connected with wires in a very easy way.

To use CIRC, you only need METAFONT and LaTeX 2$_\varepsilon$. You do not need PostScript or any drawing tool. You can expand CIRC easily with your own symbols, written in METAFONT.

I started developing CIRC in order to have a tool for drawing electrical circuit diagrams; however, the use of CIRC has shown that it is a useful package for creation of a large range of block schematics.

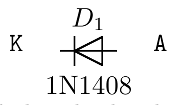
The problem of drawing block schematics can be divided into drawing the symbols and drawing the connections between them. Macros exist for both parts in CIRC. As is common in TeX, the diagram should be described in an abstract and generic way.

Step by step, I worked out macros so that you never have to deal with distances when describing a diagram. Distances are necessary only to determine spaces between symbols.

The main building blocks of CIRC are so-called pins which mark points for referencing.

Most block schematic diagrams consist of symbols connected at determined points at or around the symbols. These are called pins. Every pin gets a name within CIRC that is composed of the name of the symbol, a number given to the symbol in the current diagram, and a short pin name. The last is a specific abbreviation for the pin.

The following example will explain the ideas.



One pin is called cathode, the other anode. The short pin names are K and A. (Cathode is abbreviated as in German with a K.) The diode has the abbreviation D, and in the above example it is numbered with 1. So the pin names for this diode are D1K and D1A.

Using CIRC, you draw in a 7 pt (≈ 2.5 mm) grid.

A current drawing position exists and the drawing direction is specified with every symbol.

To get the example you have to type in: `\D1 {1N1408} K r`. The last two letters are specific for the drawing direction. They mean: draw the cathode (**K**) at the current drawing position and the symbol to the **r**ight. After drawing, the position is changed to the anode.

More or less, all symbols are drawn in the same manner.

If you have to draw larger diagrams, it is necessary to change the position to that of a specific pin. This is easily done with `\frompin` *pin name*.

As you can imagine, this moves the position to the given pin. With `\_` you can change by amounts of the grid. The position will be moved by two units of the grid **d**ownwards by `\_ 2 d`.

Mostly you have to connect the symbols with solid lines. Up to this point, only horizontal and vertical lines can be drawn with CIRC. With the command `\-` you can draw a solid line with a given length.

For example: `\- 3 u` will draw a line of 3 units of the grid **u**pwards. But this method of drawing with a given distance should only be used to draw lines that give space between symbols. The more generic way is to use `\htopin` or `\vtopin`.

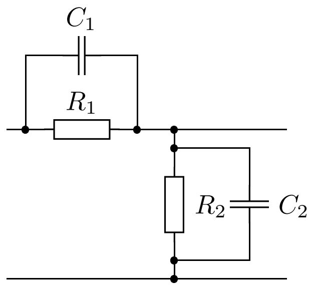
After both of these commands, a pin name is expected. These commands draw a line from the current position to the position of the given pin. The command `\htopin` uses only the x-part of the given position and draws a horizontal line; `\vtopin` does the same vertically.

Now you have an overview of the commands in CIRC. Please note that all parameters are seperated by spaces, and empty lines are like spaces.

The complete syntax and all available symbols with their names may be obtained from the documentation. CIRC and the accompanying documentation is stored on CTAN in the directory `macros/generic/diagrams/circ`.

As of this issue of *TUGboat*, I have had no time to update CIRC on CTAN, and hope I can do it in February of 1998.

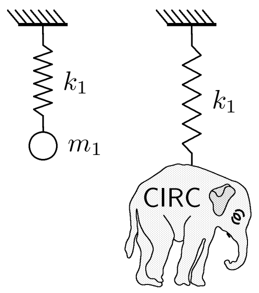
Finally other examples.



Produced by:

```
\begin{circuit}0 \P1
\- 1 r \.1 \- 1 r \R1 {} r
\- 1 r \.2 \- 2 r \.3
\- 1 d \.4 \- 1 d \R2 {} d
\- 1 d \.5 \- 1 d \.6
\htopin P1
\frompin .1 \- 4 u \hcenterto R1 \C1 {} r
\htopin .2 \vtopin .2
\frompin C1l \htopin .1
\frompin .4 \- 4 r
\vcenterto R2 \C2 {} d
\vtopin .5 \htopin .5
\frompin C2t \vtopin .4
\frompin .3 \- 6 r \P2
\frompin .6 \htopin P2
\end{circuit}
```

As described in the opening paragraph, you can draw more than electric circuit diagrams with CIRC.



A small library with symbols for vacuum techniques and commands for drawing flowcharts is also available. Andreas Tille has also written a library for optics.

DVIPDF and Graphics

Sergey Lesenko
Institute for High Energy Physics
Scientific Information Department
142284 Protvino, Moscow Region, Russia
lesenko@mx.ihep.su

Abstract

This paper describes how the *dvipdf* program inserts BMP, JPEG, PNG and EPS graphics into its output. It also discusses geometric transformations of image, text and rule objects.

Introduction

For today's scientific publications, we have to provide for accessibility via the Internet, and need an effective presentation format for our documents which supports a wide range of hypertext and graphic features. One option is the Portable Document Format (PDF) [1], probably the most popular system for distributing complex formatted documents. PDF permits us to use color, geometric transformations and included images (vector and bitmap). This paper describes how a document in PDF with such graphic features may be prepared using TEX and *dvipdf*.

Graphic commands

Although TEX has only a limited capability to deal with graphics, it has the wonderful property of the `\special` command, which allows us to perform arbitrary tasks at the level of driver programs, which include *dvipdf*. This is based on *dvips* [2], and it would be preferable to support the same commands. However, *dvipdf* is oriented to a more confined output format, and a new set of `\special` commands has been introduced to permit better performance. The syntax for these has already discussed in [8], and all the `\special` commands for graphics are listed in Table 1

Only a minimal number of parameters for the DVI file is required, and the full set needed in the PDF output is computed by *dvipdf*. This means that writing macros or incorporating the `\special` commands into style packages is not difficult. The `color` [3], `graphics` [4] and `graphicx` [4] packages, for instance, have 'dvipdf' drivers which make use of the appropriate `\special` commands.

A particular problem with inclusion of bitmap graphic formats like BMP, JPEG and PNG is that the user has to provide TEX with the image size (directly or via an additional *bb* file). Encapsulated

Geometric	
Begin rotation	`pdf: /ROT` *angle* `<<`
End rotation	`pdf: /ROT >>`
Begin scaling	`pdf: /SC` *hscale vscale* `<<`
End scaling	`pdf: /SC >>`
Color	
Begin color	`pdf: /C` *Yellow* `<<`
End color	`pdf: /C >>`
Page	
Page color	`pdf: /BG` *Blue*
Page rotate	`pdf: /ROTPAGE` *angle*
EPS	
Mode	`pdf: /IMAGE` *rgb16m*
Resolution	`pdf: /RES 600`
Insertion	`pdf: /GRAPH` *file llx lly urx ury*
BMP, JPEG and PNG	
Insertion	`pdf: /GRAPH` *file width height*

Table 1: `\special` commands supported by *dvipdf*

PostScript files have a `%%BoundingBox` line which TEX can read, but it cannot (easily) read binary formats. Scaling and rotation can use the commands of the standard LATEX graphics package.

For example, the following code inserts a JPEG image, rotates it, and scales it. Note the optional argument to `\includegraphics` to specify the original image size:

```
\resizebox*{2in}{!}{%
\rotatebox{90}{%
\includegraphics[12cm,9cm]{photo.jpg}}}
```

Basic algorithms

Since geometric transformations in PDF are one of its most powerful features, we will begin our description of *dvipdf* innards in this area.

To allow both single level transformations and nested transformations, *dvipdf* supports a 16 level stack. For each level eight parameters are recorded: the initial coordinates (x, y) of the reference point for the layer, offsets to its position after transformation (dx, dy) and factors (a, b, c, d) for the current transformation matrix (CTM). Each new transformation (or layer) pushes the current parameters and calculates new parameters. Corresponding returns from each layer pops the parameters. To define the coordinate of each point on current layer, *dvipdf* computes its position (ddx, ddy) in relation to the reference point and adds the coordinate of the reference point ($x + dx$, $y + dy$).

The transformations for points apply equally to other types of object (rule, text and image). Let us consider the simplest object (a rule); this may be dealt with as a rectangle or a region — a rectangle is described with two points, and a region with four points. To optimize the output, the two types of rule (with rotating and without it) are treated differently. Rules without rotating are treated as rectangles and described with two points A, B (Fig. 1(a)). This requires four arguments (ulx, uly, w, h): base point, width, and height. Rotated rules are treated as regions with four points A, B, C, D (Fig. 1(b)). This requires eight parameters: (llx, lly, lux, luy, urx, ury, ulx, uly)

Apart from geometric transformation, there are other PDF objects that use the same algorithm (although they are used only during interactive viewing and are not printed). These objects are link annotations and bookmarks. To specify a link annotation, and the destination for a link annotation or bookmark, we need to know the rectangle of the location. To compute the rectangle, we firstly utilise the algorithm for the definition of a region, and then calculate a bounding box (llx, lly, urx, ury) for this region. This is defined using two points A, B (Fig. 1(c)).

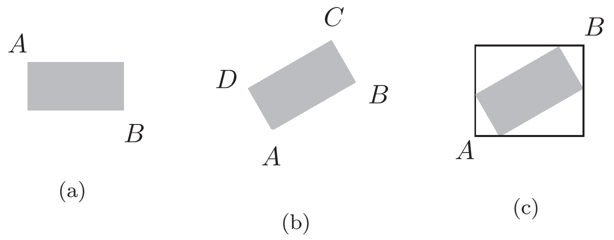


Figure 1:

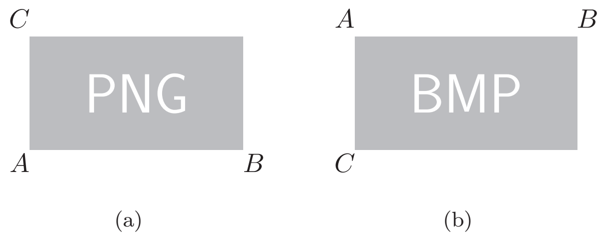


Figure 2:

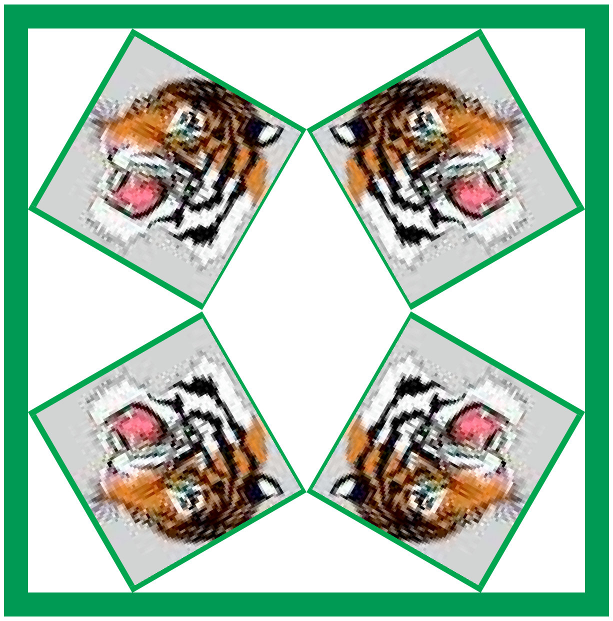
Now we can consider transformation of text and images. These objects are managed using the PDF page marking operators **Tm** (for text) and **cm** (for images). They are supported in *dvipdf* in similar ways, using the geometric figure of a parallelogram. It is described by three points (Fig. 2(a)). The first point is lower-left corner, the second point is the lower-right corner and the third point is the upper-left corner. The first point A is defined as a base point (x, y) and other two points B, C as offsets (a, b) and (c, d) from the base point. These six parameters (a, b, c, d, x, y) allow us to do all the transformations that are needed.

Since a BMP image has a bottom-to-top order and it is saved during the initial parse, its parallelogram is described by other points (Fig. 2(b)), where the base point A is the upper-left corner.

Color management for text (here 'text' includes rules too) is nested. The Current version of the color stack permits us to reverse the order of pages and to produce separated pages.

Images

It is usually impossible to know exactly with which resolution our document will be viewed or printed by its eventual receiver. So when we deal with bitmap image formats like BMP, JPEG and PNG, the best



```
\setres{300}
\setimage{rgb16m}
\fboxsep = 0 cm
\fboxrule = 0.6cm
\baselineskip = 0pt

\def\epsm{\textcolor[named]{Green}{\fbox{%
\scalebox{-1}[1]{%%
\includegraphics{tiger.eps}}}}}


\def\new{\scalebox{-1}[1]{%
\rotatebox{30}{\resizebox*{2in}{!}{\epsm}}}%
\rotatebox{30}{\resizebox*{2in}{!}{\epsm}}}
\def\newt{\scalebox{1}[-1]{\new}}
\resizebox*{8cm}{!}{%
\textcolor[named]{ForestGreen}{\fbox{%%
\vtop{\hbox{\new}\hbox{\newt}}}}}
```

Figure 3:

we can do is preserve the original resolution in the PDF output. This allows us to avoid getting concerned with intermediate conversion of resolution. The same principle is also applied to the mode of the image, which is simply preserved (i.e., mono, gray, RGB, CMYK and Indexed).

Dealing with EPS images is a different matter. This type of vector image is inserted after processing with GhostScript [6]. Ghostscript allows us to create a bitmap image, with appropriate settings of mode and resolution. How do we establish those settings? To work them out, we need to set the appropriate resolution (R)and mode via `\special` commands. *dvipdf* can then compute the real resolution R_x and R_y for Ghostscript:

$$R_x = R \cdot \frac{W_{actual}}{W_{original}}$$

$$R_y = R \cdot \frac{H_{actual}}{H_{original}}$$

where $W_{original}$ and $H_{original}$ are the width and height, (as derived from the BoundingBox in the EPS file), and W_{actual} and H_{actual} are the sizes of the parallelogram sides. In some images the form may not be a rectangle, so they are calculated as the distance between corners.

Using the current version of Ghostscript, bitmap images are produced corresponding to the page size, so the possibility has been added of producing them with the rectangle as the BoundingBox.

At present we ask Ghostscript to produce temporary files, and these are then placed in the output (in future versions it should be possible to merge Ghostscript's output stream directly into *dvipdf*'s output). We can select an option to save the temporary files and to re-use them for the next run of *dvipdf*, if the same parameters are used for images. The unique parameters for each image are recorded as EPSF structured comments (file name, date, time and file size, and parameters for for Ghostscript processing, i.e., mode and X, Y resolutions). We can generate these parameters and compare them during the next processing by *dvipdf*, and name temporary files accordingly. We adopt the simple solution of computing a Cyclic Redundancy Check value (CRC) [7] on the basis of the parameter set and use this CRC as the basis for naming temporary files. We can thus avoid calling Ghostscript wherever possible, and so reduce the time taken to prepare our PDF document.

Repeated images (for example a logo on every page of a document) are organized as references to a common object with just one instance. For bitmap images the basis of deciding if something can be stored as a common object is just the name of graphics file — rotating and resizing are done at the time of the object instance. For EPS images, we also need to consider resolution and mode. To illustrate this use of common objects, four tigers are placed together in Fig. 4, with the TEX code used to produce the picture. Only one copy of the graphics file is included in the output.

Acknowledgements

I would like to thank Michel Goossens and Mimi Burbank for their support of this project, Laurent Siebenmann for test samples, David Carlisle and Sebastian Rahtz for adapting their graphics and hypertext packages to *dvipdf*, and Sebastian Rahtz for editing this paper and helping with the English.

I am very grateful to Tim Bienz for his patient explainations of PDF, and Peter Deutsch for his useful remarks.

Finally thanks to Tomas Rokicki for his wonderful *dvips* which has been the starting point for this project.

References

[1] Tim Bienz and Richard Cohn, *Portable Document Format Reference Manual*, Adobe Systems Incorporated, 1993, Addison-Wesley Publishing Company. ISBN 0-201-62628-4. This document is available from `http://www.adobe.com/supportservice/devrelations/PDFS/TN/PDFSPEC.PDF`

[2] Tomas Rokicki, *Dvips: A TEX Driver*, distributed with *dvips*, version 5.58, 1994. electronic distribution from `labrea.stanford.edu`.

[3] David Carlisle, *The color package* (on CTAN)

[4] David Carlisle and Sebastian Rahtz, *The graphics package* (on CTAN)

[5] David Carlisle and Sebastian Rahtz *The graphicx package* (on CTAN)

[6] L. Peter Deutsch, *Aladdin Ghostscript* version 4.03, 1996 electronic distribution `ftp.cs.wisc.edu://pub/ghost/aladdin`

[7] L. Peter Deutsch, *RFC 1952: GZIP 4.3 specification*, `ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html`

[8] Sergey Lesenko, *The DVIPDF Program,* TUGboat 17, 3, September 1996, pages 252–254.

From SGML to HTML with help from TeX

Christopher B. Hamlin
American Institute of Physics, 500 Sunnyside Boulevard, Woodbury, New York 11797
`chamlin@nassau.cv.net`

Introduction

At this time there is still no fast and standard way of presenting mathematics in HTML pages. Various ideas have been tested and the W3C has just released a draft math markup proposal. When combined with freely available fonts containing the required mathematical characters, we can see much potential for the future.

For now it seems that there is only one common denominator suitable for quickly browsing HTML with lots of math: preprocess the mathematics into images and embed links in the HTML that call in the images.

The following describes work at AIP in converting physics research articles from SGML to HTML, creating the needed images for math. This allows us to present abstracts or full articles on the web. The tools used are TeX, AMS-LaTeX, dvips, Image Alchemy (for image processing), various PostScript (PS) fonts, gcc, and perl.

Source SGML

AIP uses a DTD based on the ISO 12083 DTD for its abstracts and articles. Math is basically in the 12083 model, though there are minor extensions in the AIP DTD. The AIP DTD also includes another, simpler math model for backward compatibility with older data. Deviations from standard 12083 math are handled by transforming them into 12083 format, so they can be ignored for now.

Though it might be possible to just set all math as GIFs, it seems that it should be possible to use the ability of HTML in setting bold, italic, superscripts, and subscripts, along with a handful of special characters. First we look to see what the SGML math looks like. Note that this discussion may include some interpretation specific to the AIP use of SGML math, but the ideas will probably be similar to other uses. We will be mainly focusing on inline math; displayed equations will be set as single images using the same techniques.

The 12083 math model can be simplistically viewed as a string of items to be set. Each item consists of a base followed by optional embellishments. The base can be a character, entity, or element. The embellishments can be a sup (superscript), inf (subscript), top, bottom, or middle. Together, sups and infs will be called “scripts”. A script may be set either before the base (the location attribute is “pre”) or after the base (the location attribute is “post”). Top embellishments are set over the base, bottoms are set below, and middles are set as overprints on the base.

It should also be noted that while sups and infs can only be set explicitly with the use of the sup or inf elements, there are a number of entities that are implicit embellishments, either top, bottom, or middle. Also, the order of the embellishments in the SGML is used in rendering them: scripts are set left to right, while top and bottom are set from the inside out.

Here are a few examples to show how embellishments work.

scripts:
`a<sup>2</sup>` gives a^2
`a<sup location="pre">2</sup>` gives ${}^2a$
`a<sup>2</sup><inf>1</inf>` gives a_1^2
`a<sup>2</sup><inf arrange="stagger">1</inf>` gives $a^2{}_1$

top (explicit, with `<top>` tag):
`a<top>&ast;</top>` gives $\overset{*}{a}$

top (implicit, with entities):
`a&dot;` gives $\dot{a}$
`a&macr;&dot;` gives $\dot{\bar{a}}$

middle:
`a<middle>&ast;</middle>` gives $\mathrlap{*}a$

bottom:
`a<bottom>&ast;</bottom>` gives $\underset{*}{a}$

combination:
`a&macr;<sup>2</sup><top>&ast;</top><inf>1</inf>` gives $\overset{*}{\bar{a}}{}_1^2$

In considering the interactions between the base and its embellishments, it can be seen that there are three areas that can be set separately: the pre scripts; the base and any tops, bottoms, or middles; and the post scripts. There is little or no interaction among these three zones other than using the height/depth of the base for setting the script baseline. We will ignore this effect for in-line math since it rarely has much effect. (Remember that the resolution of a computer screen is 72 dots per inch, so the smallest unit on a screen is ≈1 pt.) We set display equations as a unit so this question does not arise there.

Reading and processing the SGML

The controlling program in the SGML to HTML translation is a C program called `s2h`. There are three stages in the translation: read and parse the SGML, apply transformations to the SGML, and finally produce the desired output. Splitting the process up into these parts gives `s2h` some flexibility. Input may come from standard input, a file, or a database query. Tranformations may be applied depending on the output desired or the type of input. Output can be to standard output, a file, or to a string for database insertion and can be in various formats (ASCII, HTML, TEX).

The SGML input is read by `s2h` and parsed into a simple tree structure that directly represents the structure of the SGML. Nodes are elements, characters, or entities. After the tree has been created, several transformations are applied.

For example, the equation

$$\mathbf{b} + 2x^2{}_1 \pm 3\ddot{a}$$

could be input in SGML as

```
<bold>b</bold>&plus;2x<sup>2</sup>
<inf arrange="stagger">1</inf>&plusmn;3a&uml;
```

and would create a tree with `<bold>`, `&plus;`, `2x`, `<sup>`, `<inf>`, `&plusmn;`, `3a`, and `&uml;`, on the first level. Only the content of the elements would be on a lower level.

Some transformations would normally be applied to the tree at this point:

1. Character transformations to normalize the input. This is mainly to identify accented characters that can be set in HTML.
2. The math is normalized to represent the older, simpler AIP math model in the newer 12083 math model.
3. An older AIP font model is normalized to the the 12083 model.
4. The structure of the tree is changed to directly attach the embellishments to their bases and to separate the various types of embellishments into different lists. This takes the embellishments out of the normal tree structure and associates them strictly with the base.
5. Contiguous character data is normally kept in long strings rather than split up into separate nodes, but it may need to be split up so that a single character can be used as a base for an embellishment.

In the transformed tree, `2x` would be split up and the sup and inf would be attached to the `x` on its post script list. The `3a` would also be split and the `a` and `&uml;` would be combined into `&auml;`.

Now we have a tree where the top level consists of just bases. The embellishments are out of the normal structure and attached to the bases. They are also sorted into separate lists for pre scripts, post scripts, tops, middles, and bottoms, retaining their relative ordering within each such list since this will determine the order in which they are set.

Now, some output

To start, we look at how to output to HTML for math that doesn't need GIFs. This means there can be no tops, bottoms, or middles, and complicated bases — e.g., fractions or roots — are impossible.

To output to HTML we just move along the top level of the SGML structure. For each base, first do the pre list, then do the base, then do the post list. In doing the base we can first do work on reaching the base, then we output the base's content, and finally we can also do work on leaving the base. When we work on the content of the base (or an embellishment) we just apply the same idea to the first level of its contents. In other words, this simple left-to-right processing just works recursively to format a transformed SGML tree or subtree.

In outputing the bold element, the HTML tag for starting bold (`<b>`) is output at the start and the HTML for ending bold (`</b>`) is output at the end. Whatever the contents are, they will be output in between these tags and so will end up bold.

Here is our "simple" example output in HTML:

```
<b>b</b>+2<i>x</i><sup>2</sup>
<inf>1</inf>&plusmn;3<i>&auml;</i>
```

Notice that there is already the complication of setting Latin letters in italic when they are in math. This can be controlled because any math in an article would be inside an SGML formula element. When outputing character or entity data it is necessary to check to see if you are in a formula, and if so to put the character in italic if it is a Latin-based letter.

We see that a fair amount of math can be set with just the normal HTML facilities. However, even the simplest math will run aground because of the limited character set available in HTML. This is the beginning of the use of TeX in the project.

GIF interlude

When including GIF images in-line in HTML text there arises the problem of vertical alignment. By using the alignment attribute of the HTML img tag, one can align an image on the baseline by either its bottom or middle. When setting a character or math image that does not extend below the baseline we just align the image by its bottom. For example, $\boxed{\alpha}$. However, what do we do if we wish to set a β? Aligning by its bottom gives $\boxed{\beta}$. Aligning by its middle gives $\boxed{\beta}$. One compromise is to align by its middle after making sure that the top and bottom halves of the GIF are equalized by adding in white space. This gives $\boxed{\beta}$. Note that this can have a very bad effect on the leading. However, leading is already bad in HTML, and any in-line images or superscripts just make it worse, so this may not be such a high price to pay.

Setting special characters

Let us now work on creating simple GIFs to represent special characters (entities in the SGML).

1. First we keep a control table for all legal entities. This table will contain ASCII, TeX, and (possibly) HTML translations of all entities.
2. When translating, if there is no HTML representation then we use the TeX translation to create a TeX file.
3. Since TeX knows the width, depth, and height of all the boxes it sets, have TeX typeset the entity and check the depth. If it is >1 pt, say, balance the height and depth by setting the lesser of the two equal to the greater. Alignment info is written to the log file to tell the translator whether the final GIF should be bottom or middle aligned. The translator reads the log file after TeX runs and sets the GIF alignment via the align attribute of the HTML img tag.
4. `dvips` is run to create a PS file from the `dvi` file. The PS file can then be rendered into GIF format using Image Alchemy. But it's a problem to keep the extra white space needed to balance the top and bottom of the GIF for middle alignment. Image Alchemy can autocrop a PS file when creating a GIF, but then it (correctly) throws away the white space. `dvips` can be told to create an EPS file with a BoundingBox comment. This comment gives the lower-left and upper-right points of a box that contains all the printing on the page. This comment can be used by Alchemy for cropping if present. Unfortunately, `dvips` (correctly) does not include white space in the bounding box. But TeX knows the height, depth, and width of the box being output, *including* white space. Since each GIF is set separately, and all we care about is the one piece of math we are setting, we can further customize the TeX run:
   (a) Have TeX write into the log file the dimensions of the math output box.
   (b) Override `\output` to do nothing but ship out the box, which then comes out with the upper-left point of the box 1 inch in and down from the upper-left corner of the dvi page.

   From this information accurate bounding-box values are calculated and then inserted into the PS `BoundingBox` comment. Telling Alchemy to clip to the bounding-box dimensions then gives (fairly) accurate clipping and lets us retain the white space inserted by TeX.

Reusing stock GIFs for special characters

Since we don't want to recreate character entities each time, we can create them before creating any HTML. Then the same GIF will be called in each time the character is shown on the screen and no computation needs to be done at translation time. We can do this as follows:

1. Go through the control file to get all the entity names and their TeX translations.
2. Render all the characters to GIFs. Collect the names of all the entities that need to be middle aligned.
3. When translating, call in the stock GIF with an img tag. Set the alignment using the information collected in the preceding step.

How we store and recall the GIF files may change according to the final product requirements, but storing them according to entity name is convenient since it is already a unique identifier in AIP's SGML.

In building the `s2h` program the alignment information is stored in an alignment control file as a simple list of entities that are to be middle aligned. In creating the `s2h` executable this control file is run through a perl script that creates C source code. This code is then compiled into `s2h` so that it knows how to align all the stock entity GIFs when it calls them in.

Scripts complicate the use of stock GIFs since the GIFs we have created will be the wrong size for a script. Therefore we create scriptsize versions of all the characters and using these smaller versions when characters are set in scripts. The use of images in scripts works well in HTML, fortunately. For example, the SGML `x<sup>&alpha;</sup>` would be translated to `x<sup><img align = "bottom" src = "alpha-script.gif"></sup>` in HTML.

Custom GIFs for math

We have seen that all the special characters can be created beforehand and much can be done with standard HTML tagging. What can't be done, and how does the translator actually decide what to do in HTML and what to send through TeX at translation time?

Certain elements cannot be done in HTML at all. This includes roots, overlines, fractions, and arrays. We call these bad elements. Entities that can normally be shown using our stock GIFs can present a problem if occurring inside bold or bold italic. Such entities are called bad entities. When scripts are kerned or contain bad elements or entities, they are bad scripts.

With the preceding definitions, and remembering our notion of a base with various attached embellishments, we can now define an algorithm:

When translating SGML, move along the top level of the tree. For each item you encounter, do the following:

1. If any of the pre script embellishments are bad scripts, do all the pres with TeX. Otherwise do them all via HTML.
2. If the base is a bad element or entity, or if it has top, bottom, or middle embellishments, do the base and its embellishments and contents in TeX. Otherwise do the base element in HTML and apply this algorithm on the contents of the base.
3. Apply rule 1 to the post embellishments.

Note that the alignment mechanism outlined for special characters works fine for more complicated math. Display math (the `<dformula>` element) is simply defined as a bad element so that the entire equation is done as a GIF. GIFs for display math are always bottom aligned with no height or depth adjustment.

Reusing custom GIFs

After all this, it would be nice to be able to reuse the GIF for $\bar{M}$, or for the script combination $\frac{1}{2}$. In fact, using just a few base and script GIFs can produce a lot of different math since the bases may be used with any of the script GIFs or with HTML scripts, and the script GIFs may be used with any of the base GIFs or with HTML bases.

In order to track the GIFs an array of pointers into the SGML tree is maintained. Before rendering a subtree it is checked against the subtrees in the array and a previously rendered GIF is used if one exists. It is also necessary to compare the font contexts since this can have an important effect on the subtree's rendered appearance (see following section).

The *ideal* solution would be to "stringize" the SGML subtree (along with its font context) into a key that uniquely describes the GIF. Then commonly used math fragments could actually be reused for many articles, producing further efficiency. This idea has not been implemented.

Font handling

`s2h` sets all TeX fragments in math mode. This was done because it was rare to set nonmath through TeX. Only a few text-mode accents and a hyphen had to be provided in order to set all the usual TeX characters in math mode. Font-style changes are used in AIP's SGML instead of entities for each letter. For example, script M is input as `<script>M </script>`, not `&scrM;`, and bold script M is set as `<bold><script>M</script></bold>`.

It took a little while to realize how easy it is to implement this font scheme in LaTeX 2$_\varepsilon$, thanks to its designers' forethought. Everything maps simply: math is set with `\mathnormal` (the default), roman with `\mathrm`, italic with `\mathit`, script with `\mathcal` (using Y&Y's MathTime script, which includes lowercase letters), etc. Bold is set with `\boldsymbol`. If bold in math implies upright Latin letters, as it does for AIP's SGML, merely use `\boldsymbol{\mathrm{...}}`. Note that the difference between math and italic is obvious in TeX — in HTML it is necessary to look at characters and entities output from math to see if they must be set in italic with HTML's `<i>` tag, while leaving numbers, punctuation, and other characters and entities in the default roman.

The algorithm for deciding what needs to go through TeX seems to work generally but its recursive nature means that we may be several levels deep before TeX is invoked. So small pieces of math may be set without their normal font context. For example, a small piece of math may be `<bold>x&minus;&alpha;</bold>`. There is no need to set the x or minus via TeX, but the bold alpha cannot be set with HTML and so must be set with TeX (it is a bad entity, according to our algorithm). But it is not enough to set `$\alpha$` since it will not be set bold. It is necessary to create a font context in TeX that matches that of the small piece of math we are setting. In the case of the above we would need to set `$\boldsymbol{\mathrm{\alpha}}$`. The `\mathrm` is not needed for this case but would be necessary if a Latin letter occurred.

A special case is the occurrence of special accents outside math; for example, for names. The character $\dot{z}$ is not uncommon but is not available in HTML. If this occurs in an author's name then it should not be set in italic. Thus it is necessary to assume a roman font context if the math fragment being set is not inside math in the SGML. So if you had `z&dot;` outside math you would set the `z&dot;` in TEX as follows: `$\mathrm{\Dot{z}}$`.

A further complication is that the HTML tagging can impose a font context that may or may not be known when the SGML is converted. Perhaps the authors' affiliation will be set italic in HTML, perhaps not. It may be possible to control this effect when known beforehand.

Customizing TEX

As mentioned, LATEX 2$_\varepsilon$ is an excellent base for such work, especially in its font handling. Though the number of fonts and families is a potential problem, it never has been in practice. Using the AMS-LATEX package for its math and font handling provides most of what one needs for math typesetting. The `\underset` and `\overset` macros can be used to implement bottom and top embellishments, respectively. AMS-LATEX also does a nice job of setting combinations of the most common accents. The amssymb package provides many predefined symbols from the AMS Fonts.

After using the AMS packages a few hundred characters were still missing, about half of which were phonetic characters. These characters already existed in PS fonts used by AIP's composition system so two virtual fonts were created — one for phonetic characters and one for other characters. Several characters were created with TEX macros (e.g., lambda-bar).

After the fonts were created there was still some work to do in defining accents that TEX doesn't normally provide. A simple overprinting macro was defined for middle embellishments. `\mathaccent` was used for overaccents where actual accent characters already existed in fonts. `\overset` was used to implement overaccents where only normal-sized characters exist (e.g., harpoon overaccents). Similarly, underaccents were implemented by using either `\ooalign` (e.g., for a math-mode cedilla) or `\underset`.

Though the interaction between top, middle, and bottom embellishments could be a factor, it never really is. The different types rarely get used together, and the quality that can be achieved via GIFs is already generally low.

Custom Legal Documents for the IBM AutoLoan Exchange

Douglas Lovell
IBM T. J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532
dcl@watson.ibm.com

Introduction

Our group at the IBM T. J. Watson Research center develops transaction-oriented systems which reduce the processing barriers between businesses and their customers. We do this by developing advanced, computationally active user-interfaces, reengineering business processes, and building interfaces directly to existing data processing systems.

Early in 1996 we faced the task of providing document support for a pilot project we were undertaking with Chase Manhattan Automotive Finance Corporation (hereafter "Chase") a subsidiary of The Chase Manhattan Bank. The project would connect automobile dealers to the Chase automobile loan decision system via the internet. It would give dealers a loan decision in minutes for most credit applications. It would replace a system of data entry from faxes that produced loan decision times measured in hours to days.

Dealers know that customers generally do not return to the dealership a second time. If they say they'll be back, they won't. When a customer leaves without buying a car, odds are there will be no sale. Quick financing approval for customers with good credit would allow dealers to give approved financing and sell an automobile to a customer on the crucial first visit.

In addition to an electronic credit application, the system would allow dealers to submit an electronic loan contract. The electronic contract would enable the financial institution to almost immediately provide funds, thus reducing the "float" time between the dealer's payment for the automobile and the bank's payment to the dealer.

The first lenders on the system would anticipate growing market share due to increased service to the dealer. All lenders on the system would enjoy lower costs for approving loans.

In spite of the electronic nature of this reengineering effort, lenders, dealers, and purchasers would (for the forseeable future) need printed credit applications, contracts, and other documents related to the legal terms, conditions, disclosures, and obligations understood by all parties. In particular, purchasers must drive away from the dealership in their new automobiles with a copy of the agreements they have made to pay for them.

Our Decision to use TeX

We knew we needed to print, at a minimum, credit applications and financing contracts. Given this broad requirement we began exploring in greater detail what the document printing requirements would be, what the documents look like, and what our document objectives were. We generated and analyzed several technical approaches and selected one for further development.

Characterization of the document problem Our first move was to gather as many of these documents as we could to learn what they were like. Chase was able to provide us with a sampling of their financing contracts from various states.

Figure 1 shows a representative example for New York.

We made the following observations:

1. The documents were printed front and back
2. They were standard width but lengths varied up to 28 inches, all longer than 14 inches
3. They were bound at the top with carbons (or carbonless), as white, yellow, and pink copies for the lender, purchaser, and dealer.
4. The copies were usually identical, however; in some cases the purchaser copy was missing the section used by the lender or dealer to assign the contract to another party
5. There was a large variety of type sizes and weights, but not a large variety of typefaces
6. There was a large variety of layout and graphics features– boxes, rules, brackets, columns, paragraph styles, bullets, enumerated lists at multiple levels, etc.
7. The documents were primarily printed with a single color. There was some use of shading using screens. There was rare use of a second color.

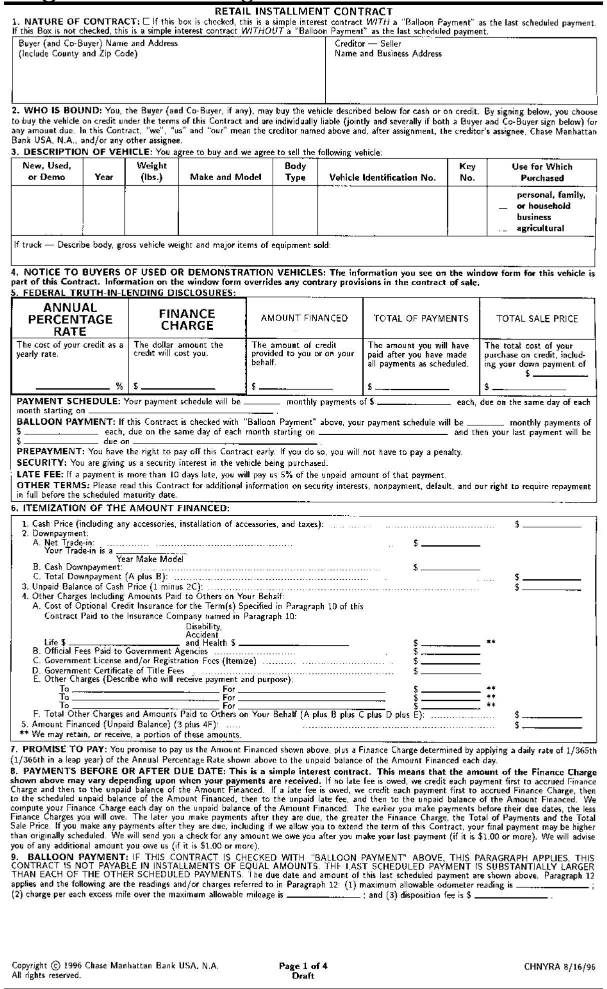


Figure 1: First Page of A Chase Retail Contract

RETAIL INSTALLMENT CONTRACT

1. NATURE OF CONTRACT: ☐ If this box is checked, this is a simple interest contract *WITH* a "Balloon Payment" as the last scheduled payment. If this Box is not checked, this is a simple interest contract *WITHOUT* a "Balloon Payment" as the last scheduled payment.

Buyer (and Co-Buyer) Name and Address (Include County and Zip Code)	Creditor — Seller Name and Business Address

2. WHO IS BOUND: You, the Buyer (and Co-Buyer, if any), may buy the vehicle described below for cash or on credit. By signing below, you choose to buy the vehicle on credit under the terms of this Contract and are individually liable (jointly and severally if both a Buyer and Co-Buyer sign below) for any amount due. In this Contract, "we", "us" and "our" mean the creditor named above and, after assignment, the creditor's assignee, Chase Manhattan Bank USA, N.A., and/or any other assignee.

3. DESCRIPTION OF VEHICLE: You agree to buy and we agree to sell the following vehicle:

New, Used, or Demo	Year	Weight (lbs.)	Make and Model	Body Type	Vehicle Identification No.	Key No.	Use for Which Purchased
							personal, family, __ or household business __ agricultural
If truck — Describe body, gross vehicle weight and major items of equipment sold:							

4. NOTICE TO BUYERS OF USED OR DEMONSTRATION VEHICLES: The information you see on the window form for this vehicle is part of this Contract. Information on the window form overrides any contrary provisions in the contract of sale.

5. FEDERAL TRUTH-IN-LENDING DISCLOSURES:

ANNUAL PERCENTAGE RATE	**FINANCE CHARGE**	AMOUNT FINANCED	TOTAL OF PAYMENTS	TOTAL SALE PRICE
The cost of your credit as a yearly rate.	The dollar amount the credit will cost you.	The amount of credit provided to you or on your behalf.	The amount you will have paid after you have made all payments as scheduled.	The total cost of your purchase on credit, including your down payment of $ ______
______ %	$ ______	$ ______	$ ______	$ ______

PAYMENT SCHEDULE: Your payment schedule will be ______ monthly payments of $ ______ each, due on the same day of each month starting on ______.

BALLOON PAYMENT: If this Contract is checked with "Balloon Payment" above, your payment schedule will be ______ monthly payments of $ ______ each, due on the same day of each month starting on ______ and then your last payment will be $ ______ due on ______.

PREPAYMENT: You have the right to pay off this Contract early. If you do so, you will not have to pay a penalty.

SECURITY: You are giving us a security interest in the vehicle being purchased.

LATE FEE: If a payment is more than 10 days late, you will pay us 5% of the unpaid amount of that payment.

OTHER TERMS: Please read this Contract for additional information on security interests, nonpayment, default, and our right to require repayment in full before the scheduled maturity date.

6. ITEMIZATION OF THE AMOUNT FINANCED:

1. Cash Price (including any accessories, installation of accessories, and taxes):		$ ______
2. Downpayment:		
A. Net Trade-in: Your Trade-in is a ______ Year Make Model	$ ______	
B. Cash Downpayment:	$ ______	
C. Total Downpayment (A plus B):		$ ______
3. Unpaid Balance of Cash Price (1 minus 2C):		$ ______
4. Other Charges Including Amounts Paid to Others on Your Behalf:		
A. Cost of Optional Credit Insurance for the Term(s) Specified in Paragraph 10 of this Contract Paid to the Insurance Company named in Paragraph 10: Life $ ______ Disability, Accident and Health $ ______	$ ______ **	
B. Official Fees Paid to Government Agencies	$ ______	
C. Government License and/or Registration Fees (Itemize)	$ ______	
D. Government Certificate of Title Fees	$ ______	
E. Other Charges (Describe who will receive payment and purpose):		
To ______ For ______	$ ______ **	
To ______ For ______	$ ______ **	
To ______ For ______	$ ______ **	
F. Total Other Charges and Amounts Paid to Others on Your Behalf (A plus B plus C plus D plus E):		$ ______
5. Amount Financed (Unpaid Balance) (3 plus 4F):		$ ______
** We may retain, or receive, a portion of these amounts.		

7. PROMISE TO PAY: You promise to pay us the Amount Financed shown above, plus a Finance Charge determined by applying a daily rate of 1/365th (1/366th in a leap year) of the Annual Percentage Rate shown above to the unpaid balance of the Amount Financed each day.

8. PAYMENTS BEFORE OR AFTER DUE DATE: This is a simple interest contract. This means that the amount of the Finance Charge shown above may vary depending upon when your payments are received. If no late fee is owed, we credit each payment first to accrued Finance Charge and then to the unpaid balance of the Amount Financed. If a late fee is owed, we credit each payment first to accrued Finance Charge, then to the scheduled unpaid balance of the Amount Financed, then to the unpaid late fee, and then to the unpaid balance of the Amount Financed. We compute your Finance Charge each day on the unpaid balance of the Amount Financed. The earlier you make payments before their due dates, the less Finance Charges you will owe. The later you make payments after they are due, the greater the Finance Charge, the Total of Payments and the Total Sale Price. If you make any payments after they are due, including if we allow you to extend the term of this Contract, your final payment may be higher than originally scheduled. We will send you a check for any amount we owe you after you make your last payment (if it is $1.00 or more). We will advise you of any additional amount you owe us (if it is $1.00 or more).

9. BALLOON PAYMENT: IF THIS CONTRACT IS CHECKED WITH "BALLOON PAYMENT" ABOVE, THIS PARAGRAPH APPLIES. THIS CONTRACT IS NOT PAYABLE IN INSTALLMENTS OF EQUAL AMOUNTS. THE LAST SCHEDULED PAYMENT IS SUBSTANTIALLY LARGER THAN EACH OF THE OTHER SCHEDULED PAYMENTS. The due date and amount of this last scheduled payment are shown above. Paragraph 12 applies and the following are the readings and/or charges referred to in Paragraph 12: (1) maximum allowable odometer reading is ______; (2) charge per each excess mile over the maximum allowable mileage is ______; and (3) disposition fee is $ ______.

Copyright © 1996 Chase Manhattan Bank USA, N.A. All rights reserved. **Page 1 of 4 Draft** CHNYRA 8/16/96

8. There was some variation in in presentation from state to state.
9. There was some variation in content from state to state; but, much of the content was the same.

At the dealerships, the documents are stored in boxes as they came from the printing company. The financing agent at the dealer puts the preprinted document into an impact printer which blindly types data at given coordinates. A successful print results when the program doing the printing has coordinates which match the form, and when the form is aligned correctly in the printer.

We learned that the business of supplying the preprinted forms and the software coordinates to print the data on them is lucrative for the printing company and expensive for the banks. The business is somewhat painful for the dealer, who has to keep all of the forms stocked. It is also wasteful of resources. The dealers and suppliers discard all of the preprinted forms whenever a bank releases a revision, or when a state or federal law changes and all of the banks release revisions.

We thought we could improve on this by printing the documents from a laser printer. The dealer could stock blank legal paper and get document revisions from us at a fraction of what the printers charge to add a new preprinted form to their form printing program.

We had to begin by redefining features of the preprinted forms that we could not reproduce on a laser printer. One of these was the format. There is no ready supply of laser printer paper in 28 inch lengths, nor are there any acceptably priced printers which will handle this. One possibility was to use a laser printer that supported a ledger format. We hoped to avoid this due to the expense of installing such a specialized printer at each dealer site. We wanted to use a standard office printer.

Does a "single document law," stating that the agreement must be one document, mean that the document must be a single sheet of paper? It turns out that multiple page contracts are acceptable to the financial institutions, provided they are numbered in a "Page n of m" format. The format of the preprinted forms as long, single sheets was a convenience for shipping, handling, and feeding into an impact printer. It ensured that the entire document was present without introducing a need for stocking multiple sets of carbon-duplicated pages.

Another feature we could not readily produce on a laser printer was the second color. A second color was used on the Chase contracts for the states of California and Virginia, which have laws requiring that specific features of the contract be highlighted. Whether the highlighting must be achieved with a second color or whether it could be addressed by some other graphic mechanism– bold and big typeface, white space surrounding, bold frames surrounding –was an open question for some time. We investigated a color ink-jet printer in case the color features could not be redesigned; but, we made a decision not to let a few two-color contracts scuttle the whole printing project.

We learned that the multiple carbons do not have to be printed on differently colored sheets. It was sufficient to print multiple copies provided we identified each copy in the footer, e.g. "dealer copy," "customer copy."

Printing objectives Given the capability to reproduce the documents on a laser printer, we produced the following requirements:

1. Provide a means to model all of the layout and graphics features of the printed contracts.
2. Provide a means to print the documents with or without data in the spaces provided for data.
3. Provide a means to re-use portions of a document in other documents. We felt this would be important for efficiency and consistency given that much of the document content was repeated in multiple documents.
4. Provide a means to display portions of a document in the user interface.
5. The process for adding data to the document and printing must run without any special user actions, other than pressing a "print" button on the user interface.
6. Find a solution that can be realized quickly at reasonable cost.

Part of the variation in the content of the financing contracts was in the disclosures and notices which appear above the signatures on the documents. One of our goals was to transmit an electronic contract, including signature, from the dealer to the bank. This implied that we might need to display part or all of the contract in the user interface – at least the disclosures – before taking the purchaser's and dealer's signature.

We believed that we should consider extracting parts of the document content for display in the user interface.

As we would print each document specifically for each financing agreement. The following additional capabilities seemed reasonable:

1. Omit (or include) specific portions of the document as applicable to the agreement.
2. Reformat paragraphs and boxes to fit the data.

The first additional capability allows us to eliminate paragraphs pertaining to options, such as "optional credit life insurance" or "balloon payment options," which are not exercised for a particular agreement.

The second solves the classic problem we've all encountered when completing forms– blank spaces too small or grossly oversized for the data we need to write into them.

Technical alternatives Having worked with LaTeX in the past, I believed that TeX could carry the project where we needed to go. Before settling on TeX, we explored a set of other strategies. We separate these strategies into two classes– overlay and template.

Overlay strategies use a static image of the form with blank spaces for data. The static image is typically obtained by scanning a paper original, or by using a preprinted form. The overlay process prints the data over the static image using a map which matches data items with locations of the blank spaces.

Overlay strategies faithfully reproduce features of the printed document. When the image quality of a faxed document is acceptable, the image file can be acceptably small. The data maps may be produced fairly quickly without much skill.

Overlay strategies fail to meet the two additional capabilities we hoped to incorporate in our system. Given a static image of the form, there is no stretching of spaces too small for the data, no shrinking of spaces too large, and no capability to eliminate unneeded paragraphs and repaginate the document.

Overlay strategies do not readily lend themselves to extraction of content for the user interface, or to re-use of content in multiple documents. An overlay has no internal structure that may be taken apart and reassembled in new ways.

When an overlay document changes the entire document must be rescanned and the data map rebuilt with new data locations. There is also a trade-off with overlays between image quality and file size. High quality text requires very large image file sizes.

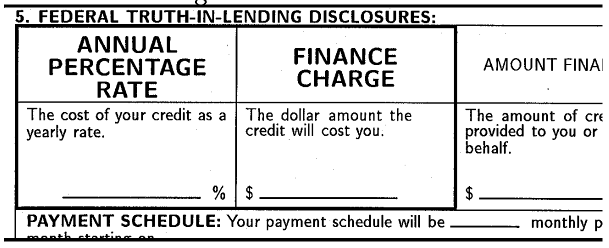
We found that the overlay strategy is acceptable for some documents which do not require high quality text, sharp lines, or reformatting. We use it to print credit applications, for example. We did not find the overlay strategy acceptable for our contract printing requirements. For those we turned to template strategies.

Template strategies represent the document as a series of layout and graphics controls with text. Place-holders in the document mark positions for the data values. A merge process that we call "data injection" replaces the data place-holders with actual data. The typesetting system reformats the document with the data in place.

A number of word processors and page composition systems have formatting and data injection capabilities; however, we eliminated these immediately because they were not designed as unattended background processes and use mostly proprietary data formats.

This left us with text markup languages which have typesetting engines on the PC platforms we run. We briefly considered SGML, but the high-level markup must still be translated into some lower-level typesetting commands, like TeX. The full generalization of SGML was more than our needs

Figure 2: Portion of the Federal Truth-In-Lending Disclosure



5. FEDERAL TRUTH-IN-LENDING DISCLOSURES:

ANNUAL PERCENTAGE RATE	**FINANCE CHARGE**	AMOUNT FINA
The cost of your credit as a yearly rate. ______ %	The dollar amount the credit will cost you. $ ______	The amount of cre provided to you or behalf. $ ______

PAYMENT SCHEDULE: Your payment schedule will be ______ monthly p
month starting on

required. While we were concerned with high-level markup, we were more immediately concerned with using a typesetting engine that could reproduce all of the graphic features of the contracts. We also had LaTeX as an example of higher-level markup implemented using TeX.

After some exploration my initial inclination to try TeX seemed justified. Feeling reasonably assured, we set out to prove to ourselves that we could build our contract printing solution using TeX as the typesetting engine.

Proof of Concept

Given a decision to use TeX, we had to show that we could reproduce the contracts in TeX and that TeX could support our need to inject data into and extract content from them. In a project like this it is important to get something working quickly, then improve upon the working prototype. We decided on a two phase campaign to prove our template printing approach using TeX. First we would produce the contract document needed for the Chase pilot using TeX markup. Second, we would implement a small program to inject data into the document.

The first document The task of creating the first document was assigned to me. Prior to this project I had used LaTeX to typeset papers and letters. The contract documents we had to duplicate presented a new set of challenges. Knowing that the LaTeX standard formats were not appropriate for these documents, I began immediately using plain TeX.

I began with the basic page layout and a paragraph of text, then tackled the parts of the documents that had a large number of rules and alignments. We determined that these were the most difficult portions of the documents to reproduce and chose to work on these first to demonstrate that TeX could reproduce them.

The section which gave the most difficulty was the "Federal Truth-In-Lending Disclosures" (see fig. 2).

The challenging features were center-justified text, text centered vertically as well as horizontally, paragraphs within columns, and rules which changed weight going across the page. I had to fully understand the way TeX builds boxes in its various modes and how to control an `\halign`.

Learning TeX was a great deal of fun. The TeX layout model of boxes, glue, and springs was a delight to learn. Most confusion came from understanding the modes– the difference between vertical, horizontal, and restricted horizontal modes– and more, when the transitions occur between modes. Controlling vertical space on the transition from horizontal back to vertical mode was a puzzle.

It took two weeks to code the first page of the contract (fig. 1) given a study of the first few chapters of Knuth's TeXBook (Knuth, 1994), and using the task to direct further reading and experimentation. Most of the rest of the contract was text supplied by a fast typist. We were now well satisfied that we could faithfully reproduce the contract using TeX.

Data injection The second task was to develop a method to get our data values into the document. Our data comes to the document printing subsystem as a stream of characters which we parse into named "tables" and "lists."

A table contains a set of named "fields," each of which has a value. A list contains an array of such sets. We refer to a table field using the name of the table and the name of the field, e.g. "AutoContract.CashPrice." We refer to a list field by adding an index, e.g. "AutoOtherCharges[0].Amount"

We have a set of functions written in the C programming language which return values from the data stream given table, list, and field names.

We initially develped data macros to support table values only. We designed the first macro, "`\DataTarget`" with three arguments'

1. Table name – the name of the table,
2. Field name – the name of the field from which to take the value, and
3. Default width – the size of the blank to leave when there is no value.

This macro output an `\hrule` on the baseline using the given width. It completely ignored the first two arguments. This provided a reasonable default behavior which enabled us to process the documents without any data present and produce a usable blank document.

To get data into the document I wrote, in C, a simple text filter which accepted a document file and a data file. It output a copy of the document file with data in place. The filter passed all characters until encountering the string, "`\DataTarget`." It then parsed the three arguments– table, field, and width. The filter used the first two arguments to find a data value. If there was a data value it output the value inside of a macro, "`\DataValue`." If there was no value, the filter output the width inside of a macro, "`\DataBlank`."

The `\DataValue` and `\DataBlank` macros could format their arguments in any way we chose. We chose to format the value of `\DataValue` in the `\tt` font. `\DataBlank` output an `\hrule` on the baseline with width given in the third parameter.

The preprocessing filter did its job very satisfactorily. We had proved TEX and our data injection process to be a viable, capable solution to our document printing needs. We had not yet tested reading a portion of a document or displaying a portion in the user interface; however, given that the TEX sources were all ASCII text files and given our experience with the preprocessor, we believed there would be no serious obstacles to implementing those functions.

Pilot

The user interface for gathering credit application data and the electronic submission and approval process was well under way before we began work on printing contract documents. We were able to integrate the Chase contract into the system before giving a preview of the system, with Chase, to the National Automobile Dealers' Association (NADA) convention in February 1996.

The then unnamed IBM AutoLoan Exchange system created a small sensation at NADA '96. Dealers and financial institutions were excited about what we had done and were anxious to take part. It was clear that we had made a significant advance with high impact and significant benefit in an area that had been neglected by technology providers. The industry response demonstrated considerable promise for our system.

The success was *not* primarily due to documents. Dealers were impressed that the system could give quick loan approvals and promised to close financing contracts and enable rapid delivery of payment for the cars they sold. It didn't hurt that we could produce a completely filled-in contract from a laser printer– one more thing they hadn't seen done before.

What the success meant for documents was that the document portion of the system had to grow fast. We had to scale it to multiple states and multiple products. Given a successful pilot project with Chase, we would need to produce documents for multiple financial institutions as well. We had to produce hundreds of documents. We needed to build a document creation team.

The first problem was finding people skilled with TEX. It appears that TEX is primarily used by professionals as a tool for doing their other work, or publishing their work. These are not people who are available to write automobile finance contracts.

Another niche for TEX is in the scientific and technical publishing business. We did identify a few people from this community who bid to do the work.

It was amusing to try to explain needed skills to representatives from personnel agencies. None had heard of "tech." Giving the spelling and saying "tex" as in "Texas" helped some; but, searches in their databases for the keyword "tex" yielded few potential candidates. Queries for markup skills– SGML or GML –met with little better success.

Fortunately, one vendor was willing to dig deeper to find the skills we needed. He began with the listings in the back of *TUGboat*, and with a reference supplied from an email inquiry to TEX Users Group. He did the phone calling and followed leads to identify people who could code the documents and experts who could write the macros.

The backgrounds of the people who worked on our documents is indicative of the problem of finding skilled document production help for TEX. They were:

1. a Physicist turned TEX expert,
2. a Crystolographer turned IS department manager,
3. a former secretary to a large university mathematics department, and
4. me, a computer programmer.

In addition, while we preferred to find people who could come to work with us on-site, it became clear that we would have to settle for a distributed, electronic, virtual workplace. We coordinated by telephone and shared files by ftp transfer. I had to act as administrator and librarian to keep synchronized the hundreds of files we generated.

Many of the TEX experts who bid for the macro work focused on the data injection portion of the problem, perhaps because we had explained this most clearly. In fact, this was the problem we felt most confidently we had solved.

The part of the problem less clearly expressed, but of greatest concern to us was the problem of efficiently producing the full, large variation in page layouts and typographic features of automobile finance contracts. We felt that a flexible, extensible, high-level markup was needed to meet our goals of

1. having administrative level document production staff quickly produce fifty to seventy documents for a new financial institution,
2. faithfully reproducing the style and content of those documents, and
3. quickly tooling for a new style by reusing as much as possible the work done to produce previous styles for previous financial institutions.

The TEX expert we selected understood this problem and directly addressed it with a syntax for extensible markup and a framework for TEX macro development and reuse (Ogawa, 1994; Baxter, 1994).

Rollout

At this time we have complete contracts for most of the fifty states for Chase, and a few dozen states' documents for three other financial institutions. Chase has begun using the documents for actual, legally binding financing contracts at dealerships using the ALX system.

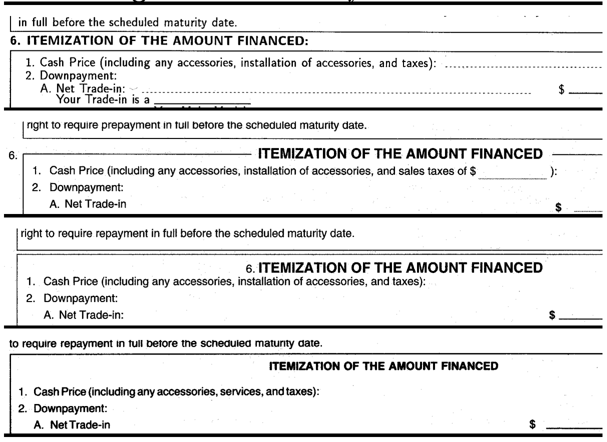
The help we got from an expert TEX "insider" was crucial to our success with these documents. He was able to code some of the features which were giving us a hard time, such as page numbering and footers, section cross-references, and two-column layout. More importantly, he was able to capture features as separate elements which could be applied repeatedly as needed in the many documents we produced. He was also able to develop content-level commands we used to code the structure of the document. He configured the appearance of these for each document style. Where the appearance varied within a document, he provided attributes we could code to guide the presentation.

Consider the case with sections. The presentation of the section number and title varies from document to document. Figure 3 shows examples.

Those of us coding the documents wrote a `\section` command with a `title` and `label` attribute. Where there was a box around the section we could specify `framing`. The document style took care of the font change of the title position of the title, and position of the text relative to the section number.

All of this meant that the document coders could focus on producing the content of the documents using fairly high-level markup. The details of presentation, the hard and messy typesetting considerations, were taken care of by lower-level

Figure 3: Section style variations


in full before the scheduled maturity date.

6. ITEMIZATION OF THE AMOUNT FINANCED:

1. Cash Price (including any accessories, installation of accessories, and taxes): ....
2. Downpayment:
   A. Net Trade-in: .... $ ____
   Your Trade-in is a ____________

right to require prepayment in full before the scheduled maturity date.

6. **ITEMIZATION OF THE AMOUNT FINANCED**

1. Cash Price (including any accessories, installation of accessories, and sales taxes of $ ________):
2. Downpayment:
   A. Net Trade-in $ ____

right to require repayment in full before the scheduled maturity date.

6. **ITEMIZATION OF THE AMOUNT FINANCED**

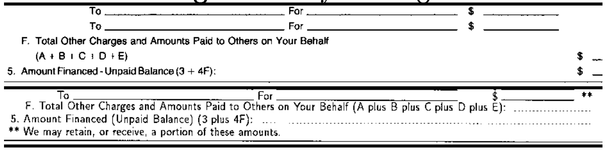
1. Cash Price (including any accessories, installation of accessories, and taxes):
2. Downpayment:
   A. Net Trade-in: $ ______

to require repayment in full before the scheduled maturity date.

ITEMIZATION OF THE AMOUNT FINANCED

1. Cash Price (including any accessories, services, and taxes):
2. Downpayment:
   A. Net Trade-in $ ______

Figure 4: Style change


To ____________ For ____________ $ ________

To ____________ For ____________ $ ________

F. Total Other Charges and Amounts Paid to Others on Your Behalf
(A + B + C + D + E) $ _

5. Amount Financed - Unpaid Balance (3 + 4F): $ _

To ____________ For ____________ $ ________ **

F. Total Other Charges and Amounts Paid to Others on Your Behalf (A plus B plus C plus D plus E): ....

5. Amount Financed (Unpaid Balance) (3 plus 4F): ....

** We may retain, or receive, a portion of these amounts.

presentation code. This allowed us to specify a large lead time for first-of-a-kind documents and realize a short turn-around time for n'th-of-a-kind documents.

The only drawback was that I no longer well understood the formatting code. We became reliant on the consultant. When we wanted a change quickly we had to appeal to him to give our work priority, or resort to low-level commands to implement the change ourselves, locally in the document.

One example occurred when a client added a footnote within an itemization section (see fig. 4).

The consultant was not able to implement this immediately; so, we had to revert the `\item` markup to an `\halign`.

We believe that, as we do more and more documents, fewer and fewer of these new typesetting needs will arise. As we add each new feature to our toolkit we will be able to produce new document styles with less new coding effort. We have had some encouragement in this belief in our experience with adding new financial institutions. The second and third financial institution required considerably less work due to the large amount of code reused from the Chase pilot.

Conclusion

The IBM AutoLoan Exchange has delivered on much of its promise. We now deliver rapid loan decisions and contract funding for dealers subscribed to the system through Chase. with many other finance institutions and dealers to follow. The application has defined a new direction for the business of automobile financing unforseen by existing vendors in the field.

The document capabilities of the system form a small, but essential part of the overall success of the IBM AutoLoan Exchange in defining a new paradigm for the business of making and closing automobile financing. Out go truck-loads of preprinted, carbonless forms in quadruplicate. In come laser printers and reams of fresh, blank, legal-sized laser printer paper.

Dealers print only the documents they need, when they need them, complete with the data values for each particular financing deal. When a form changes there are no bundles of preprinted, obsolete forms to discard.

Our world is a networked, distributed world of electronic commerce where complex legal documents are delivered from a server and customized by a client at the point of delivery. We have shown that TEX can provide the capability needed to typeset these documents transparently, without recourse to a page-layout or word-processing system.

The TEX typesetting system was vital to the success of the printing subsystem of the IBM AutoLoan Exchange. TEX can play a vital role wherever custom legal documents are needed in the world of the Net.

Acknowledgements

Thank you to Stephen Boies and Lauretta Jones, visionary leaders of the IBM AutoLoan Exchange project.

Thank you to Arthur Ogawa and his many helpers in the TEX community who provided the advanced TEXnology to make the IBM AutoLoan Exchange documents really work.

Thank you to Alan Bednowitz and Catherine McGinnis for their detailed and dedicated work to produce nearly one hundred contracts to date for the IBM AutoLoan Exchange.

References

William Erik Baxter An object-oriented programming system in TEX *TUGboat*, 15(3):331–338, September 1994

Knuth, Donald E. *The TEXbook.* Addison Wesley, Reading, Massachusetts, 1994.

Ogawa, Arthur. "Object-oriented programming, descriptive markup, and TEX". *TUGboat* **15**(3), 325–330, 1994.

Breaking equations

Michael Downes
American Mathematical Society
PO Box 6248
Providence, RI 02940
USA
mjd@ams.org

Introduction

Some of the inconvenient aspects of writing displayed equations in TeX are of such long standing that they are scarcely noticed any more except by beginning users. For example, if an equation must be broken into more than one line, `\left` ... `\right` constructs cannot span lines. This is a report on a new LaTeX package called `breqn` that substantially eliminates many of the most significant problems (described at length in the next section). Its main goal is to support automatic linebreaking of displayed equations, to the extent possible within the current limitations of TeX and LaTeX. Such linebreaking cannot be done, however, without substantial changes under the hood in the way math formulas are processed. Some of the changes are radical enough that it would be more natural to do them in LaTeX3 than in LaTeX2e — e.g., for LaTeX3 there is a standing proposal to have nearly all nonalphanumeric characters be active by default; having `^` and `_` active this way would have eased some implementation problems. Using the package in LaTeX2e is possible, with some extra care.

Current shortcomings in LaTeX equation handling

Hindrances for authors The following difficulties affect authors using the standard LaTeX `equation` and `eqnarray` environments. Some of them are ameliorated by the use of the `amsmath` package. (The first four also apply for plain TeX; and the main reason the next three don't apply as well is that plain TeX replaces them with a more substantial shortcoming: no automatic numbering at all.)

1. Line breaks must be inserted by hand.
2. Breaks are sensitive to changes in fonts or column width; and altering them is onerous.
3. A break within `\left`-`\right` delimiters requires extra work, especially if there is any difficulty getting the sizes to match.
4. Use of `\halign` freezes available shrink. Thus, for example, suppose that a given formula fits within the column when done with an `equation` environment; the exact same formula may fail to fit when done with an `eqnarray` environment, because `eqnarray` uses `\halign` internally.
5. Punctuation at the end of an equation logically belongs with the surrounding text but it must be entered with the body of the equation in order to print in the right place. This discord is especially noticeable when promoting formulas from inline math to display.
6. A numbered equation that takes several lines in an `eqnarray` requires awkward use of `\nonumber` to keep from getting a number on each line.
7. Numbers may overlap the equation body without warning (in `eqnarray` and similar structures).
8. There is no easy way to specify a variant equation number for an individual equation.
9. The space around equal signs in `eqnarray` is noticeably larger than the normal spacing for such symbols. This looks bad when adjacent equations are done one with `equation` and one with `eqnarray`.
10. There is no easy way to center an equation number across multiple lines of a broken equation. Some users manage to infer that `array` is the natural approach for this, but a plain `array` has various spacing faults for this purpose, and uses text style instead of display style for the contents.
11. There is no easy way to add a frame around the body of an equation (with or without including the equation number). You can just about do it with a one-line equation if there's no number and if you know about `\displaystyle`. But with multiline equations it's rather more difficult (use of `array` is again indicated, but it brings all the deficiencies cited in the preceding item).

The bosonic part of the action takes the form

$$I = I_{00} + I_{01} + I_{10} + \cdots \tag{14}$$

where

$$\begin{aligned} I_{00} \quad = \quad & \frac{(2\pi)^3}{\alpha'^2} \int d^6x\sqrt{-G}e^{-\Phi} \left[R_G + G^{MN}\partial_M\Phi\partial_N\Phi\right. \\ & \left.-\frac{1}{12}G^{MQ}G^{NR}G^{PS}H_{MNP}H_{QRS}\right] \end{aligned} \tag{15}$$

where $M, N = 0, ..., 5$ are spacetime indices.

```
\begin{eqnarray}\nonumber
I_{00}&=& \frac{(2\pi)^3}{\alpha^{\prime 2}}
  \int d^6x \sqrt{-G}e^{-\Phi} \left[R_G+G^{MN}
  \partial_M\Phi\partial_N\Phi\right.\\
&& \left.-\frac{1}{12}G^{MQ}G^{NR}G^{PS}H_{MNP}H_{QRS}\right]
\end{eqnarray}
```

Figure 1: Typical equation problems in ordinary LaTeX: (a) different spacing around the equals signs in (14) and (15) because one uses `equation` and the other uses `eqnarray`; (b) equation (15) is a single equation but because it covers two lines `\nonumber` must be used on the first line; (c) and then the number is not vertically centered on the entire equation; (d) the sizes of `\left [` in the first line and `\right ]` in the second line don't match (they could be made to match, with extra work); and (e) the minus sign at the beginning of the second line is getting (wrong) unary spacing. This example is from (Duff, Minasian, and Witten, 1996), with only a couple of minor adaptations.

Issues of typeset quality

1. Symbol spacing tends to go wrong at the start of continuation lines (cf (Kopka and Daly, 1995, §5.4, p 136)). When a line break is taken before a binary infix operator, the operator will typically get unary operator spacing, though it shouldn't. (See Figure 1.)
2. Use of `\halign` (as in `eqnarray`) keeps the display short spaces from ever being applied, even when a group of equations begins with a short equation that would get the reduced spacing if it occurred by itself.
3. No distinction is made between consecutive, separate equations and lines of a single, broken equation.
4. Standard methods for reducing the type size of an individual equation all have adverse side effects; typically, the wrong line-spacing gets used for the text preceding the equation.
5. When a multiline block of text is displayed and numbered like a formula, the base-to-base spacing above and below doesn't work quite right.

Features and misfeatures of the `amsmath` package

As compared with the standard LaTeX facilities for equations, the `amsmath` package addresses some of the problems mentioned above, but introduces a few new misfeatures of its own.

Features

1. The `split` and `multline` structures match up better with the logical structure of individual equations and equation groups.
2. The multiple-equation environments `align`, `gather`, etc., use the correct spacing for equal signs.
3. The `\tag` command makes it easy to get variant equation numbers.
4. Overlap of the equation number on the equation body is mostly prevented.
5. There is more control over page breaking.
6. Environments `aligned`, `cases`, etc., can be used as building blocks in building up more complicated displays.

Misfeatures

1. For technical reasons, abbreviations like `\bal`, `\eal` for `\begin{align}`, `\end{align}` don't work.
2. There are inconsistencies between the `multline` and `split` environments; for example, the equation number for `multline` does not get centered the way it does for `split`.
3. The `equation` environment is implemented as a subcase of the `gather` environment, which means that it inherits the `\halign` deficiencies mentioned above: horizontal shrink isn't used; the short skip possibility is disregarded; and

it also is rendered unabbreviable, as described above. (Although work-arounds exist, they aren't particularly well known and deviate from canonical LaTeX syntax.)

TeXnical difficulties Looking at the above lists of deficiencies, one may well wonder why they have not been better addressed before now, more than ten years after LaTeX (and AMS-TeX) were first developed (1983–1985). One of the contributing reasons, however, is surely the intransigence of the TeXnical difficulties involved.

- TeX lacks low-level support for typical display-breaking conventions; for example, break penalties are provided only on the right side of mathbin and mathrel symbols.
- Math/text defaults for `$$` and `\eqno` are backwards. If TeX's display structure had been envisioned as a purely typographical device and started out in text mode rather than in math mode, a number of difficulties would never arise. The same can be said for `\eqno`. Thus providing a simple way such as `$$` to start a math display would have been better relegated to the macro level, not hardwired into the primitive display mechanism.
- `\left`–`\right` subformulas are wrapped in an unbreakable box.
- Displayed equation macros have been mainly written towards the *typographical* structure embodied in TeX's `$$` mechanisms, instead of towards the actual logical structure of the material (distinguishing single equations from equation groups, intra-equation punctuation from external punctuation and so on).

Features of the `breqn` package

- ☐ Overlong equations can be broken automatically to the prevailing column width following standard conventions. There will always be some equations that need special line-breaking attention from the author, but for those that don't, the process is highly automated, including standard indention conventions, avoiding overlap with the equation number, and so on.
- ☐ Line breaks can be specified in a natural way even within `\left` ... `\right` delimiters. Preferred but nonmandatory breakpoints can be specified within equations by `\linebreak` with an optional argument, as usual.
- ☐ Separate equations in a group of equations are written as separate environments instead of being bounded merely by `\\` commands. This simple change dispels, as a side effect, the problem of wrong math symbol spacing at the beginning of continuation lines.
- ☐ Horizontal shrink is made use of whenever feasible (most other equation macros are unable to get at it when it occurs between `\left` ... `\right` delimiters or in any sort of multiline structure). (However, shrinkable space inside fractions, square roots, overlined quantities, etc., is not unfrozen by this package. That is a less tractable problem.)
- ☐ The `\abovedisplayshortskip` is used when applicable (other equation macros fail to apply it in equations of more than one line).
- ☐ Displayed 'equations' that contain mixed math and text, or even text only, are handled naturally by means of a `dtext` environment that starts out in text mode instead of math mode.
- ☐ The punctuation at the end of a displayed equation can be handled in a natural way that makes it easier to promote or demote formulas from/to inline math, and to apply special effects such as omitting the punctuation, as favored by some of the more progressive book designers.
- ☐ Equation numbering is handled in a natural way, with all the flexibility of the `amsmath` package (features like `\tag` and `subequations` are provided under different guises) and with no need for a special `\nonumber` command.
- ☐ Unlike the `amsmath` equation environments, the `breqn` environments can be called through user-defined abbreviations such as `\beq` ... `\eeq`.
- ☐ It is easy to set local options for a single equation environment, e.g., changing the type size or adding a frame.
- ☐ It is possible to specify different vertical space values for the space between lines of a long, broken equation and the space between separate equations in a group of equations.
- ☐ Page breaking before, within, and after displayed math formulas is subject to more sophisticated control than it is with other extant equation macros.
- ☐ A rather noteworthy 'feature': as it pushes the envelope of what is possible within the context of LaTeX2e, the `breqn` package will tend to break other packages when used in combination with them, or to fail itself, when there are any areas of internal overlap; successful use may in some cases depend on package loading order.

Environments and commands All of the following environments take an optional argument for applying local effects such as changing the typesize or adding a frame to an individual equation.

`dmath` Like `equation` but supports line breaking and variant numbers.

`dmath*` Unnumbered; like `displaymath` but supports line breaking

`dtext` Like `equation` but starts out in text mode

`dtext*` Unnumbered variant of `dtext`

`dgroup` Like the `align` environment of `amsmath`, but with each constituent equation wrapped in a `dmath`, `dmath*`, `dtext`, or `dtext*` environment instead of being separated by `\\`. The equations are numbered with a group number. When the constituent environments are the numbered forms (`dmath` or `dtext`) they automatically switch to 'subequations'-style numbering, i.e., something like (3a), (3b), (3c), ..., depending on the current form of non-grouped equation numbers. See also `dgroup*`.

`dgroup*` Unnumbered variant of `dgroup`. If the constituent environments are the numbered forms, they get normal individual equation numbers, i.e., something like (3), (4), (5), ... Numbered and unnumbered forms can be mixed in the natural way, as needed.

`darray` Similar to `eqnarray` but with an argument like `array` for giving column specs. Automatic line breaking is not done here.

`darray*` Unnumbered variant of `darray`, rather like `array` except in using `\displaystyle` for all column entries.

Restrictions on the use of the `breqn` package

math symbol setup In order for automatic line breaking to work, the operation of all the math symbols of class 2, 3, 4, and 5 must be altered (relations, binary operators, opening delimiters, closing delimiters). This is done by an auxiliary package `flexisym`. If you use anything other than the standard LaTeX set of math symbols from the fonts `cmex`, `cmsy`, `cmmi`, you will probably need to do some configuration of the load process for the `flexisym` package.

subscripts and superscripts Because of the changes to math symbols of class 2–5, writing certain combinations such as `^+` or `_\pm` or `^\geq` without braces will lead to error messages; in general, except for letters and digits, any single math symbol must be enclosed in braces when sub or superscripted: `^{+}`, `_{\pm}`, `^{\geq}`. Actually, there is no visible sanction in the LaTeX book for omitting such braces: it uniformly shows braces in all sub and superscript examples — perhaps because the problem described here already exists in standard LaTeX to a lesser extent, as you may know if you ever tried `^\neq` or `^\cong` (in the case of `\neq` the symbol simply prints incorrectly, *without* giving an error message).

Grinchuk (Grinchuk, 1996) encountered the same sort of technical complications regarding braces around superscripted math symbols in his efforts to support Russian-style formula breaks as I did, and thinks (as I do) that turning the `^` and `_` characters into active characters might be the best user-friendly solution.

Breaking long equations

Let's begin by reviewing some first principles. Some facets of the typesetting task for displayed equations are so 'self-evident' that they rarely receive any explicit attention, but only by paying explicit attention to everything can we be sure of getting a strong grasp of the whole picture, and of the relative significance of various constraints.

Displayed mathematical expressions A *displayed mathematical expression*, commonly referred to as a *displayed formula* or *displayed equation*, is a sentence fragment whose nucleus is some sort of math formula — not necessarily containing an actual equals sign — and that stands by itself in the text column with extra space above and below. The purpose of treating the expression this way might be

- to emphasize it
- to facilitate reference to it
- to suggest its structure by the way the lines are broken and indented, or
- simply to avoid breaking it, if it contains no acceptable break points and leaving it in-line cannot be done without adversely affecting the inter-word spacing of the text.

On the subject of breaking equations, there is a statement in *The TeXbook* (Knuth, 1986, Chapter 19, p 195) suggesting (with all the weight of Knuth's mathematical typesetting knowledge behind it) that attempting to do it automatically would be, er, well, foolish:

> It's quite an art to decide how to break
> long displayed formulas into several lines;
> TeX never attempts to break them, because
> no set of rules is really adequate. The
> author of a mathematical manuscript is
> generally the best judge of what to do, since

> break positions depend on subtle factors of mathematical exposition. For example, it is often desirable to emphasize some of the symmetry or other structure that underlies a formula, and such things require a solid understanding of exactly what is going on in that formula.

My motivation, however, for seeking to provide automatic linebreaking in the `breqn` package is the observation that among the 10% or so of equations that need to be broken, two-thirds have entirely conventional breaks, and many of the remaining can be handled nicely by having the author indicate preferred line breaks instead of forcing line breaks. And indeed, the continuation after the above quote is less pessimistic:

> Nevertheless, it is possible to state a few rules of thumb about how to deal with long formulas in displays

Fitting a displayed equation into the available text width In the following examples the gray blocks demarcate the nominal column width within which the displayed expression must fit.

Example 1. A substantial majority of equations fit comfortably within the available width, even in relatively complex mathematical material.

$$a = b + c$$

Example 2. As an equation grows longer it begins to approach the point where not all the material will fit on a single line.

$$a = b + c + d + e + f$$

Example 3. If the equation is just slightly wider than will fit, the preferred strategy is to squeeze it a little by reducing the space around binary operator symbols such as $+$, $-$, or $\otimes$.

$$a = b{+}c{+}d{+}e{+}f{+}g$$

Compare the spaces around the plus signs.

You might think that reducing the space around relation symbols would be a good idea as well; however, the default math space settings in LaTeX do not have any shrinkability in the space for relation symbols.

Example 4. Shrinkable spaces inside a subscript, superscript, fraction, root, or other special construct are frozen and cannot shrink. There is a simple reason for this: TeX cannot print any sort of object that breaks out of the basic linear symbol stream into two dimensions, except by putting it into a vbox or by raising or lowering a box from the current baseline. In either case, the material winds up inside a box, and boxes have a fixed width determined at the time of construction. A fraction consists of a numerator box stacked atop a denominator box. Thus spaces within the numerator and denominator don't shrink, even when that might be useful to help an expression fit on a single line. The following equation contains no more material, horizontally speaking, than the previous one; it is the lack of shrinking that makes it overrun the right margin.

$$a = \frac{b + c + d + e + f + g}{2}$$

Example 5. In subscripts and superscripts mathrel and mathbin spaces are entirely *omitted*! So 'to shrink or not to shrink' isn't even a question.

$$a = x_{b+c+d+e+f+g} - 2$$

When shrinking isn't enough

Example 6. Suppose that an equation still exceeds the available width after all available shrink capacity has been used up. Then the obvious way to deal with the over-wide condition is to break off the tail end of the equation and move it down to the next line. Commonly the break is chosen to fall just before a binary operator such as $+$ or $-$.

$$\begin{aligned} a = b + c + d + e\\ + f + g \end{aligned}$$

In some publishing traditions, the binary operator symbol is repeated before and after the break (Grinchuk, 1996):

$$\begin{aligned} a = b + c + d + e +\\ +f + g \end{aligned}$$

The chief feature of the `breqn` package is that long equations will break automatically at the conventional places and the continuation lines will be indented by the conventional amounts (configurable). The general idea is to set the equation as a special sort of TeX paragraph, very much like a list item (with ragged-right, rather than justified, text). Compare the outlines of

1. A typical list item, with the text set ragged right instead of justified.

and

$$\begin{aligned} \sigma(2^{34} - 1, 2^{35}, 1) = -3 + (2^{34} - 1)/2^{35} + \cdots\\ + 7/2^{35}(2^{34} - 1) \cdots \end{aligned}$$

Example 7. Breaking an equation anywhere between a pair of delimiters is usually less desirable, by an order of magnitude, than breaking elsewhere. Bad:

$$\begin{aligned} a = b + (c + d) + (e \\ + f) + g \end{aligned}$$

Better:

$$\begin{aligned} a &= b + (c + d) \\ &\quad + (e + f) + g \end{aligned}$$

But TEX does not provide any native way to test whether a given point falls between delimiters or not. So between-delimiter breaks can be more highly discouraged only if the delimiter symbols are themselves programmed to support such a test. This is in fact what the `breqn` package does.

Example 8. By giving suitable LATEX definitions to delimiter symbols such as `(`, `)`, `\langle`, `\rangle`, etc., it is possible to suppress line breaks within paired delimiters. With vert bars, however, there is a bit of a problem. The standard method of entering vert bar symbols in a document is just to use the ASCII vert bar character from the keyboard. When the verts are paired, mathematicians have little trouble telling which ones are openers and which ones are closers; but computer software such as LATEX doesn't have the discriminatory powers of a human mathematician. It is difficult, for example, for LATEX to discern that a line break such as the following is undesirable:

```
a = b +|c +d| +|e +f| +g
```

$$\begin{aligned} a = b + |c + d| + |e \\ + f| + g \end{aligned}$$

Example 9. Without any explicit indication of directionality, we might envision programming some sort of heuristic choice mechanism into the TEX definition of the vert bar, such as 'the first vert that comes along is an opener; the next one is a closer, and alternate thereafter'. Unfortunately, there are no heuristics for this particular problem that are actually good enough in practice.

If we have directional vert bar symbol commands `\lvert`, `\rvert`, `\lVert`, `\rVert`, we could rewrite the earlier example as follows, allowing LATEX to easily tell where line breaks should be discouraged more strongly.

```
a = b +\lvert c +d\rvert
    +\lvert e +f\rvert +g
```

$$\begin{aligned} a &= b + |c + d| \\ &\quad + |e + f| + g \end{aligned}$$

Even better is for paired use of the vert bars to be encapsulated in a macro with some meaningful name chosen by the author:

```
\newcommand\abs[1]{\lvert#1\rvert}
```

Both the `amsmath` package and the `breqn` package (via `flexisym`) provide `\lvert` etc.

Example 10. But then consider the following example. Which break looks better: This one?

$$\begin{aligned} a &= b \\ &\quad + (c + d + e + f + g) \end{aligned}$$

Or this one?

$$\begin{aligned} a = b + (c + d + e \\ + f + g) \end{aligned}$$

Example 11. For completeness I should have pointed out an intermediate form with the second line indented to the usual position:

$$\begin{aligned} a &= b + (c + d + e \\ &\quad + f + g) \end{aligned}$$

This illustrates the point that when a line break is taken between delimiters, we generally prefer to have the delimiters clear their contents: all material within the scope of an opening delimiter should be indented at least as much as the delimiter itself, and closing delimiters should fall further to the right than the rightmost point of the material they encompass.

Example 12. Some displayed mathematical expressions don't include any relation symbol. Then the usual practice is to indent subsequent lines by a fixed amount, say one or two quads:

$$\begin{aligned} & a + b + c \\ &\qquad + e + f + g \end{aligned}$$

Example 13. An alternative indention strategy that may be used for two-line equations, especially when the first line contains no natural alignment point, is to start the first line at (near) the left margin and end the last line at (near) the right margin. In the `amsmath` package this functionality is provided by the `multline` environment (Downes, 1995). This layout may seem more well-balanced than the other if the second line is much shorter than the first.

$$\begin{multline} a + b + c + d + e \\ + f + g \end{multline}$$

Example 14. Even if a relation symbol is present, the distribution of material between the left-hand side and the right-hand side might make a break in the former more reasonable than a break in the latter. Compare

$$\begin{aligned} (a + b + c + d + e + f + g) \\ = \alpha \end{aligned}$$

and

$$\begin{aligned}&(a+b+c+d\\&\qquad+e+f+g)=\alpha\end{aligned}$$

Example 15. A displayed formula may have the form 'A and B':

```
Y_n=Y_k\quad\text{and}\quad Z_n=Z_k
```

$$Y_n = Y_k \quad \text{and} \quad Z_n = Z_k$$

This is a subcase of a more general list pattern that may take various other forms:

$$Y_n = Y_k, \qquad Z_n = Z_k$$
$$X_n = X_k, \quad Y_n = Y_k, \quad \text{and} \quad Z_n = Z_k$$
$$X_n = X_k, \qquad Y_n = Y_k, \qquad Z_n = Z_k$$

Example 16. It is not uncommon for a displayed equation to have an attached condition or qualifier (Knuth, 1986, Chapter 19, p 185):

$$Z_n = X_n \bmod q \qquad \text{for all } n \geq 0.$$

In most cases, if there is not enough room for everything to fit on one line, breaking between the assertion and the condition is better than breaking elsewhere. Compare

$$\begin{gathered}Z_n = X_{1n} + \cdots + X_{kn} \bmod q\\ \text{for all } n \geq 0.\end{gathered}$$

and

$$\begin{aligned}&Z_n = X_{1n} + \cdots\\&\qquad + X_{kn} \bmod q \quad \text{for all } n \geq 0.\end{aligned}$$

The breqn package provides a \condition command for such material:

```
\begin{dmath*}
Z_n=X_{1n} +\cdots+X_{kn} \bmod q
\condition[]{for all $n\geq 0$}
\end{dmath*}.
```

By default a comma is added by the \condition command, but that can be overridden by an optional argument (in this example, empty).

Example 17. In a similar but rarer construction, the extra material on a line is some sort of annotation rather than a condition — i.e., it is not essential to the truth of the assertion.

$$Z_n = X_n \bmod q \qquad \text{(cf. [KF79])}$$

Algorithm for breaking equations The algorithm used by the breqn package for finding the optimal arrangement of an equation is necessarily rather complex. A portion of it is shown in Figure 2.

Some examples

Now consider Equation Example A. It doesn't quite fit in *The TEXbook*'s column width. Knuth suggests

```
$$\eqalign{\sigma(2^{34}-1,2^{35},1)
  &=-3+(2^{34}-1)/2^{35}
    +2^{35}\!/(2^{34}-1)\cr
  &\qquad+7/2^{35}(2^{34}-1)
    -\sigma(2^{35},2^{34}-1,1).\cr}$$
```

with the comment 'The idea is to treat a long one-line formula as a two-line formula, using \qquad on the second line so that the second part of the formula appears well to the right of the = sign on the first line.'

With the amsmath package the treatment would be nearly identical, but using the split environment instead of \eqalign:

```
\begin{equation*}\begin{split}
\sigma ...
    ... ,1).
\end{split}\end{equation*}
```

With the breqn package it's all automatic: the contents of the equation are written exactly the same as they would be if no line break were needed — no ampersands, no qquads, no break-here commands:

```
\begin{dmath*}
\sigma(2^{34}-1,2^{35},1)
  =-3+(2^{34}-1)/2^{35}+2^{35}\!/(2^{34}-1)
   +7/2^{35}(2^{34}-1)
   -\sigma(2^{35},2^{34}-1,1)
\end{dmath*}.
```

The first relation symbol is taken to indicate the primary alignment point, and if the total width is greater than column width, extra lines will be aligned and indented according to the class of symbol that immediately follows the break.

In a similar example (Knuth, 1986, Exercise 19.17), the point of the exercise is adding the equation number. In plain TEX this requires switching from \eqalign to \eqalignno and taking a bit of extra care with the horizontal spacing (the \quad in the last line).

```
$$\eqalignno{x_nu_1+\cdots+x_{n+t-1}u_t
   &=x_nu_1+(ax_n+c)u_2+\cdots\cr
   &\qquad+\bigl(a^{t-1}x_n+c(a^{t-2}
    +\cdots+1)\bigr)u_t\cr
   &=(u_1+au_2+\cdots+a^{t-1}u_t)x_n
    +h(u_1,\ldots,u_t). \quad&(47)\cr}$$
```

With the breqn package, the equation number is automatically placed at the location decreed by the documentclass (left, right, top, middle, bottom) and the dmath environment contains only the body of the

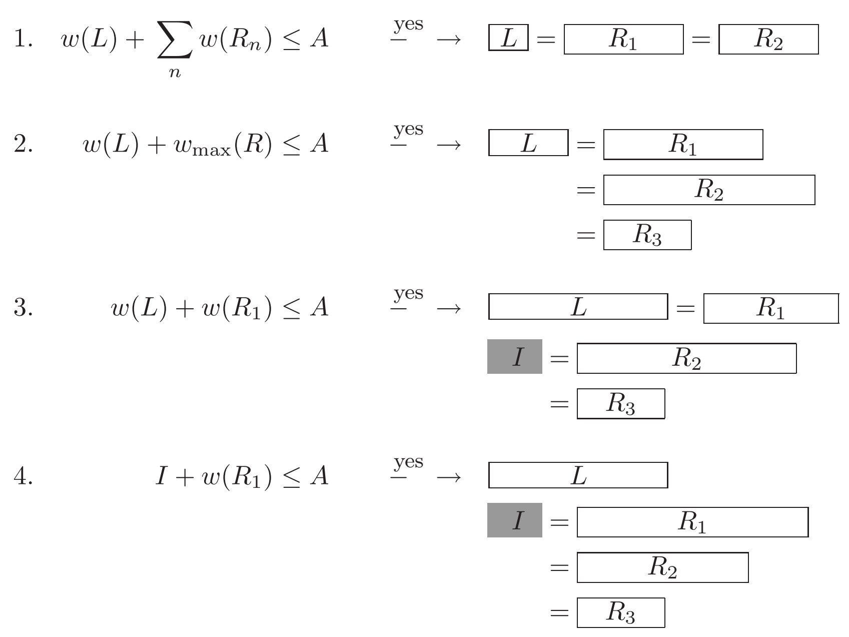

1. $w(L) + \sum_n w(R_n) \le A$
yes
L = R_1 = R_2
2. $w(L) + w_{\max}(R) \le A$
yes
L = R_1 = R_2 = R_3
3. $w(L) + w(R_1) \le A$
yes
L = R_1
I = R_2 = R_3
4. $I + w(R_1) \le A$
yes
L
I = R_1 = R_2 = R_3

Figure 2: Equation-breaking algorithm: w width, L left-hand side, R right-hand side, A available width, I indent

---

Equation Example A. This equation (Knuth, 1986, Chapter 19, p 195) remains just a few characters too wide even after shrinking the spaces around the + and − symbols.

$\sigma(2^{34}{-}1,2^{35},1)={-3}{+}(2^{34}{-}1)/2^{35}{+}2^{35}/(2^{34}{-}1){+}7/2^{35}(2^{34}{-}1){-}\sigma(2^{35},2^{34}{-}1,1).\blacksquare$

Knuth discussed how to break the equation using `\eqalign`, with the following result:

$$\begin{aligned}\sigma(2^{34}-1,2^{35},1) &= -3+(2^{34}-1)/2^{35}+2^{35}/(2^{34}-1)\\ &\qquad +7/2^{35}(2^{34}-1)-\sigma(2^{35},2^{34}-1,1).\end{aligned}$$

Equation Example B. A change in the representation of the fractions would allow the entire equation of Equation Example A to fit on a single line:

$$\sigma(2^{34}-1,2^{35},1) = -3+\frac{2^{34}-1}{2^{35}}+\frac{2^{35}}{2^{34}-1}+\frac{7}{2^{35}(2^{34}-1)}-\sigma(2^{35},2^{34}-1,1)$$

Most mathematicians would probably find this alternative more readable, as well.

---

equation. This has the potential to save authors a lot of work if they ever have to switch a book from one book design to another one that has different equation numbering conventions.

```
\begin{dmath}
x_nu_1+\dotsb+x_{n+t-1}u_t
  =x_nu_1+(ax_n+c)u_2+\dotsb
   +\bigl(a^{t-1}x_n
   +c(a^{t-2}+\dotsb+1)\bigr)u_t
  =(u_1+au_2+\dotsb+a^{t-1}u_t)x_n
   +h(u_1,\dotsc,u_t)
\end{dmath}.
```

An interesting feature of the next example (Knuth, 1986, Exercise 19.9) is that it is the second relation symbol, not the first one, to which the remaining relation symbols are aligned.

$$\begin{aligned}T(n)\le T(2^{\lceil\lg n\rceil}) &\le c(3^{\lceil\lg n\rceil}-2^{\lceil\lg n\rceil})\\ &< 3c\cdot 3^{\lg n}\\ &= 3c\,n^{\lg 3}.\end{aligned}$$

```
$$\eqalign{T(n)\le T(2^{\lceil\lg n\rceil})
    &\le c(3^{\lceil\lg n\rceil}
      -2^{\lceil\lg n\rceil})\cr
    &<3c\cdot3^{\lg n}\cr
    &=3c\,n^{\lg3}.\cr}$$
```

With the amsmath package that would be written

```
\begin{equation*}
\begin{split}
T(n)\le T(2^{\lceil\lg n\rceil})
  &\le c(3^{\lceil\lg n\rceil}
    -2^{\lceil\lg n\rceil})\\
  &<3c\cdot3^{\lg n}\\
  &=3c\,n^{\lg3}.
\end{split}
\end{equation*}
```

Using the dmath* environment you would use a \> command (note — not ampersand) to mark the relation symbol as the preferred alignment point:

```
\begin{dmath*}
T(n)\le T(2^{\lceil\lg n\rceil})
  \>\le c(3^{\lceil\lg n\rceil}
    -2^{\lceil\lg n\rceil})
  <3c\cdot3^{\lg n}
  =3c\,n^{\lg3}
\end{dmath*}.
```

Groups of equations

For an unaligned group of equations (see Equation Example C), Knuth recommended \displaylines:

```
$$\displaylines{%
  \hfill x\equiv x;\hfill\llap{(1)}\cr
  \hfill\hbox{if}\quad x\equiv y\quad
   \hbox{then}\quad y\equiv x;\hfill
   \llap{(2)}\cr
  \hfill\hbox{if}\quad x\equiv y\quad
   \hbox{and}\quad y\equiv z\quad
   \hbox{then}\quad x\equiv z.\hfill
   \llap{(3)}\cr}$$
```

With the amsmath package, the appropriate environment is gather:

```
\begin{gather}
  x\equiv x;\\
  \text{if}\quad x\equiv y\quad
    \text{then}\quad y\equiv x;\\
  \text{if}\quad x\equiv y\quad
    \text{and}\quad y\equiv z\quad
    \text{then}\quad x\equiv z.
\end{gather}
```

With the breqn package, it would be written as

```
\begin{dgroup*}[aligned={F}]
\begin{dmath}
  x\equiv x
\end{dmath};
\begin{dmath}
  \text{if}\quad x\equiv y\quad
    \text{then}\quad y\equiv x
\end{dmath};
\begin{dmath}
  \text{if}\quad x\equiv y\quad
    \text{and}\quad y\equiv z\quad
    \text{then}\quad x\equiv z
\end{dmath}.
\end{dgroup*}
```

A simple example of aligned equations (Knuth, 1986, Chapter 19, p 190):

```
$$\eqalign{
  X_1+\cdots+X_p&=m,\cr
  Y_1+\cdots+Y_q&=n.\cr
}$$
```

In LaTeX that would be written

```
\begin{eqnarray*}
  X_1+\cdots+X_p&=&m,\\
  Y_1+\cdots+Y_q&=&n.
\end{eqnarray*}
```

With the breqn package it would be written

```
\begin{dgroup*}
\begin{dmath*}
  X_1+\dotsb+X_p = m
\end{dmath*},
\begin{dmath*}
  Y_1+\dotsb+Y_q = n
\end{dmath*}.
\end{dgroup*}
```

Here is an equation group with a single common number (Knuth, 1986, Exercise 19.10):

```
$$\eqalign{%
  P(x)&=a_0+a_1x+a_2x^2+\cdots+a_nx^n,\cr
  P(-x)&=a_0-a_1x+a_2x^2-\cdots
         +(-1)^na_nx^n.\cr}\eqno(30)$$
```

In LaTeX this would need to be done with the array environment.

```
\begin{equation}
\begin{array}{rcl}
  P(x)&=&a_0+a_1x+a_2x^2+\cdots+a_nx^n,\\
  P(-x)&=&a_0-a_1x+a_2x^2-\cdots
          +(-1)^na_nx^n.
\end{array}
\end{equation}
```

But without further adjustments, spacing around the equals signs would be egregiously large, as in the eqnarray environment, and the interline spacing would be too small, and so on (see Equation Example D). Using the breqn package:

```
\begin{dgroup}
\begin{dmath*}
  P(x)=a_0+a_1x+a_2x^2+\dotsb+a_nx^n
\end{dmath*},
\begin{dmath*}
```

Equation Example C. This is from (Knuth, 1986, Chapter 19, p 194).

$$x \equiv x; \tag{1}$$
$$\text{if} \quad x \equiv y \quad \text{then} \quad y \equiv x; \tag{2}$$
$$\text{if} \quad x \equiv y \quad \text{and} \quad y \equiv z \quad \text{then} \quad x \equiv z. \tag{3}$$

Equation Example D. Aligned equations done with the LaTeX `array` environment in order to get the equation number centered:

$$\begin{array}{rcl} P(x) & = & a_0 + a_1x + a_2x^2 + \cdots + a_nx^n, \\ P(-x) & = & a_0 - a_1x + a_2x^2 - \cdots + (-1)^na_nx^n. \end{array} \tag{30}$$

Notice the too-large spacing around the equal signs and the too-small interline spacing. These can be corrected, but only by rather cumbersome extra work.

```
 P(-x)=a_0-a_1x+a_2x^2-\dotsb+(-1)^na_nx^n
\end{dmath*}.
\end{dgroup}
```

Perhaps some of the lines are numbered and others are not (*The TeXbook* Chapter 19 p 192):

$$\begin{aligned} (x+y)(x-y) &= x^2 - xy + yx - y^2 \\ &= x^2 - y^2; \end{aligned} \tag{31}$$
$$(x+y)^2 = x^2 + 2xy + y^2. \tag{32}$$

```
$$\eqalignno{(x+y)(x-y)&=x^2-xy+yx-y^2\cr
    &=x^2-y^2;&(4)\cr
  (x+y)^2&=x^2+2xy+y^2.&(5)\cr}$$
```

In LaTeX this would typically be accomplished in with the `eqnarray` environment and `\nonumber`. With the `amsmath` package it would be done with a combination of `split` and `align`:

```
\begin{align}
\begin{split}
  (x+y)(x-y)&=x^2-xy+yx-y^2\\
    &=x^2-y^2;
\end{split}
\\% align break
  (x+y)^2&=x^2+2xy+y^2.
\end{align}
```

With the `breqn` package the logical distinction between a long, split equation and equations that are entirely separate is clearer. Among other things, this makes it possible for documentclasses to specify that the space between distinct equations should be larger than the space between the lines of a single multiline equation, and/or should stretch more if necessary to fill up a page.

```
\begin{dgroup}
\begin{dmath}
(x+y)(x-y) =x^2-xy+yx-y^2
     =x^2-y^2
\end{dmath};
\begin{dmath}
  (x+y)^2 =x^2+2xy+y^2
\end{dmath}.
\end{dgroup}
```

From *The TeXbook* Chapter 19 p193. You can also insert a line of text between two equations, without losing the alignment. For example, consider the two displays

$$x = y + z$$

and

$$x^2 = y^2 + z^2.$$

which Knuth demonstrates as

```
$$\eqalignno{x&=y+z\cr
  \noalign{\hbox{and}}
  x^2&=y^2+z^2.\cr}$$
```

The `amsmath` package provides an `\intertext` command for this situation:

```
\begin{align*}
  x&=y+z\\
\intertext{and}
  x^2&=y^2+z^2.
\end{align*}
```

The `dmath` environment reimplements 'intertext' as an environment. Among other things this means that the text can contain a bit of verbatim, should that ever be necessary.

```
\begin{dgroup}
\begin{dmath*}
  x =y+z
\end{dmath*}
\begin{intertext}
and
\end{intertext}
\begin{dmath*}
  x^2 =y^2+z^2
\end{dmath*}.
\end{dgroup}
```

Examples from (Fomin, Gelfand, and Postnikov, 1997). Note: the authors had added space by hand before all the end-of-display punctuation to get the amount of space they desired. With `breqn` conventions this is handled automatically and the amount of space is easy to change.

$$\partial_w(fg) = w(f)\partial_w g + \sum(\lambda_i - \lambda_j)\partial_{wt_{ij}} g \ ,$$

```
\begin{dmath*}
\partial_w(fg)=w(f)\partial_w g
  +\sum(\lambda_i-\lambda_j)\partial_{wt_{ij}}g
\end{dmath*},
```

$$\partial_w(fg) = w(f)\partial_w g + \sum(\lambda_i - \lambda_j)\partial_{wt_{ij}} g \ , \qquad (2.5)$$

```
\begin{dmath}
\partial_w(fg)=w(f)\partial_w g
  +\sum(\lambda_i-\lambda_j)\partial_{wt_{ij}}g
\end{dmath},
```

$$\partial_i\partial_j = \partial_j\partial_i \quad \text{for } |i-j| > 1 \ ,$$
$$\partial_i\partial_{i+1}\partial_i = \partial_{i+1}\partial_i\partial_{i+1} \ ,$$
$$\partial_i^2 = 0 \ .$$

```
\begin{dgroup*}[aligned={F}]
\begin{dmath*}
\partial_i \partial_j = \partial_j \partial_i
  \condition[]{for $\abs{i-j}>1$}
\end{dmath*},
\begin{dmath*}
\partial_i\partial_{i+1}\partial_i
  =\partial_{i+1}\partial_i\partial_{i+1}
\end{dmath*},
\begin{dmath*}
\partial_i^2 = 0
\end{dmath*}.
\end{dgroup*}
```

$$\partial_i\partial_j = \partial_j\partial_i \quad \text{for } |i-j| > 1 \ , \qquad (2.3)$$
$$\partial_i\partial_{i+1}\partial_i = \partial_{i+1}\partial_i\partial_{i+1} \ , \qquad (2.4)$$
$$\partial_i^2 = 0 \ . \qquad (2.5)$$

```
\begin{dgroup*}
\begin{dmath}
\partial_i \partial_j = \partial_j \partial_i
  \condition[]{for $\abs{i-j}>1$}
\end{dmath},
\begin{dmath}
\partial_i\partial_{i+1}\partial_i
  =\partial_{i+1}\partial_i\partial_{i+1}
\end{dmath},
\begin{dmath}
\partial_i^2 = 0
\end{dmath}.
\end{dgroup*}
```

Examples from (Fomin, Gelfand, and Postnikov, 1997) (continued).

$$\begin{aligned} \partial_i \partial_j &= \partial_j \partial_i \quad \text{for } |i-j| > 1 \ , \\ \partial_i \partial_{i+1} \partial_i &= \partial_{i+1} \partial_i \partial_{i+1} \ , \\ \partial_i^2 &= 0 \ . \end{aligned} \tag{2.3}$$

```
\begin{dgroup}
\begin{dmath*}
\partial_i \partial_j = \partial_j \partial_i
  \condition[]{for $\abs{i-j}>1$}
\end{dmath*},
\begin{dmath*}
\partial_i\partial_{i+1}\partial_i
  =\partial_{i+1}\partial_i\partial_{i+1}
\end{dmath*},
\begin{dmath*}
\partial_i^2 = 0
\end{dmath*}.
\end{dgroup}
```

$$\begin{aligned} e_j^k(e_{i+1}^k - e_{i+1}^{k+1}) &= -e_i^k(e_{j+1}^{k+1} - e_{j+1}^k) \\ &\cdots \\ &= \sum_{l \geq 1} (e_{i+1-l}^{k+1} e_{j+l}^k - e_{j+1}^{k+1} e_{i+1-l}^k) \ . \quad \square \end{aligned}$$

```
\begin{dmath*}[qed={T}]
e_j^k(e_{i+1}^k-e_{i+1}^{k+1})
  = -e_i^k(e_{j+1}^{k+1}-e_{j+1}^k)
  ...
  = \sum_{l\geq 1}
    (e_{i+1-l}^{k+1} e_{j+l}^k
     - e_{j+1}^{k+1} e_{i+1-l}^k)
\end{dmath*}.
```

$$\psi : f \mapsto F \ , \quad f = F(\mathcal{X})(1) \ . \tag{5.4}$$

```
\begin{dgroup}[inline={T}]
\begin{dmath*}
\psi : f\mapsto F
\end{dmath*},
\begin{dmath*}
f=F(\mathcal{X})(1)
\end{dmath*}.
\end{dgroup}
```

The reduced minimal Gröbner basis for I_3^q consists of

$$H_1^3 = x_1 + x_2 + x_3 \ ,$$

$$H_2^2 = x_1^2 + x_1 x_2 + x_2^2 - q_1 - q_2 \ ,$$

and

$$H_3^1 = x_1^3 - 2x_1 q_1 - x_2 q_1 \ .$$

```
\begin{dgroup*}
\begin{dmath*}
H_1^3 = x_1+x_2+x_3
\end{dmath*},
\begin{dmath*}
H_2^2 = x_1^2+x_1 x_2+x_2^2-q_1-q_2
\end{dmath*},
\begin{intertext}
and
\end{intertext}
\begin{dmath*}
H_3^1 = x_1^3  - 2 x_1 q_1 - x_2 q_1
\end{dmath*}.
\end{dgroup*}
```

Additional features under consideration

- With the package option `refnumbers`, all equations remain unnumbered except those that are actually referred to (this idea coming from Enrico Bertolazzi's `easyeqn` package).
- In an equation group, a vertical bracket reaching from the equation number indicates the lines to which the number applies.
- More explicit support for the Russian typesetting conventions for equation breaks, as described by Grinchuk (Grinchuk, 1996). The `flexisym` provides a natural entry point.

Concluding remarks

I think it may be clear from the above discussion that the subject of typesetting equations is far too broad and complex to be covered in adequate detail in the space available here. I have perforce limited myself to sketching out the goals of the `breqn` package, the LaTeX2e user syntax, and some of the more significant constraints on the problems it attempts to solve. Further details are available in the documentation that accompanies the package (which, unlike this article, will continue to evolve as details change, as they surely will given that the package is barely into alpha stage as I write).

References

Borde, Arvind. *Mathematical TeX by Example*. Academic Press, New York, 1995.

Downes, Michael J. *AmS-LaTeX Version 1.2 User's Guide*. American Mathematical Society, Providence, R.I., 1995.

Duff, M. J., R. Minasian, and E. Witten. "Evidence for heterotic/heterotic duality". *xxx.lanl.gov e-Print archive* (hep-th/9601036), 1996.

Fomin, Sergey, S. Gelfand, and A. Postnikov. "Quantum Schubert polynomials". *J. Amer. Math. Soc.* **10**(3), 565–596, 1997.

Grinchuk, Mikhail Ivanovich. "TeX and Russian traditions of typesetting". *TUGboat* **17**, 385–388, 1996.

Knuth, Donald E. *The TeXbook*. Addison Wesley, Reading, Mass., 1986.

Kopka, Helmut and P. W. Daly. *A guide to LaTeX 2_ε*. Addison-Wesley, Wokingham, England, 1995.

The LaTeX3 Project

Frank Mittelbach and Chris Rowley
LaTeX3 project
Frank.Mittelbach@eds.com, C.A.Rowley@open.ac.uk

Abstract

This article describes the motivation, achievements and future of the LaTeX3 Project*, which was established to produce a new version of LaTeX, the widely-used and highly-acclaimed document preparation system. It also describes how you can help us to achieve our aims.

Note for Archive maintainers, Authors, Publishers and Distributors: The project team request that, whenever possible, you include this article in any of the following:

- Books about TeX and LaTeX.
- Instructions for authors on using LaTeX.
- The printed documentation of CD-ROM collections that contain LaTeX.
- On-line collections that include a significant proportion of documents encoded in LaTeX.

Outline

The purposes of the LaTeX3 system can be summarized thus: it will greatly increase the range of documents which can be processed; and it will provide a flexible interface for typographic designers to easily specify the formatting of a class of documents.

The LaTeX3 Project Team is a small group of volunteers whose aim is to produce this major new document processing system based on the principles pioneered by Leslie Lamport in the current LaTeX.

The major visible work of the team before 1997 was the development of the current *standard* version of LaTeX. This was first released in 1994 and has since then been actively mainatined and enhanced by extensions to that core system. They will continue to develop and maintain this system, releasing updated versions every six months and recording these activities in the LaTeX bugs database (see below).

Although LaTeX may be distributed freely, the production and maintenance of the system does require expenditure of reasonably large sums of money. The LaTeX3 Project Fund has therefore been set up to channel money into this work. We know that some users are aware of this fund as they have already contributed to it—many thanks to all of them! If you want to know more about how you can help the project, see Page 197—and thanks in advance for your generosity in the future.

Background

With TeX, Knuth designed a formatting system that is able to produce a large range of documents typeset to extremely high quality standards. For various reasons (e.g., quality, portability, stability and availability) TeX spread very rapidly and can nowadays be best described as a world-wide de facto standard for high quality typesetting. Its use is particularly common in specialized areas, such as technical documents of various kinds, and for multilingual requirements.

The TeX system is fully programmable. This allows the development of high-level user interfaces whose input is processed by TeX's interpreter to produce low-level typesetting instructions; these are input to TeX's typesetting engine which outputs the format of each page in a device-independent page-description language. The LaTeX system is such an interface; it was designed to support the needs of long documents such as textbooks and manuals. It separates content and form as much as possible by providing the user with a generic (i.e., logical rather than visual) mark-up interface; this is combined with style sheets which specify the formatting.

Recent years have shown that the concepts and approach of LaTeX are now widely accepted. Indeed, LaTeX has become the standard method of communicating and publishing documents in many academic

* In addition to the authors, current members of The LaTeX3 Project Team are Johannes Braams (NL), David Carlisle (UK), Michael Downes (USA), Alan Jeffrey (UK) and Rainer Schöpf (DE).

disciplines. This has led to many publishers accepting LaTeX source for articles and books; and the American Mathematical Society now provides a LaTeX package making the features of AMS-TeX available to all users of LaTeX. Its use has also spread into many other commercial and industrial environments, where the technical qualities of TeX together with the concepts of LaTeX are considered a powerful combination of great importance to such areas as corporate documentation and publishing. This has also extended to on-line publishing using, for example, PDF output incorporating hypertext and other active areas.

With the spreading use of SGML-compliant systems (e.g., Web-based publishing using HTML or XML) TeX again is a common choice as the formatting engine for high quality typeset output: a widely used such system is *The Publisher* from ArborText, whilst a more recent development is the object-oriented document editor Grif. The latter is used for document processing in a wide range of industrial applications; it has also been adopted by the Euromath consortium as the basis of their mathematician's workbench, one of the most advanced of the emerging project-oriented user environments. Typeset output from SGML-coded documents in these systems is obtained by translation into LaTeX, which will therefore soon also be a natural choice for the output of DSSSL-compliant systems.

Because a typical SGML Document Type Definition (DTD) uses concepts similar to those of LaTeX, the formatting is often implemented by simply mapping document elements to LaTeX constructs rather than directly to 'raw TeX'. This enables the sophisticated analytical techniques built into the LaTeX software to be exploited; and it avoids the need to program in TeX.

Motivation

This increase in the range of applications of LaTeX has highlighted certain limitations of the current system, both for authors of documents and for designers of formatting styles.

In addition to the need to extend the variety of classes of document which can be processed by LaTeX, substantial enhancements are necessary in, at least, the following areas:

- the command syntax (attributes, short references, etc);
- the layout specification interface (style design);
- the level of robustness (error recovery, omitted tags);
- the extendibility (package interface);
- the layout specification of tabular material;
- the specification and inclusion of graphical material;
- the positioning of floating material, and other aspects of page layout;
- the requirements of hypertext systems.

Further analysis of these deficiencies has shown that some of the problems are to be found in LaTeX's internal concepts and design. This project to produce a new version therefore involves thorough research into the challenges posed by new applications and by the use of LaTeX as a formatter for a wide range of documents, e.g., SGML documents; on-line PDF documents with hypertext links.

This will result in a major re-implementation of large parts of the system. Some of the results of such rethinking of the fundamentals are already available in Standard LaTeX, notably in the following areas:

- Font declaration and selection;
- Font and glyph handling within mathematical formulas;
- Handling multiple font glyph encodings within a document;
- Allowing multiple input character encodings within a document;
- A uniform interface for graphics inclusion;
- Support for coloured text;
- Building and interfacing new classes and extension packages.

Description

The strengths of the present version of LaTeX are as follows:

- excellent standard of typesetting for text, technical formulas
  and tabular material;
- separation of generic mark-up from visual formatting;
- ease of use for authors;
- portability of documents over a wide range of platforms;
- adaptability to many languages;
- widespread and free availability;
- reliable support and maintenance by the LaTeX3 project team.

These will be preserved and in many cases greatly enhanced by the new version which is being developed to fulfill the following requirements.

- It will provide a syntax that allows highly automated translation from popular SGML DTDs into LaTeX document classes (these will be provided as standard with the new version).

The syntax of the new LaTeX user-interface will, for example, support the SGML concepts of 'entity', 'attribute' and 'short reference' in such a way that these can be directly linked to the corresponding SGML features.

- It will support hypertext links and other features required for on-line structured documents using, for example, HTML and XML.
- It will provide a straightforward style-designer interface to support both the specification of a wide variety of typographic requirements and the linking of entities in the generic mark-up of a document to the desired formatting. These two parts of the design process will be clearly separated so that it is possible to specify different layouts for the same DTD.

  The language and syntax of this interface will be as natural as possible for a typographic designer. As a result, this language could easily be interfaced to a visually-oriented, menu-driven specification system.

  This interface will also support DSSSL specifications and style-sheet concepts such as those used with HTML and XML.
- It will provide an enhanced user-interface that allows expression of the typesetting requirements from a large range of subject areas. Some examples are listed here.
  - The requirements of technical documentation (e.g., offset layout, change bars, etc).
  - The requirements of academic publishing in the humanities
    (critical text editions, etc).
  - The requirements of structural formulas in chemistry.
  - Advanced use of the mathematical-typesetting features of TeX.
  - The integration of graphical features, such as shading, within text.
  - the integration of hypertext and other links in on-line documents using systems such as HTML, XML and PDF.

  Special care will be taken to ensure that this interface is extensible: this will be achieved by use of modular designs.
- It will provide a more robust author-interface. For example, artificial restrictions on the nesting of commands will be removed. Error handling will be improved by adding a more effective, interactive help system.
- It will provide access to arbitrary fonts from any family (such as the PostScript and TrueType fonts) including a wide range of fonts for multilingual documents and the specialist glyphs required by documents in various technical and academic areas.
- The new interfaces will be documented in detail and the system will provide extensive catalogues of examples, carefully designed to make the learning time for new users (both designers and authors) as short as possible.
- The code itself will be thoroughly documented and it will be designed on modular principles. Thus the system will be easy to maintain and to enhance.

The resulting new LaTeX will, like the present version, be usable with any standard TeX system (or whatever replaces it) and so will be freely available on a wide range of platforms.

LaTeX documentation

A complete description of Standard LaTeX can be found in:

LaTeX: A Document Preparation System Leslie Lamport, Addison-Wesley, 2nd ed, 1994.

The LaTeX Companion Goossens, Mittelbach and Samarin, Addison-Wesley, 1994.

A recent addition to the publications closely associated with the project is:

The LaTeX Graphics Companion Goossens, Mittelbach and Rahtz, Addison-Wesley, 1997.

This LaTeX distribution comes with documentation on several aspects of of the system. The newer features of the system are described in the following documents:

LaTeX 2ε for authors describes the new features of LaTeX documents, in the file `usrguide.tex`;

LaTeX 2ε for class and package writers describes how to produce LaTeX classes and packages, in the file `clsguide.tex`;

LaTeX 2ε font selection describes the new features of LaTeX fonts for class and package writers, in the file `fntguide.tex`.

For further contacts and sources of information on TeX and LaTeX, see the addresses on Page 198.

The LaTeX3 Project Fund

Although LaTeX may be distributed freely, the production and maintenance of the system does require expenditure of reasonably large sums of money. There are many necessities that need substantial financing: examples are new or enhanced computing equipment and travel to team meetings (the volunteers come from many different countries, so getting together occasionally is a non-trivial exercise).

This is why we are appealing to you for contributions to the fund. Any sum will be much appreciated; the amount need not be large as small contributions add up to very useful amounts. Contributions of suitable equipment and software will also be of great value. This appeal is both to you as an individual author and to you as a member of a group or as an employee: please encourage your department or your employer to contribute towards sustaining our work.

We should like to see funded projects that make considerable use of LaTeX (e.g., conferences and research teams who use it to publish their work, and electronic research archives using it) include contributions to this fund in their budgets.

We are also asking commercial organisations to assess the benefits they gain from using, or distributing, a well-supported LaTeX and to make appropriate contributions to the fund in order that we can continue to maintain and improve the product. If you work for, or do business with, such an organisation, please bring to the attention of the relevant people the existence and needs of the project.

In particular, we ask that all the large number of organisations and businesses that distribute LaTeX, within other software or as part of a CD-ROM collection, should consider pricing all products containing LaTeX at a level that enables them to make regular donations to the fund from the profit on these items. We also ask all authors and publishers of books about LaTeX to consider donating part of the royalties to the fund.

Contributions should be sent to one of the following addresses:

TeX Users Group, P.O. Box 2311
Portland, OR 97208-2311 USA
Fax: +1 503 223 3960
Email: `tug@tug.org`

UK TUG, 1 Eymore Close, Selly Oak
Birmingham B29 4LB UK
Fax: +44 121 476 2159
Email: `uktug-enquiries@tex.ac.uk`

Cheques should be payable to the user group (TUG or UKTUG) and be clearly marked as contributions to the LaTeX3 fund. Many thanks to all of you who have contributed in the past and thanks in advance for your generosity in the future.

Contacts and information

In addition to the sources mentioned above, LaTeX has its home page on the World Wide Web at:

`http://www.tex.ac.uk/CTAN/latex/`

This page describes LaTeX and the LaTeX3 project, and contains pointers to other LaTeX resources, such as the user guides, the TeX Frequently Asked Questions, and the LaTeX bugs database.

More general information, including contacts for local User Groups, can be accessed via:

`http://www.tug.org/`

The electronic home of anything TeX-related is the Comprehensive TeX Archive Network (CTAN). This is a network of cooperating ftp sites, with over two gigabytes of TeX material:

`ftp://ftp.tex.ac.uk/tex-archive/`
`ftp://ftp.dante.de/tex-archive/`

For more information, see the LaTeX home page.

Language Information in Structured Documents: A Model for Mark-up and Rendering

Frank Mittelbach and Chris Rowley
LaTeX3 project
`Frank.Mittelbach@eds.com, C.A.Rowley@open.ac.uk`

Abstract

In this paper* we discuss the structure and processing of multi-lingual documents, both at a general level and in relation to a proposed extension to the (no longer so new) standard LaTeX. Both in general and in the particular case of this proposal, our work would be impossible without the enormous support, both practical and moral, we get from our fellow members of the LaTeX3 project team† (who maintain and enhance LaTeX) and from people all over the world who contribute to the development of LaTeX with their suggestions and comments.

Introduction

The paper starts by examining the language structure of documents and from this a language tag model for LaTeX is developed. It then discusses the relationship between language and document formatting and the types of actions needed at a change of language. This will lead to a model that supports the specification of these actions and of their association with the tag structure in the abstract document.

The model is then extended to provide the necessary support for regions that have their own visual context or that receive content from other parts of the document, thus breaking the basic tree structure of an abstract document — this is in the section entitled "Special Regions".

Finally a high level summary of the required interfaces is given. A full formal specification, to be used for a prototype implementation in LaTeX, is currently under development—a first public test implementation is expected to exist for the 1997/12/01 release of LaTeX.

If you are interested in the issues raised in this paper or in other aspects of our work to enhance LaTeX, please join the project's electronic discussion list. To do this, please send a message to:

`listserv@relay.urz.uni-heidelberg.de`

Containing this line:

`subscribe LATEX-L` *`your name`*

* This paper was originally given at the Multilingual Information Processing symposium, March 1997, Tsukuba, Japan.

† Current LaTeX3 project team members are Johannes Braams (NL), David Carlisle (UK), Michael Downes (USA), Alan Jeffrey (UK) and Rainer Schöpf (DE).

Language Structure of Documents

Structured documents can be understood as being explicitly or implicitly labeled with "language tags" denoting that a portion of the document contains data written in a certain "language".

These tags have the following properties:

- They impose on the document a hierarchical tree structure that may not be compatible with that document's other logical structure, e.g., there might be a language change in the middle of a logical element such as a list item.[1]
- At any one point in the document the "current language" can be determined.

The term "language" in this context is somewhat vague and might need further qualification; but for the purpose of the following discussion it is sufficient to define it as a 'label' whose value affects certain aspects of formatting.

Hierarchy of language tags

The structure created by attaching such language tags to the text can be considered to be of varying complexity. The simplest case would be to regard this as a flat structure: for each point in the document only a "current" language is defined, disregarding the fact that certain language segments can be considered to be embedded within others. This model of language within documents is, for example, employed within the current Babel system where, by default, all language changes are in this sense global.

[1] However, for practical purposes it is normally possible and acceptable to artificially force the structure imposed by the language tags into the logical hierarchy imposed by other tags.

In a more complex model each area has a "current" language but may be embedded within a nest of larger areas, each in its own language. In such a model, a change of language has a different quality, and therefore may invoke different formatting changes, depending on the level in the hierarchy at which it occurs.

Our investigations lead us to conclude that, to properly render a document, one needs a combination of both models:

- the concept of a base language for very large portions of a text (for most documents this will in fact be only one such language for the full text): this has a flat structure, there is only one base language at any point in the text;
- the concept of imbedded language segments: these are nestable (to any number of levels) and are used for relatively small-scale insertions within a base language, such as quotations or names.

Language tag (visual) structure

In addition to the nesting structure of language tags, there is a more visual component that influences rendering of a document: the paragraph structure. To properly model this typographical treatment it is necessary to classify the language tags according to whether a language segment contains only complete paragraphs or is part of the running text of a single paragraph. A begin/end pair of tags is called a "block-level" tag if its body consists of complete paragraphs and a "paragraph-level" tag otherwise. As later examples will show, the typographical treatment for these two types is often different.

A Tag Model for LaTeX

To support the above model, including both nesting of language tags and the differentiation between block- and paragraph-level tags, the following tag structure for a system like LaTeX is proposed:

- A document language tag (implicit). This tag can be used to attach language-related typographical actions that should not change even if the document contains more than one base language.
- Base-language tags: used only at top-level, no nesting. These tags denote the major language(s) within a document. In the case of essentially mono-lingual documents the base language would be the same as the document language.
- Language-block tags: contain complete paragraphs, nestable. These denote larger imbeddings either directly within the base language or further down in the nesting hierarchy.
- Language-fragment tags: only within paragraphs, nestable. These denote smaller imbeddings but are otherwise identical to language block tags.

Note that since, at least in the logical structure of a document, paragraphs can occur within paragraphs, block tags can be nested within fragment tags.

Document interfaces

As LaTeX 2_ε does not have built in support for named attributes, its support for language changes is best implemented by introducing additional language tags (commands and environments). A concrete syntax for these tags could include the following:

- A preamble declaration for the document language (this is also the base language in mono-lingual documents) with the language-label as argument.
- A base-language change command with the language-label as argument. This command is declarative to highlight the flat structure of base languages.
- A language-environment with the language-label as argument and text as body. Such an environment starts a new paragraph so as to enforce the block-level nature of the tag.
- A language-command with the language-label and text both as arguments. In contrast to the environment, this command applies language-related actions to its second argument, which cannot directly contain full paragraphs.

For LaTeX3 we shall probably normalize this interface by supporting a language attribute on appropriate tags. This would allow, for example, a trivial translation of the language features currently being proposed for HTML into LaTeX for rendering purposes. However, even in that case generic tags for changing language are necessary as typical documents contain language changes that do not coincide with the tag boundaries of other logical tags.[2]

Language-dependent Processing

Setting up the tags tells us only how to encode a multi-lingual document. We now need to specify how these tags affect the processing of the document; how do we attach actions to them? Before answering this question we shall first discuss a

[2] It is proposed that HTML 3.2 supports a `<span>` tag for this purpose.

number of representative examples of the effects of language on this processing, classified according to the categories input, transformation and formatting.

The actions shown below are all commonly related to a change of language within a document. Nevertheless, it is not the case that each of them should necessarily be implemented by attaching them firmly to language changes. For some it might be more appropriate to freeze them for the whole document or to attach them to areas within the document that do not coincide with language boundaries.

Input

Input encodings Entering text in a certain language often requires special input methods (this is especially true for languages with complex scripts) but even in cases where direct keyboard entry is possible it might be necessary to add information about the keyboard codepage that is to be used, so as to interpret the source characters correctly. At present LaTeX supports variable interpretation of the upper half of the 8-bit plane, thus allowing source text to be 8-bit encoded in one of the many keyboard encodings used world wide.

Short-refs With the development of language packages and the subsequent development of the Babel system, it became common practice to extend the mark-up language of LaTeX using so called "short-refs" as a compact method for inputting certain commands. Short-refs are character sequences that do not start with TeX's escape character, i.e., usually '\', but nevertheless act like commands. That is, they do not represent the equivalent glyph sequence but have either additional effects (e.g., the punctuation marks in French typography, which produce additional space) or even denote completely different actions (e.g., `""` for a break point without a hyphen).

In addition to the above short-refs, some TeX fonts implement short-refs by using (or misusing) the ligature mechanism to implement arbitrary input syntax, e.g., ` `` ` generating " or `---` generating an em-dash.

Short-refs can be used for different purposes:

- providing a compact input notation for commonly used textual commands such as characters with diacritical marks;
- providing a compact and readable input notation for special applications, e.g., `==>` for `\Longrightarrow`;
- providing typographical features not otherwise supported (e.g., extra space in front of punctuation characters).

The first two items are related to input syntax and not directly linked to the language of the current text although historically they have been provided by language packages, e.g., `"a` as a short-ref for `\"{a}` was implemented by `german.sty` and within Babel its meaning gets deactivated within regions marked up as belonging to other languages.

The third item is directly related to language since short-refs of this type are used to implement a typographic style that is characteristic of a language in such a way that the user is not forced to use explicit mark-up in the document.

Transformations

Here, 'transformations' include only manipulations of the source text that are independent of formatting information (i.e., those that act entirely on the logical document). Usually such transformations enrich the document content in one way or the other by using knowledge stored outside the document source.

Generated text This is text that is not directly encoded in the source document but is produced from tags therein. Generated text can be classified into two categories: content-related and structure-related. Here content-related text is that generated by tags that can appear anywhere in the source text (a typical LaTeX example would be the `\today` command) while structure-related text refers to text that is associated with a high level logical structure (e.g., the heading produced for a bibliography or the fixed text used in a figure caption).

While it is imaginable to keep structure-related text in one language even though the surrounding language changes, content-related text most likely will have to change at every language tag.

Hyphenation The finding and marking of possible hyphenation points is, perhaps, the most obvious language-related transformation. Indeed, it is often considered to be the defining characteristic of a 'language'.

When using TeX this relationship is unfortunately obscured by some technical details of the implementation of hyphenation. One of these is that TeX's hyphenation does not depend only on the 'language' but also on the current font encoding (which can differ within a single language). Another is TeX's restriction that one can properly hyphenate a whole multi-lingual paragraph only if the font encodings used therein share a single lower-case table (and this is likely not to be the case if more than one script is present).

Upper- and lower-case transformations The mapping between upper- and lower-case characters (for those writing systems that make such a distinction) is language-dependent (and not just script-dependent): for example, in Turkish ı→I and i→İ in contrast to the usual mapping i→I used in most other languages. There can also be a one-to-many mapping as for the German ß that maps to SS.

Formatting

Although each of the examples listed here has been documented as characteristic of the typography associated with a particular language, they are all also aspects of the design over which a document designer may wish to have control that is independent of the language of the text.

Direction The direction of the text and, more generally, the writing system used are very strongly associated with the language in use.

Micro-rendering This covers the details of rendering at the level of individual glyphs and the relationships, often complex, between the characters which form the textual part of the logical document and the glyphs used to render this text, especially when aiming for the highest levels of typographic quality. These details often depend on what glyphs are provided by the available fonts. Also, when using TEX, this level of formatting is typically controlled entirely by the choice of font, whereas it should be possible to specify such details independent of the font since they also depend on the language in use.

Some examples:

- The precise positioning of diacritics depends on the language; e.g., a language such as German with many umlauts puts them closer to the top of the basic letter than is typically done with the diaeresis in English or French typography.
- The use of aesthetic ligatures varies from language to language, e.g., the ffl-ligature is traditionally not used in Portuguese and Turkish typography (implementing this is difficult in TEX since these transformations are normally controlled entirely by the font and there is no simple way to 'turn them off').

Macro-rendering More global aspects of typography can also be language-dependent, for example:

- the formatting of in-line quotes (i.e., what 'quotation marks' to use);
- rendering of enumerations;
- aspects of page layout (e.g., float placement).

As with most language-related actions they usually have a wide range of formatting possibilities and can be considered to depend, at least partially, on house style or other factors.

Attaching Actions to Change of Language

Having described some typical changes that need to be made at a language tag, we now look at how to tie particular actions to a particular tag, noting that it is not sensible, for example, to change every aspect of the formatting if only an in-line fragment of a few words is to be in a different language.

Attaching actions to tags

First we note the following facts.

- The type of actions that are required at language tags can be modeled by setting the values of a collection of parameters to those appropriate for the new language.
- Some actions may not make sense at certain levels of the hierarchies. For example, while one wants to use the correct hyphenation algorithm at any level of the hierarchies changing of micro-rendering, such as the positioning of diacritics, might be applied only to language changes for whole paragraphs but not for fragments.
- However, for most actions it is not possible to specify one place in the hierarchies that will produce the correct location of that action for *all* documents. The correct place might, for example, depend on document type or on a particular house style.

There are two (at least) possibilities for specifying, for a particular document, where in the tag hierarchy an action should be 'attached' (see Figure 1). These are by the nesting-level in the hierarchy of language tags or by the visual type of the language tags as described in the section entitled "Language tag (visual) structure". These visual tag-types implicitly define a partial hierarchy, from the top: document, base, block, fragment.

In both cases an action is defined to be executed down to a prescribed level in the hierarchy. As noted above, different actions might be executed down to different levels so there needs to be a mechanism to specify this level for each action. To limit the complexity of the model we think it is advisable to assume that this stopping level depends on the action but not on the language. It was pointed out in Tsukuba that this is probably an oversimplification, i.e., that there exist cases where it would be better to model the formatting of language-related

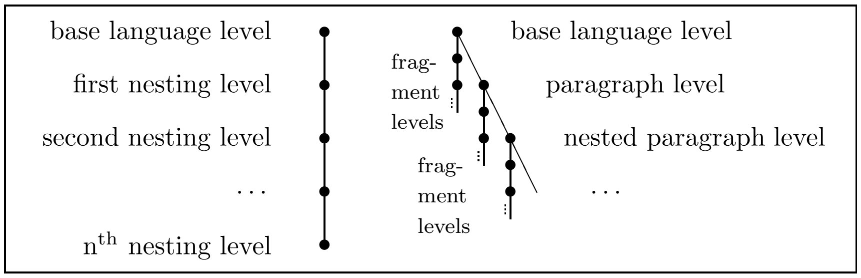


Figure 1: The two hierarchies

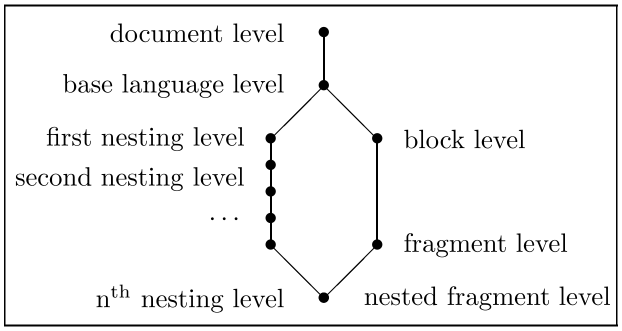


Figure 2: Tag hierarchy diagram (THD)

items by attaching of language/action pairs to levels. However, we think that these cases are sufficiently rare that they can be handled by the action itself.[3]

It is also possible to combine these two hierarchies and allow the attachment of actions to tags via either hierarchy (see Figure 2). In this case, for each action it is necessary to define:

- on which of the two hierarchies the stopping of the action depends;
- down to what level the action is carried out in that hierarchy.

Data structures for this model

For this model of language tags/actions, the system needs to specify the contents of the following three data structures.

Tag hierarchy diagram (THD) While combining the two hierarchies we have simplified their structure (compare figures 1 and 2), i.e., multiple nestings of paragraphs are collapsed into a single node. At the same time a new root node (document-level) was added. This node serves as an anchor point for typographic requirements that should stay fixed throughout the document even if the base language changes.

The required number of significant nestings in the hierarchy of nesting-levels is an open question but probably $n = 3$ is sufficient to specify typical formatting requirements.

The two end points of the hierarchies (n^th nesting-level and nested-fragment-level) are combined as they essentially mean to carry out attached actions in all cases, thus it does not matter on which hierarchy they are specified.

Another interesting point is that the two base-language-levels,one from each hierarchy, are combined.[4]

Nevertheless, it should be noted that the "level" of a tag within the THD is logically described by a pair of nodes (one on each hierarchy) even though in some cases these nodes collapse into one.

Language actions table (LAT) This two-dimensional table (indexed by parameter-group and language-label) stores the effect of each action (i.e., the value for a parameter-group) for each language (possibly only a default value if no value has been explicitly defined for that language). Each entry is an expression that returns a set of values appropriate to the parameter-group.

[3] An action that depends both on language and level could be specified in the model by executing it on all levels with an additional conditional within the action body testing for the current language.

[4] From this it follows that in this model a base language change is only allowed between paragraphs.

It may be possible[5] to also allow special actions to be specified, such as:

- leave unchanged;
- use some default (e.g. the value for the document language).

Parameter assignment map (PAM) This one-dimensional table maps each each action (modeled by a parameter-group) to a single node in the THD.

Such an assignment means that this parameter group changes its value (using the method specified in the LAT) at all levels down to (and including) the node to which it is mapped.

Special Regions

The scheme we have outlined so far will work well for the main text of many documents but it needs to be supplemented in order to handle formatting of the following material (called special regions):

- regions that contain text which has moved from other parts of the document, e.g., table of contents, running heads;
- regions of text that are first formatted and then the whole block is moved, e.g., (from LaTeX) floating tables, footnotes;
- regions that can contain elements breaking the type hierarchy, e.g., paragraphs in table-cells.

There are several problems that arise when "moving things around" in a document: one of these, which arises only when logical (unformatted) text is being moved, is the need to move language information with the moving text. This is needed even if the text being moved is in the document language since this may not be the current language at the point to which it moves. Thus the data-type for 'logical stuff being moved' must be the text and a language-label (describing its language).

Formatting special regions

A problem that affects the formatting of all special regions is how to specify the language to be used and the effective level of language tags contained within the special region. It is not possible to simply extend the THD and PAM from the main part of the document since these assume that the nesting of the language tags in the logical document is faithfully represented in the formatted document. This is very clearly not the case with regions such as floats or end-notes which appear visually in totally unrelated parts of the document. It is also not true for paragraphs within tables since these can be, logically, paragraphs within paragraphs, and our classification of language tags into types does not allow for this.

[5] Such details can have large effects on the efficiency of the implementation, thus we are being cautious here.

One possible solution to this problem is to allow the specification of a local PAM for each type of special region. This requires:

- a method to set the starting-language for the region;
- the specification of a local PAM for the region.

The disadvantage of this solution is its inherent complexity: for each special region the designer of a document class needs to specify a full mapping of all language-related actions to the tag hierarchy (the local PAM). Since the numbers of both the special regions and the language-related actions are potentially unlimited, this would result in either a very complex set-up mechanism or the use of general defaults (e.g., the local PAM nearly always inherits from the global document PAM) in which case the solution is unnecessarily complicated.

A practical solution

A simpler solution is to use the PAM from the main document but to allow the specification, for each type of special region, of how the information from the PAM is used. This would be done by specifying the following:

- a method to set the starting-language for the region;
- the actual initialisation-level (init-level) for the change to this starting language;
- the effective level (inner-level), as far as imbedded tags are concerned, of this change to the starting-language for the region.

We now give an expanded description of these items.

Starting language In the case of special regions that receive unformatted text the starting-language will directly affect only the text generated by the region's tags themselves as each bit of received text will carry its own language label (see the section entitled "Special Regions"). In the case of regions that move after being formatted it defines the default language used when formatting this region.

Initialization At the start of the region, actions are executed as if the region started with a tag whose level (in the THD, i.e., a pair of nodes) is init-level using this starting-language. This results in setting parameters to values suitable for that starting-language whilst allowing for a region to move to a different visual context.

Inner processing Within the region, language tags are processed as if the region started with a tag whose level (in the THD) is inner-level (inner-level must be at least as deep[6] as init-level in the THD). This allows finer control over the subset of actions executed at imbedded language tags.

Interfaces for the Rendering Model

The following interfaces will be provided for use by writers of class and package files:

- specifying the THD (this will probably be fixed, at least in the first version);
- specifying entries in the PAM;
- specifying entries in the LAT;
- specifying explicitly that a language-command (i.e., parameter-group) will potentially be used by the current package or class;[7]
- specifying the starting-language and init/inner levels for special regions;
- handling language information for moving text.

In addition to the new commands and environments outlined in the section entitled "Document interfaces", the following interfaces will be provided for use in documents (the first two must be in the preamble):

- specifying the document-language;
- specifying all the languages used in a document;
- possibly an interface for overwriting the starting language of a particular special region

The second item above is not strictly necessary as the information can be obtained by processing the document; however, a large saving of time and space can be made if the full list of languages actually used is specified in the preamble.

[6] An alternative model would be to also allow inner-level to be one less than init-level. This would mean that language tags within the special region are acting as language changes on the same level as the starting language of the region.

[7] These declarations allow the local customizations for all language actions to be stored in one place (e.g., PAM or LAT modifications); the system can then select only those that are actually needed for the current document.

New font tools for TEX

Werner Lemberg
Kleine Beurhausstr. 1
D-44137 Dortmund
Germany
a7971428@unet.univie.ac.at

Abstract

The following font tools will be described in this paper:

- VFlib, the *Vector Font Library*, which has been written by Kakugawa Hirotsugu (角川 裕次). This library together with an included suite of demo applications is aimed to provide a uniform interface for accessing fonts in many different formats. With the help of configuration files it is possible to access fonts with other encodings, to define new (virtual) fonts, and others. Using VFlib as a basis for `dvi` drivers, it will be no longer necessary to have `pk` files for any font format except for Metafont.
- FreeType, developed by David Turner, Robert Wilhelm, and the author of this paper. This is a platform-independent library to render bitmaps from TrueType fonts. What makes it different from other freely available TrueType tools is a TrueType interpreter to process hints.
- `ttf2pk` and `hbf2gf`, maintained resp. written by the author. Both programs are part of the CJK package and are used to convert CJK TrueType and HBF fonts into TEX fonts.

Introduction

None of the tools are really new, but until now they either have not been presented to the (English) TEX community or have been hidden in other packages. Most of them are work in progress, and there is a good chance that this paper is already out of date when it is printed.

All of them have a CJK[1] background more or less; Kakugawa originally wanted easy Japanese font support for `dvi` drivers, my two programs are still only useful for CJK fonts, and FreeType was also soon extended to manage CJK TrueType fonts. Nevertheless, work is going on to internationalize the tools, making them useful for a broader audience.

In the bibliography you can find the locations from which to download the packages.

VFlib

Until recently documentation was only available in Japanese (see [2]) — the main reason why this great tool is virtually unknown outside of Japan. Now a translation into English has been made; this quote from `basic.txt` tells what VFlib is:[2]

[1] Chinese/Japanese/Korean

[2] In general, I'm following the documentation files very closely, omitting not so important or too technical details.

Today, we have many font files and many different font file formats. When we need software to display or print characters which does not depend on a windowing system and/or an operating system, we must write interface routines for accessing font files in each application software again and again. To do this, programmers must have knowledge of font file formats; it will be a hard task for programmers if the number of font formats that an application software supports becomes large.

VFlib is a font library written in C providing several functions to obtain bitmaps of characters. VFlib hides the font format of font files and provides a unified API for all supported font formats. Thus, programmers for application software need not have knowledge on font file formats. Instead, any software using VFlib can support various font file formats immediately.

[...]

Currently, VFlib supports the following font file formats: PCF, BDF, HBF, TrueType, GF, PK, TFM, *ShotaiKurabu* (書体倶楽部,

a vector font format for Japanese Kanji),[3] and JG (another vector font format for Japanese Kanji). Especially, VFlib can be used as a font module for drivers/previewers of dvi files (TEX/LATEX); part of this package is a sample dvi previewer for X Windows written with only 400 lines of C code.

VFlib stands for *Vector Font library* — earlier versions of the library have supported Japanese vector fonts only, hence the name.

Basic concepts. The VFlib module consists of two parts: the library itself which must be linked to the application program, and a font database file (called 'vflibcap' since the file format resembles the format of a termcap database; see below for an example).

Font classes and font drivers. VFlib can handle multiple font file formats. Reading a font file is done by an internal module in VFlib corresponding to the font file format. This internal module is called a 'font driver'. Service units provided by font drivers are called 'font classes'. From an end-user's point of view, font formats are distinguished by the names of font classes. Font drivers themself are not visible for end-users.

Some font drivers may not read font files on disk; they may generate glyphs and outlines by internal computation only. In addition, some font drivers may return glyphs which are obtained as glyphs by another font class.

Font names and searching. In VFlib, a font is specified by a 'font name' on opening. First, VFlib checks if the font name is given in vflibcap or not. If the font name is found, VFlib reads the description for the font in vflibcap, invokes a font driver corresponding to the font class name and opens the font file.

If the font name is not given in a vflibcap file, a font searching mechanism is invoked. Since there are so many font files for X Window and TEX, this feature has been introduced to avoid writing an entry for each font file. Various font drivers will be called to see whether the font can be opened; a list of font drivers for font searching is given in the vflibcap file.

Fonts described in a vflibcap file are called 'explicit fonts' and fonts that are searched by the font search feature are called 'implicit fonts'.

For TEX fonts, the kpathsea library will be used for searching.

The vflibcap database. Each (virtual) font as provided by VFlib has its inherent information on point size, pixel size, and resolution of the target device. In addition to these font metrics are defined for each glyph.

Some font file formats do not have such concepts; in this case, missing information either is given in a vflibcap file or the specific font driver provides default values. For instance, a TrueType font is a vector font and is not restricted to a certain point size and resolution of the target device (since vector fonts can be scaled to any size). Another example is the ShotaiKurabu font format which does not have font metric information at all: a font driver for this font format generates virtual font metrics using the data given in a vflibcap file.

Here a small excerpt of a vflibcap file suitable for Japanese TEX (JTEX); omissions are indicated with three dots.

```
VFlib-Defaults:\
  :implicit-font-classes=ascii-jtex-kanji,\
                          gf/pk:\
  :extension-hints=pk=ascii-jtex-kanji,\
                   gf=ascii-jtex-kanji,\
                   pk=gf/pk,\
                   gf=gf/pk,\
                   .ttf=ttf:\
  :variables-default-values=\
    $TeX_KPATHSEA_PROGRAM=\
      /usr/local/teTeX/bin/xldvi:

TeX-Defaults:\
  :kpathsea-mode=ljfour:\
  :dpi=600:\
  :kpathsea-program-name=\
    $TeX_KPATHSEA_PROGRAM:

TrueType-Defaults:\
  :extension=.ttf:\
  :aspect=1:\
  :dpi=600:\
  :font-directories=\
    /dos/texmf/fonts/truetype/japanese:\
  :platform=microsoft:


mincho-jtex:\
  :font-class=ttf:\
  :font-file=uwjmg3.ttf:\
  :magnification=0.92:\
  :writing-direction=h:\
  :character-set=jisx0208_1983:\
  :encoding-force=sjis:

mincho-5pt:\
  :point-size=5:\
  :inheritance=mincho-jtex:

...
```

[3] Other transcription forms are *SyotaiKurabu* or *Syotai-Club*.

```
mincho-10pt:\
  :point-size=10:\
  :inheritance=mincho-jtex:


min:\
  :font-class=ascii-jtex-kanji:\
  :kanji-adjustment-file=\
    /usr/local/VFlib/ascii-jtex/ttf.adj:

min5:\
  :kanji-font=mincho-5pt:\
  :inheritance=min:

...

min10:\
  :kanji-font=mincho-10pt:\
  :inheritance=min:
```

The format of a capability file is somewhat strange: each entry must be formally on one line (which can be split with a trailing backslash). The name of a capability entry is the first name starting a line; all capability descriptions then follow in the format

:⟨*entry 1*⟩:⟨*entry 2*⟩:...:

A single colon ':' is equivalent to the construction ':⟨*whitespace*⟩:', making it convenient to break a line after the colon. Capability descriptions have the format

⟨*cap description*⟩=⟨*cap value*⟩

The first equal sign separates the description from the value (which can be e.g. a list containing equal signs too).

The capability entry 'VFlib-Defaults' defines global default values for VFlib. 'implicit-font-classes' specifies a list of font classes for implicit font search; the font class drivers are invoked in the order of that list for searching. 'extension-hints' gives an ordered list of pairs indicating which extension of an implicit font needs which driver to handle. In the above example there are e.g. two entries for files ending with pk: one for Japanese TeX and one for standard TeX. If the first driver fails, the second will be called. Finally, 'variables-default-values' gives a list of default values which can be overridden at run-time if supplied as an argument string to the initialization function of VFlib.

'TeX-Defaults' primarily gives initialization values for kpathsea. Users of web2c and teTeX should be quite familiar with the description names and its meanings.

All other entries are used to define a Japanese TrueType font as a font for JTeX. The final JTeX font is built from various layers, starting with the entry 'mincho-jtex' which defines the TrueType font name, the writing direction, the character set, etc. 'mincho-5pt' is an example of how to inherit font capabilities: only a point size declaration has been added. Then the 'min' base font class is specified, calling the 'ascii-jtex-kanji' font driver and using an adjustment file `ttf.adj` for all fonts of this class.[4] Note that in 'min' no fonts are defined — the concept is similar to object oriented languages where some base classes are defined on which virtual classes are built. With 'min5' or 'min10' the top level is reached, using all previously constructed font classes.

NFSS would not require size quantization of the font; it's easy to add a proper entry like this:

```
min-nfss:\
  :kanji-font=mincho-jtex:\
  :inheritance=min:
```

A font defined in this way can then be used at any size; VFlib would scale the font appropriately.

The VFlib API. The number of functions is small due to the identical interface for all font formats. Before opening any font you have to call `VF_Init` to initialize the library with a vflibcap file. Opening and closing of a font are handled with the functions `VF_OpenFont` and `VF_CloseFont` respectively. A glyph bitmap can be accessed in two ways. Either you specify bitmap sizes in pixels (`VF_GetBitmap2`), or you pass the resolution (in dpi) together with the point size as parameters (`VF_GetBitmap1`). Similar commands exist for getting outline (vector) data (`VF_GetOutline`) and for obtaining information on metrics (`VF_GetMetric1`, `VF_GetMetric2`). All the functions take the character code as a mandatory parameter.

Auxiliary functions are provided to free bitmap or metrics objects, to copy or scale bitmaps, and to 'dump' the glyph using ASCII characters to get similar output as `gftype`.

Finally, you can write your own font driver (to be installed with `VF_InstallFontDriver`). Font drivers must provide a small set of glyph manipulating functions,[5] and pointers to those functions are then passed to the VFlib engine. Using the function `VF_GetProp` it's easy to extract font-class-specific

[4] This file compensates the mono-width of Japanese glyphs with small offsets for certain character classes like CJK punctuation characters to improve typographical output. Character classes are a special feature of JTeX.

[5] With 'glyph' an empty box can be meant also e.g. for writing a `tfm` driver.

entries from the vflibcap file to control the new font driver.

Limitations and planned features. Many parts of the VFlib package are still undocumented, or documentation is only available in Japanese. For instance, VFlib contains a complete library for interpreting dvi files (with specials) which further simplifies the writing of dvi drivers. A suite of dvi previewers and dvi drivers is also included.

Currently, VFlib is limited to UNIX-like operating systems, but it should not be difficult to port it to other platforms because no real dependencies on UNIX features are built in.

Another useful tool yet to be written is a module or program for creating tfm files from the bitmap and vector fonts.

Planned for the near future are modules for processing PostScript fonts and TEX virtual fonts; support for Ω metrics files is already implemented, but without a vf module its use is rather academic.

FreeType

The report on FreeType (see [6]) will cover only the basics without going too much into detail — the final end-user API has not been defined yet. It is not really linked to TEX, and if you are not interested in how a rasterizer works, you should skip this section. Nevertheless, it is linked to typography, and the text presents some general principles of how outline glyphs will be handled to yield bitmaps.

FreeType is developed in a rather unusual way. The package provides the complete library code and some tools which demonstrate the use of the library in two programming languages, namely in C and in Pascal. We try to keep the library small (about 60 kByte if compiled for maximal speed with gcc); nevertheless it is highly portable since the C part is written in ANSI C, having only a few architecture dependencies which can be adjusted with a few global macro definitions.

The rasterizer. A rasterizer converts the vector data of a glyph into a pixel representation. In this short overview all complications (drop-out control, wrong contour direction, sub-banding of profiles etc.) are omitted.[6] This part of the FreeType engine is quite generic and could be adapted to, say, PostScript fonts too.

Glyphs as stored in a TrueType font (see [5] for a reference) consist of vectorial information (straight lines and Bézier curves of second order[7]) together with hinting instructions which move the points determining the glyph contours to device resolution dependent locations before rasterization.

We now assume that all point moving has been done, and that the x and y coordinates of the points are stored in a list together with a flag to indicate whether the point is *on* or *off* the curve. See Figure 1.[8]

A *scanline* is a pixel line in the target bitmap. An *outline*, also called *contour*, is a closed line that delimits an inner and an outer region of the glyph. The best way to fill a shape is to decompose it into simple horizontal segments, called *spans*. Spans are computed for each scanline. This is usually done from the top to the bottom of the shape, in a movement called *sweep* (see Figure 2). It's easy to see that there is typically more than one span per scanline. For each scanline during the sweep operation we need the horizontal (x) coordinates of the start and end points of all spans. These are computed before the sweep, in a phase called 'decomposition' which converts the glyph contours into *profiles*.

Profiles are sections of the contours which are either only ascending or only descending, i.e. *monotonic* in the vertical direction (we will also say y-monotonic). It can easily be deduced from Figure 3 that it is possible to resolve any contour into vertical profiles and horizontal lines (which are *not* part of a profile).

Each profile inherits the direction of the parent contour (this is necessary to decide whether a point is inside or outside of a contour, see below). Figure 4 shows that a contour can have multiple profiles. Profiles are also called 'edges' or 'edgelists' in other graphics libraries.

The rasterizer stores a profile as an array of x coordinates of the intersection points of the profile and the affected scanlines. To allocate a profile array without wasting memory we must know the height of that profile; with other words, we have to compute the vertical extrema (minimum and maximum). This can be done very easily for straight lines, but it is not trivial for Bézier arcs because

[6] The original document is raster.doc of the FreeType package, written by David Turner.

[7] PostScript fonts use third-order Bézier curves.

[8] A second-order Bézier curve (also called quadratic spline) is fully specified with the starting point of the curve, a control point usually off the line, and the end point of the curve. It is possible to have two consecutive off-points in the points list; in this case a virtual on-point between the two off-points will be constructed. The parametric form of a quadratic spline is

$$p(t) = (1-t)^2 p_1 + 2t(1-t)p_2 + t^2 p_3 \quad ;$$

t denotes a real number in the range $[0,1]$, p_1 is the start point, p_2 the control point, and p_3 the end point.

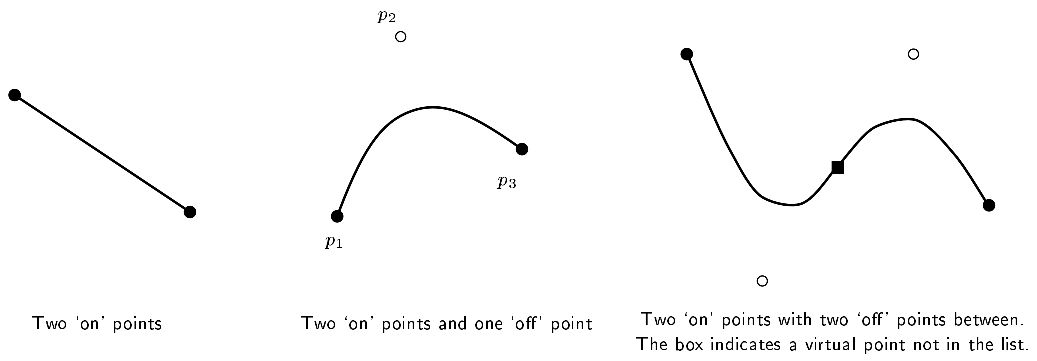

p_2
p_3
p_1
Two 'on' points
Two 'on' points and one 'off' point
Two 'on' points with two 'off' points between.
The box indicates a virtual point not in the list.

Figure 1: Possible 'on' and 'off' point combinations in a FreeType font

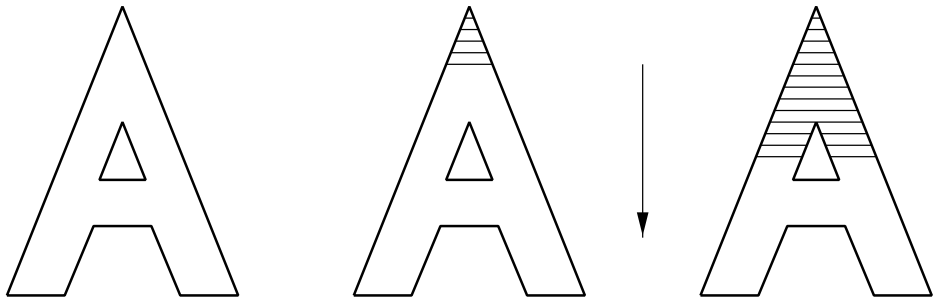


Figure 2: Filling a shape with spans. The arrow indicates the sweeping direction.

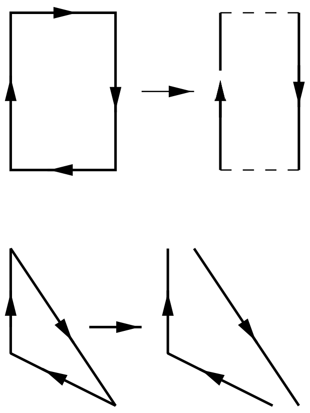


Figure 3: Decomposition of a contour into profiles

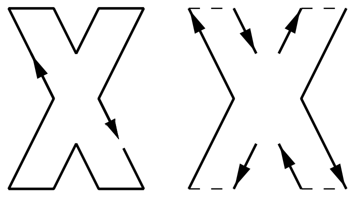


Figure 4: A contour with multiple profiles

they are not monotonic in the general case.[9] Nevertheless, a Bézier arc can be split into two subarcs with very little computation. Both subarcs are again Bézier arcs, and one of them is guaranteed to be y-monotonic. Look at Figure 5: p_i denote the points belonging to the original curve, q_i and r_i then define the subarcs (i being 1, 2, or 3). The following formulæ give the relationship between an arc and its subarcs:

$$q_1 = p_1; \quad q_2 = (p_1 + p_2)/2$$
$$r_3 = p_3; \quad r_2 = (p_2 + p_3)/2$$
$$q_3 = r_1 = (q_2 + r_2)/2$$

We stop if either all subarcs are monotonic or the subarcs become too small; in both cases we've found an extremum. This process is called *flattening*.

The next step is to compute all intersection points of profiles and scanlines. In the case of lines this is straightforward, but it is a little more complicated for splines. Fortunately we can use arc splitting again. Consider Figure 6. The horizontal lines represent scanlines, and a short segment of a profile is shown. If we continue splitting until each subarc

[9] A quadratic spline is y-monotonic if and only if the y coordinates of the points p_1, p_2, and p_3 are monotonic, i.e. ${p_1}_y \le {p_2}_y \le {p_3}_y$ or ${p_1}_y \ge {p_2}_y \ge {p_3}_y$. If ${p_1}_y = {p_2}_y = {p_3}_y$, the arc degenerates into a horizontal line.

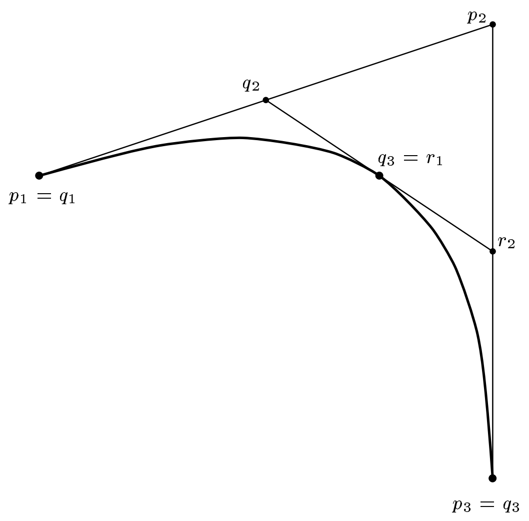


Figure 5: splitting a Bézier arc into two subarcs

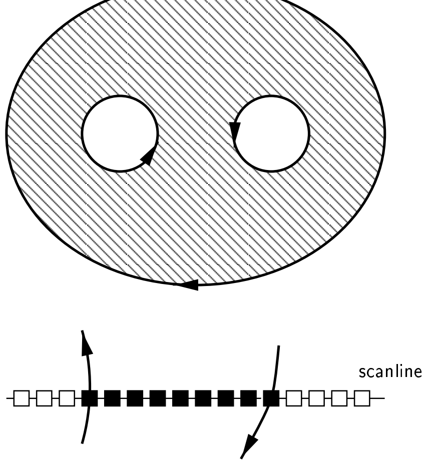


Figure 7: the interior of a glyph must always be to the right of the contour

crosses only one scanline, we can safely approximate the subarcs with straight lines for which the computational effort is minimal. Internally this has been realized with a Bézier arc stack; if the topmost arc can be replaced with a line, the intersection point is computed and the arc is popped off the stack. Otherwise the arc is popped off, then split, and the two new subarcs are pushed on the stack. This will be repeated until the stack is empty.

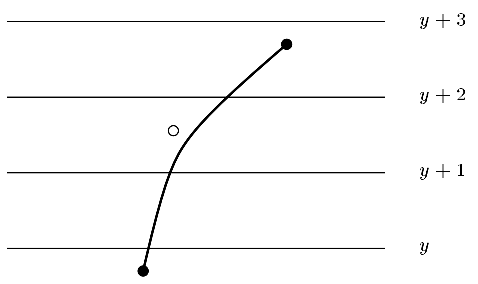


Figure 6: the stepping process: replacing sufficiently small subarcs with lines

We are done. The contours have been resolved into profiles, and the profiles have been decomposed into intersection points of the profiles and the scanlines. One last thing must be taken into account: how can we decide which side of the countour is interior and which is exterior? The TrueType specification defines that the interior is always on the right side of the contour (see Figure 7). Having an intersection point together with the contour direction, we can decide simply which pixels must be blackened.

The instruction interpreter. In figure 8 you can see what instructions basically do. To avoid dropouts or ugly shapes, points are moved to new (resolution dependent) locations before rendering, assuring that even for low resolutions good optical results can be computed. It has turned out that the TrueType specifications are often very fuzzy about certain instructions. Long debugging sessions with well hinted TrueType fonts were needed, comparing the results with rendered bitmaps of other (commercial) TrueType engines, to find out the undocumented behaviour of those instructions. Nevertheless, after mastering these obstacles, most glyphs are now rendered equally well with FreeType as with the rasterizers of Windows and the Mac.

To be added in the near future is instruction support for composite glyphs. Again the specifications are too fuzzy to allow a straightforward implementation without testing undocumented instruction properties — for instance, should the instruction code of the subglyphs be executed or only the instructions for the composite glyph? The TrueType specification says nothing about this problem.

Font tools from CJK

Both programs discussed in this section are part of the CJK package ([3]). They are specialized to CJK fonts, but work is going on to internationalize them. Script file skeletons of `MakeTeXPK` et al. are delivered with these utilities for on-the-fly font generation.

`ttf2pk`. It is currently a special tool for converting CJK TrueType fonts into `tfm` and `pk` files, but later

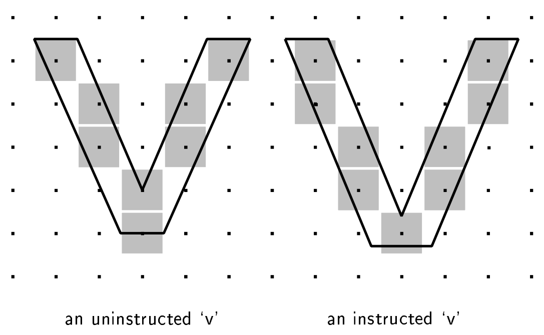


Figure 8: comparison between a hinted and an unhinted glyph

on it will use FreeType to process all kinds of TrueType fonts. The original author is Lin Yaw-Jen who modeled `ttf2pk` after `ttf2bmp` and `pbmtopk`.

`ttf2pk` takes a non-composite CJK TrueType font in Big 5, EUC,[10] or SJIS encoding and converts a certain contiguous subrange (at most 256 characters) into a `pk` and a `tfm` file. The program has a lot of command switches; a typical call looks like

```
ttf2pk ntukai01.pk ntukai01.tfm 600 1.0 \
       0xA140 256 -e Big5 ntu_kai.ttf
```

‘`ntukai01`’ is the first subfont of the ntukai font, ‘`600`’ is the resolution in dpi, ‘`1.0`’ a vertical scaling factor (for printers with different horizontal and vertical resolutions), ‘`0xA140`’ the first character code of the subfont, and ‘`256`’ the number of characters in the subfont. The switch `-e` selects the encoding of the TrueType font, and the last parameter is a full path to the TrueType font file.

Most of the other options not shown here have been inherited from `pnmtopk`; one parameter (‘`-r`’) has been added to rotate the glyphs by 90 degrees, enabling faked vertical typesetting with TeX.[11]

CJK subfonts as used in the CJK package are discussed in another paper of the proceedings ([4]).

`hbf2gf`. Similarly to `ttf2pk`, `hbf2gf` is used to split CJK bitmap fonts into subfonts. Its source code is written in CWEB; the format used by this tool is the *Hanzi Bitmap Font* format (HBF, see [1] for a complete reference). Basically the format consists of the bitmap files and a header file describing the font. Here an example of an HBF header file, describing a Chinese font with the character set CNS plane 7:

```
HBF_START_FONT 1.1
HBF_CODE_SCHEME CNS11643-92p5
FONT cns40st-5
SIZE 40 150 150
HBF_BITMAP_BOUNDING_BOX 40 40 0 -6
FONTBOUNDINGBOX 40 40 0 -6
STARTPROPERTIES 23
FONTNAME_REGISTRY ""
FOUNDRY "CBS"
FAMILY_NAME "Song"
WEIGHT_NAME "medium"
SLANT "r"
SETWIDTH_NAME "normal"
ADD_STYLE_NAME "fantizi"
PIXEL_SIZE 40
POINT_SIZE 400
RESOLUTION_X 75
RESOLUTION_Y 75
SPACING "c"
AVERAGE_WIDTH 400
CHARSET_REGISTRY "CNS11643.92p5"
CHARSET_ENCODING "0"
WEIGHT 19329
RESOLUTION 110
X_HEIGHT 34
QUAD_WIDTH 40
FONT_ASCENT 34
FONT_DESCENT 6
DEFAULT_CHAR 0x2121
ENDPROPERTIES
COMMENT "This HBF header file is in the"
COMMENT "public domain."
HBF_START_BYTE_2_RANGES 1
HBF_BYTE_2_RANGE 0x21-0x7E
HBF_END_BYTE_2_RANGES
HBF_START_CODE_RANGES 1
HBF_CODE_RANGE 0x2121-0x7C51 4040w5.bin 0
HBF_END_CODE_RANGES
COMMENT
COMMENT Rarely used characters defined by
COMMENT Ministry of Education of Taiwan,
COMMENT said to be disjoint from the
COMMENT previous planes.
COMMENT 8603 characters, 2121--7C51.
COMMENT
HBF_END_FONT
```

The syntax is very similar to the format of a BDF header (bitmap fonts used with X Windows); a small set of keywords (starting with ‘`HBF_`’; all others are BDF specific) have been added to accommodate the special needs of CJK files. As an example, the line

```
HBF_CODE_RANGE 0x2121-0x7C51 4040w5.bin 0
```

says that the file `4040w5.bin` contains glyphs with the character codes `0x2121`–`0x7C51`, starting at offset 0. ‘`HBF_BYTE_2_RANGE`’ specifies the valid range of the second bytes of the double byte font encoding

[10] EUC stands for Extended UNIX code; examples are Chinese GB, Japanese JIS, Korean KS encoding.

[11] Alas, only Big 5 encoding has both horizontal and vertical punctuation marks, but even here the set is not complete. For typographically satisfying results you need a font intended for vertical typesetting.

(see [3] for more details). Both keywords can appear more than once.

hbf2gf can be called in two 'modes': the first creates a complete set of subfonts for a particular HBF font, and the second computes the gf and tfm file of one subfont (to be used in MakeTeX... scripts). In both cases, a configuration file is used to avoid lengthy parameter lists.

We continue the example from above with an hbf2gf configuration file using the just defined CNS font (long lines are splitted and marked with a final backslash for clarification):

```
hbf_header \
 $TEXMF/fonts/hbf/chinese/cns40/cns40-5.hbf

comment \
 CNS plane 5 song 40x40 pixel font \
 scaled and adapted to 12pt

mag_x          1
design_size    12.0

x_offset       2
y_offset       -8

output_name    c5so12

checksum       123456789

dpi_x          300

pk_files       no
tfm_files      yes

pk_directory \
 $TEXMF/fonts/pk/modeless/chinese/c5so12/
tfm_directory \
 $TEXMF/fonts/tfm/chinese/c5so12/
```

Keywords must start a line; a line not starting with a known keyword is ignored. Environment variables can be specified with a starting dollar sign.

An important concept to understand hbf2gf configuration files is the difference between 'magnification' and 'scaling'.[12] The former denotes a scaling factor to reach the design size of the font (in the above example it is 1.0 to get 12 pt). Offset values (given in pixels) refer to this size. The latter then scales the font to its final size, indicated with dpi_x (and optionally dpi_y).

It is also possible to create fonts with slanted and rotated glyphs; additionally the next version will be able to produce Ω virtual fonts.

[12] The wording is a bit unfortunate because the meaning is different in TEX.

Conclusion

In a not too distant future the utilities and libraries described in this paper will more or less merge, providing a vital basis for handling fonts under Ω and TEX. With VFlib as the framework it will be possible to access already existing fonts in Unicode encoding regardless of the original encoding, create virtual fonts on the fly as needed for text processing with mixed writing directions (both horizontal and vertical), sophisticated space handling between fonts and much more.

With FreeType a new font world will be opened to TEX users working on UNIX like operating systems — after the integration of FreeType ttf2pk will be (hopefully) as useful as pstopk or gsftopk.

References

[1] Nelson Chin et al. Hanzi Bitmap Font (HBF) file format version 1.1. Available electronically from ftp://ftp.ifcss.org/pub/software/info/HBF-1.1.tar.gz, September 1994.

[2] Kakugawa Hirotsugu (裕次 角川). The VFlib package. Available from ftp://gull.se.hiroshima-u.ac.jp/1997.

[3] Werner Lemberg. The CJK package. Available from CTAN, language/chinese, 1997.

[4] Werner Lemberg. The CJK package for LATEX 2ε — multilingual support beyond babel. In *Proceedings of TUG 97*, July 1997.

[5] Microsoft corporation. TrueType 1.0 font files. Available electronically from ftp://ftp.microsoft.com/developr/drg/TrueType/ttspec.zip, November 1995.

[6] David Turner, Robert Wilhelm, and Werner Lemberg. The FreeType package. Available from ftp://ftp.physiol.med.tu-muenchen.de/pub/freetype, 1997.

The CJK package for LaTeX 2$_\varepsilon$ — Multilingual support beyond babel

Werner Lemberg
Kleine Beurhausstr. 1
D-44137 Dortmund
Germany
a7971428@unet.univie.ac.at

Abstract

With Mule (multilingual Emacs) you can write texts in multiple languages. This editor is especially designed to handle the various encodings and character sets of Asian scripts such as Big 5 and GB for Chinese, JIS for Japanese, etc. Even more, you can use multiple CJK character sets simultaneously which enables e.g. Chinese users to write simplified (*jiǎntǐzì* 简体字) and traditional Chinese characters (*fántǐzì* 繁體字) at the same time.

The CJK package is the analogue for LaTeX 2$_\varepsilon$ (to be run under standard TeX). Most of the CJK (Chinese/Japanese/Korean) encodings are implemented; an interface between Mule and LaTeX 2$_\varepsilon$ is provided by an output encoding filter for Mule. CJK is of course not restricted to Mule. Any editor/environment which is able to handle double byte encodings can be used.

If you restrict babel to (7-bit) ASCII as the input encoding it is possible to embed babel into CJK seamlessly. Using Mule's output filter, you even don't need enter LaTeX-specific accent macros; the accented characters will be converted automatically.

Included in the CJK package are auxiliary programs which can convert CJK TrueType and bitmaps fonts into pk files.

Introduction

TeX (and thus LaTeX) is an extremely flexible text formatting system which supports some multilingual features since the final version 3, mainly by allowing 8-bit input and fonts. But for CJK languages (Chinese/Japanese/Korean) this is still not enough because all encodings have more than 256 characters each.

The aim of the CJK package is to provide multibyte encoding support for LaTeX 2$_\varepsilon$ without any specific extensions of TeX. It contains modules for GB and Big 5 (Chinese), JIS and SJIS (Japanese), and KS (Korean) encoding, to name a few. In section "Unicode" on page 219 you'll find some notes on the pros and cons of Unicode in relation to TeX.

As far as I know there is only one freely available editor which is capable to display multiple character sets at the same time: Mule, the multilingual extension of Emacs.[1] A special output filter for Mule converts the internal encoding of Mule directly into something LaTeX can understand. See the section "The interface between Mule and CJK" on page 220 for more details.

Among other utilities two font converters called hbf2gf and ttf2pk are provided to convert CJK fonts into pk and tfm fonts; both programs are discussed in another paper of the proceedings ([8]).

The latest version of CJK can be found on CTAN in the directory language/chinese/CJK; various basic bitmap font packages are provided in fonts/CJK.

CJK encoding schemes

It is not possible to represent CJK character sets with one byte per character. At least two bytes are necessary, and most of the common CJK encoding schemes (GB, Big 5, JIS, KS, etc.) use a certain range for the first byte (usually 0xA1 to 0xFE or a part of it) to signal that this and the next byte represent a CJK character. As a consequence, ordinary ASCII characters (i.e., characters between 0x00 and 0x7F) remain unaffected.

[1] The next major releases of the various Emacs flavours (GNU Emacs and XEmacs) will merge the features of Mule back into Emacs.

encoding	1. byte	2. byte	3. byte
GB	0xA1–0xF7	0xA1–0xFE	—
Big 5	0xA1–0xF9	0x40–0xFE	—
JIS	0xA1–0xF4	0xA1–0xFE	—
SJIS	0xA1–0xFE	0x40–0xFC	—
KS	0xA1–0xFD	0xA1–0xFE	—
UTF 8	0xC0–0xEF	0x80–0xBF	0x80–0xBF
CNS	0xA1–0xFE	0xA1–0xFE	—

Figure 1: encoding schemes implemented in CJK

Some notes on the various encodings given in table 1:

- SJIS, also known as MS-Kanji, consists of two overlayed character sets: the so-called halfwidth Katakana (JIS X0201-1976, 1-byte characters encoded in the range `0xA1` to `0xDF`) and the (fullwidth) JIS character set (JIS X0208-1990, mapped to the remaining code points).
- Some encoding schemes (Big 5, SJIS) have gaps in the range of the second byte.
- UTF 8 (Unicode Transformation Format 8), also called UTF 2 or FSS-UTF, is a special representation of Unicode (resp. ISO 10 646). It uses multibyte sequences of various length, but only 2-byte and 3-byte sequences are implemented in CJK. ASCII characters will be used as-is — without this property it would be impossible to use UTF 8 with TeX.
- CNS is defined to have 16 planes with 94 × 94 characters. Currently 7 planes are assigned (CNS 1 to CNS 7, an eighth plane has been said to be under development).
- It's difficult to input Big 5 and SJIS encoding directly into TeX since some of the values used for the encodings' second bytes are reserved for control characters: '{', '}', and '\'. Redefining them breaks a lot of things in LaTeX; to avoid this, preprocessors are normally used which convert the second byte into a number followed by a delimiter character.

For further details please refer to [10]; Lunde discusses in great detail all CJK encodings which are or have been in use. See also section "Input encodings, output encodings, character sets" below.

The CJK package in detail

An example. Here a small lucullic text:

```
\documentclass{article}

\usepackage{CJK}
\usepackage{pinyin}

\begin{document}

\begin{CJK}{Bg5}{fs}

我很喜歡吃中國飯。

\Wo3 \hen3 \xi3\huan1 \chi1
\Zhong1\guo2\fan4.

I like to eat Chinese food
very much.

\end{CJK}

\end{document}
```

The result looks like this:

> 我很喜歡吃中國飯。
> Wǒ hěn xǐhuān chī Zhōngguófàn.
> I like to eat Chinese food very much.

This example shows that basically only two steps are necessary to write Chinese: loading CJK with `\usepackage{CJK}` and opening a CJK environment. `Bg5` selects the Big 5 encoding for Chinese written with traditional characters, `fs` the font to be used (in this case it is *fǎngsòngtǐ* 仿宋體). The procedure is slightly different if you use `cjk-enc.el` and will be described below in the section "The interface between `Mule` and CJK" on page 220.

`\usepackage{pinyin}` loads the pinyin package which is also part of CJK. It enables input of *pīnyīn* syllables, the transcription system of Chinese used in Mainland China (see also the section "The pinyin package" on page 221).

Basic concepts. To understand how CJK works behind the scenes some basic concepts have to be introduced.

Input encodings, output encodings, character sets. Since the arrival of NFSS input and output encodings are clearly separated in LaTeX. With CJK encodings the situation is a bit more complicated because some encodings are input and output encodings *at the same time*. To make things even more confused, they can form a character set also!

Consider as an example Big 5. This is a character set developed by software companies in Taiwan for Chinese written with traditional Chinese characters. It is common practice to describe CJK character sets in rows, usually (but not necessarily!) represented by the first bytes of the output encoding. For Big 5 we have 94 rows of 157 characters each. Not all rows are fully occupied: 94 × 157 = 14 758 characters are possible, but only 13 053 characters are really defined in the basic form of Big 5.[2] This character set can be split into three parts: 408 symbols (rows 1–3), 5 401 Level 1 hànzi (rows 4–38), and 7 652 Level 2 hànzi (rows 41–89). Level 1 contains the most frequent characters, Level 2 is an extension for rarely used characters occurring mainly in names.[3] With *hànzi* (漢字, *kanji* in Japanese, *hanja* in Korean) all ideographic glyphs derived from the Chinese script are denoted.

Row 1 of the Big 5 character set is mapped to `0xA1` as the first byte in Big 5 encoding, row 2 is mapped to `0xA2` etc.

You may ask: "Why only 157 characters per row? Table 1 would imply that 191 characters are available for each row." The answer is simple: due to historical reasons the range for the second byte of Big 5 input encoding (we are no longer talking about the Big 5 character set!) is split into two subranges: `0x40`–`0x7E` and `0xA1`–`0xFE`. And for convenience, almost all fonts providing the Big 5 character set can be accessed with an output encoding identical to the Big 5 input encoding.[4]

The counter example is the SJIS input encoding. Here we have two distinct Japanese character sets (JIS X0201 and JIS X0208) which will be accessed as two different fonts in the CJK package having a 1-byte and a 2-byte output encoding respectively.

The CJK macro layers. CJK makes all characters above `0x7F` active (except `0xFF`). The macro level assigned directly to the active characters is called 'binding' (stored in files with the extension `bdg`). The binding decides whether only the current byte, the next byte with the current byte or possibly the next two bytes together with the current byte represent a CJK character (example: JIS encoding has only 2-byte characters, SJIS has 1-byte and 2-byte characters).

The next level chooses the encoding of the CJK font and the output encoding of the subfonts (called 'fontencoding'; see also the sections "Subfonts" and "CJK font definition files" below for further information); the corresponding macros are stored in files with the extension `enc`. Here the proper subfont together with some font offsets will be selected and passed as arguments to the next level.

Macros from the third and last level (stored in files with the extension `chr`) finally select the proper character, check whether it is a special character etc., and print it out.

User selectable are only the encoding and fontencoding, the other levels are chosen automatically.

'Preprocessed' mode. Big 5 and SJIS encoding can't be handled well within TEX due to some characters in the range of the encodings' second bytes which interfere with the TEX control characters '`{`', '`}`', and '`\`'. It is possible to redefine them (and the CJK package provides two environments, Bg5text and SJIStext, which exactly do that), but many commands of LATEX don't work inside of them. Another annoying fact is that second bytes smaller than `0x80` are affected by case changing commands, altering the CJK characters!

Thus I have decided to program small preprocessors written in C[5] to convert the encodings into a form which won't cause problems: the second byte of an encoding will be converted into its decimal equivalent, followed by `0xFF` as a delimiter character.

This approach works for all encodings except UTF 8. CJK simply checks the presence of the `\CJKpreproc` command inserted by the preprocessor at the very beginning of the output file: if it is defined, another set of macros connecting the 'binding' and 'fontencoding' level is used. If it is undefined, `\MakeUppercase` is disabled for Big 5 and SJIS encoding.

Subfonts. What has been said about output encodings in a previous section is not the truth. It's a "little white lie", to cite a famous computer scientist whose name I can't remember yet.[6] To make large CJK fonts work with TEX we must split them. I've chosen the most compact subfont layout, i.e., 256 characters per subfont, but due to the modular concept of CJK it was not difficult to support other subfont schemes (like *poor man's Chinese* which uses one subfont per leading byte).

The number of subfonts per font is large. A JIS encoded font for example needs 35 subfonts, a Big 5 encoded font even 55. See below the section

[2] See [10] for a complete description of the extensions to Big 5.

[3] Chinese names cause a great problem for electronic data processing since every year new characters are invented; this is quite common especially in Hong Kong.

[4] This has changed with the propagation of TrueType fonts where e.g. fonts with a Big 5 character set can be accessed as Unicode encoded *and* as Big 5 encoded, provided the font has mapping tables for both encodings. The same is true for the so-called CID PostScript fonts which also can use multiple mapping tables for a particular font.

[5] Former versions of CJK contained these preprocessors written in both TEX and C, but under `web2c` it is impossible to `\write` out real 8-bit characters larger than `0x7F` to a file; you will always get the `^^xx` notation which fails in verbatim environments.

[6] Nevertheless, it is the truth for Ω which can handle fonts with more than 256 characters.

"CJK font definition files" for the naming scheme of subfonts.

The interaction with NFSS. Today I'm suprised by myself how simple it has been to adapt NFSS to subfont selection; only one internal macro of the LaTeX 2ε kernel, namely `\pickup@font`, must be redefined to gain the additional subfont functionality needed by CJK.[7] Here is the version as implemented in `CJK.sty`:[8]

```
\def\pickup@font{
  \ifx\CJK@plane \@undefined
% old definition
    \expandafter\ifx\font@name \relax
      \define@newfont
    \fi
  \else
% CJK extension
    \expandafter
    \ifx\csname \curr@fontshape/\f@size/
                \CJK@plane\endcsname \relax
      \define@newfont
    \else
      \xdef\font@name{
        \csname \curr@fontshape/\f@size/
                \CJK@plane\endcsname}
    \fi
  \fi}
```

`\CJK@plane` is only defined inside of a CJK macro on the character level. It will contain the subfont plane to be used. The only purpose of the small extension in `\pickup@font` is to define or check a new `\font@name` by appending `\CJK@plane` to the name LaTeX would construct.

CJK font definition files. Font definitions for CJK fonts are basically similar to other fonts. The main difference is that you define *classes* of subfonts instead of a single font.

Subfont names. The default name of a CJK subfont is the font family name plus a running decimal number (normally consisting of two digits). Example: `b5ka1201`, `b5ka1202`, etc. Unicode encoded fonts have two running hexadecimal digits appended instead, and some Japanese and Korean fonts follow other naming conventions. For all declarations in font definition files you have to specify the font family only; in our example this would be `b5ka12`.

Size functions. A suite of additional font size functions which take care of the subfonts has been defined. Most of them have a 'CJK' prefix or postfix to standard LaTeX size function names (`CJK`, `sCJK`, `CJKsub`, ...) to indicate a similar behaviour. Additionally some CJK size functions have a 'b' postfix to select 'poor man's boldface', as I have called the emulation of bold fonts by printing a non-bold character thrice with small offsets.[9] Providing a boldface emulation proved to be necessary because most of the freely available CJK fonts are available in one series only.

Here a sample entry for a GB encoded pixel font:

```
\DeclareFontFamily{C10}{fs}{}
\DeclareFontShape{C10}{fs}{m}{n}{
    <-> CJK * gsfs14}{}
\DeclareFontShape{C10}{fs}{bx}{n}{
    <-> CJKb * gsfs14}{\CJKbold}
```

`\CJKbold` sets an internal flag to switch on boldface emulation.

NFSS font encodings. Since version 4 of the CJK package all NFSS font attributes are supported. To achieve that it was necessary to have an NFSS font encoding for each CJK encoding. Figure 2 shows some of the currently available font encodings; all encodings defined by CJK start with an uppercase 'C', followed by two digits.

language	environment	NFSS encoding
Chinese:	Bg5	C00
	GB	C10
	CNS1–7	C31–37
Japanese:	JIS	C40
	JIS2	C50
	SJIS	C40 (fullwidth)
		C49 (halfwidth)
Korean:	KS	C60 (hanja)
		C61 (hangul)
Unicode:	UTF8	C70

Figure 2: The correlation between CJK encodings, CJK fontencodings, and NFSS font encodings. Only the most important encodings are listed here.

Some CJK encodings need more than one NFSS font encoding, as can be seen in the table (not listed here is the support for HLaTeX fonts where *four* NFSS encodings are necessary); the first digit usually represents the CJK encoding, the second digit (except for CNS encoded fonts) the specific subfont layout.

[7] `\selectfont` will also be changed slightly; nevertheless, it is not for the subfont selection but rather for the support of boldface emulation for CJK fonts.

[8] Please note that almost all files in the CJK package start with `\endlinechar -1` to avoid percent signs at the end of a line if we must suppress the newline character. The other TeX macro code fragments in this document assume the same.

[9] An explanation for the very reason of having extra size functions for bold face emulation is beyond the scope of this (already too long) article. It can be found in the documentation files of CJK.

The macro which internally maps the CJK encoding to one or more NFSS font encodings also calls `\DeclareFontSubstitution` with an empty font name macro to fake the fall-back mechanisms of NFSS.[10]

Nevertheless, you never access the NFSS font encodings directly; this will be done automatically if you open a CJK environment or use a `\CJKenc` command (see below).

CJK commands and environments. In this section you will find some important CJK commands not discussed elsewhere in this paper.

Additionally to the CJK environment CJK* will be provided: the starred form suppresses spaces after a CJK character (using `\ignorespaces`) which don't appear in texts written with Chinese or Japanese as the main language. To 'switch' from CJK to CJK* without leaving the environment you can use the `\CJKspace` command (and `\CJKnospace` for the other direction).

To access a CJK character directly you can use the `\CJKchar` command; it takes the first and second byte (represented as a number) of the character code together with an optional encoding string as parameters. This is the most portable form since no input characters larger than `0x7F` are used.

Four commands will control encodings and font encodings: `\CJKenc`, `\CJKfamily`, `\CJKencfamily`, and `\CJKfontenc`. To change the encoding inside of a CJK environment, use `\CJKenc`. It will always use the font encoding for a certain encoding which has been selected with `\CJKfontenc`. To change the font family you have two alternatives: The first is to define a family for a specific encoding with `\CJKencfamily`. If this encoding is chosen, the family defined in this way will be taken. The second is to specify a family for all encodings with `\CJKfamily`, overriding all `\CJKencfamily` commands. You must explicitly say '`\CJKfamily{}`' to reactivate any font definitions done with `\CJKencfamily`. Here is a (hypothetical) example:

```
\CJKencfamily{GBt}{hei}
\CJKfontenc{JIS}{dnp}

\begin{CJK*}{Bg5}{fs}   % this is equal to
                        % \begin{CJK*}{}{}
                        % \CJKenc{Bg5}
                        % \CJKfamily{fs}
..Text in Bg5 fangsong..% c00fs.fd used
\CJKenc{GB}
..Text in GB fangsong.. % c10fs.fd used
\CJKfamily{kai}
..Text in GB kai..      % c10kai.fd used
\CJKenc{JIS}
..Text in JISdnp kai..  % c42kai.fd used
\CJKfamily{}
\CJKenc{GBt}
..Text in GBt hei..     % c20hei.fd used
\end{CJK*}
```

'dnp' is the abbreviation for Dai Nippon Printing (大日本印刷), a big printing company in Japan providing Japanese TeX fonts. 'GBt' stands for GB traditional encoding (GB 12 345) — as far as I know no GB 12 345 font is freely available.

In case you get overfull `\hbox`es caused by CJK glyphs the macros `\CJKglue` and `\CJKtolerance` will help. The former is used for Chinese and Japanese encodings, defining the glue between CJK glyphs. Its default value is set to `\hskip 0pt plus 0.08\baselineskip`. The latter makes sense for Korean; the default value for `\CJKtolerance` is 400.

Only a subset of the commands available has been introduced here. For a complete description please refer to the various CJK documentation files.

CJK typography rules

Some special CJK characters should not start or end a line, e.g. various kinds of parentheses and many interpunctuation characters. In Japanese there is additionally a set of Hiragana and Katakana characters which are not allowed to start a line (*kinsoku shori* 禁則処理); both are supported automatically by CJK (but can be controlled if necessary). The mechanism to achieve this is quite tricky: usually you have some breakable glue (using `\hskip`) between two consecutive CJK characters (with other words, you need intercharacter spacing for Chinese and Japanese). Every character will be checked against encoding specific lookup tables whether it is a character not allowed to start or to end a line. In the former case, the glue before the character must be made unbreakable, otherwise the glue after the character.

No space will be printed after or before a CJK interpunctuation mark in Japanese and Chinese, but in Korean spaces are used between words[11] and after interpunctuation marks (which are in most cases the same as in Western languages).

[10] The latest version of HLaTeX has introduced 'HFSS' (the 'H' stands for Hangul), a font selection scheme to be run in parallel with NFSS, basically having an identical interface. A lot of advantages can be gained, mainly higher speed and better error handling.

[11] which are written as a combination with Hanja and Hangul (한글), the Korean syllable script. No intercharacter glue is used but `\discretionary{}{}{}` instead; additionally `\tolerance` is increased to `\CJKtolerance`.

As explained above we have macros for all CJK characters. How can we test whether the previous character was a CJK character, maybe even a special one? It's impossible to use `\futurelet` or other such commands because this only works on TeX's macro level, not on horizontal lists which are eventually output to a `dvi` file.

The solution I've found finally uses `\lastkern` to check the amount of the last kern.[12] If the current character is an ordinary one, a kern of 1 sp is emitted (do you remember TeX's smallest unit?), if no break should occur between the current and the next character, a kern of 2 sp is used instead. Now the next character will be processed. If it is a CJK character, `\lastkern` is tested whether it is 1 sp or 2 sp, and the appropriate action is executed.

Another typographic rule in CJK text processing is the use of different spaces, depending on context. Between a CJK character and a non-CJK character (e.g. an English word to be cited or a number) only a space having the quarter width of a (fullwidth) kanji should be typeset, and between non-CJK characters the default space has to be used. This quarter space is called *shibuaki* (四分あき). Some Japanese standards define more sophisticated rules where to print which space, but even the simple problem with having two different space widths can't be solved automatically in standard TeX. There exist Japanese adaptations of TeX which handle this internally, but you can't use these programs with other CJK languages. Nevertheless, it is solvable within Ω; see [6] for a short discussion on this topic.

The only way to manage shibuaki is to insert them manually. For this purpose I've redefined the tilde character to insert a quarter space instead of an unbreakable space (this will be still available as `\nbs`, a shorthand for `\nobreakspace`). The command `\CJKtilde` activates it; here an example:

`中國飯\ food 中國飯`	中國飯 food 中國飯
`中國飯~food~中國飯`	中國飯 food 中國飯

'~' is defined as

```
\def~{\hspace{.25em plus .125em minus .08em}
```

The effect of '~' seems to be minimal, but in underfull boxes it is really an optical enhancement.

The *Chinese Encoding Framework* (CEF)

Christian Wittern, a former employee of IRIZ ([1]), now working at the University of Göttingen, has developed CEF. Its primary aim is to access seldom used CJK encodings (most notably CNS) in a platform independent way using SGML macros of the form '`&`<*encoding*>`-`<*code*>`;`'. Examples for valid encoding values are 'C3' for CNS plane 3, 'C0' for Big 5, 'U' for Unicode; the code is given as a hexadimal number (see [15] for a detailed description).

He has also developed *KanjiBase for Windows*, an input tool for CEF which accesses a large CJK character database (which I consider the very heart of the whole system). It is also described in [15].[13]

A small example from an old Zen text (*The Records of Zhàozhōu* 趙州真際禪師語錄, taken from the IRIZ ZenBase CD 1) shows how it works:

```
0304a12
古尊宿語錄卷第十三。
趙州真際禪師語錄并行狀卷上〔南嶽下四世嗣南泉願〕。
師即南泉門人也。俗姓郝氏、本曹州郝鄉人也、諱從
諗。鎮府有塔記云、師得七百甲子歟。值武王微沐、避
地岨崍、木食草衣、僧儀不易。師初隨本師行&C3-3847;到南
泉。本師先人事了、師方乃人事。南泉在方丈內臥次、
見師來參、便問、近離什麼處。師云、瑞像院。南泉云、還
```

And here the same text with the appropriate CNS character:

```
0304a12
古尊宿語錄卷第十三。
趙州真際禪師語錄并行狀卷上〔南嶽下四世嗣南泉願〕。
師即南泉門人也。俗姓郝氏、本曹州郝鄉人也、諱從
諗。鎮府有塔記云、師得七百甲子歟。值武王微沐、避
地岨崍、木食草衣、僧儀不易。師初隨本師行脚到南
泉。本師先人事了、師方乃人事。南泉在方丈內臥次、
見師來參、便問、近離什麼處。師云、瑞像院。南泉云、還
```

The punctuation marks have been inserted by the editors and are not present in the original text.

CJK provides a small preprocessor to convert CEF macros into `\CJKchar` macros.

Unicode

Characters encoded in Unicode can't be used directly with TeX because the encoding is 16-bit wide. Instead, you have to use UTF 8 (see table 3 for the relationship between Unicode and UTF 8).

This multibyte representation has some important advantages: it is completely transparent for ASCII characters, you can always find the beginning of a multibyte sequence because the leading byte is unambiguously defined, and it is "reasonably compact in terms of number of bytes used for encoding", to cite from appendix A.2 of [13].

[12] Un Koaung-Hi (殷光熙), the author of HLaTeX ([12]) for Korean, uses a different solution: he modifies the space factor of specific characters to indicate a break point.

[13] You can find the latest version in [14] which also provides some online access and conversion.

Unicode	UTF 8		
	byte 1	byte 2	byte 3
$00000000\ 0b_6b_5b_4b_3b_2b_1b_0$ (U+0000 – U+007F)	$0b_6b_5b_4b_3b_2b_1b_0$	—	—
$00000b_{10}b_9b_8\ b_7b_6b_5b_4b_3b_2b_1b_0$ (U+0080 – U+07FF)	$110b_{10}b_9b_8b_7b_6$	$10b_5b_4b_3b_2b_1b_0$	—
$b_{15}b_{14}b_{13}b_{12}b_{11}b_{10}b_9b_8\ b_7b_6b_5b_4b_3b_2b_1b_0$ (U+0800 – U+FFFF)	$1110b_{15}b_{14}b_{13}b_{12}$	$10b_{11}b_{10}b_9b_8b_7b_6$	$10b_5b_4b_3b_2b_1b_0$

Figure 3: The UTF 8 representation of Unicode

CJK supports the whole Unicode range, not only the CJK part (provided you have fonts available), but I think that only the CJK range of Unicode makes sense with TEX under normal circumstances. The main reason is that you can have neither kerning nor automatic hyphenation if two adjacent characters come from different fonts, and this is almost inevitable because the number of (precomposed) latin characters with diacritics exceeds 700, and composition doesn't help either since the usage of `\accent` prevents kerning and hyphenation...

Another reason is that most of the currently available Unicode encoded CJK fonts provide one glyph shape per code point. However, this is not enough for typographically correct output. In figure 4 you can see the same character in three different shapes. Ken LUNDE shows in [9] that for a Unicode CJK font which really satisfies Chinese, Japanese, and Korean users about 40% more glyphs than code points are needed — this would further increase the number of TEX subfonts to be accessed simultaneously because the order of characters (or glyphs) in the CJK section of a Unicode font does not follow the character frequency but rather the order in a famous Chinese dictionary (*Kāngxī zìdiǎn* 康熙字典).

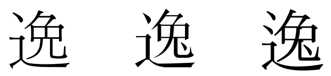


Figure 4: The Unicode character U+9038 in Japanese, Chinese, and Korean form (from left to right)

The interface between Mule and CJK

HANDA Ken'ichi (半田剣一), the main author of Mule, constructed a Lisp code frame for an interface between Mule and CJK which I filled with the needed values. It can be integrated into AUCTEX ([11]) without any great problems, making it a very convenient multilingual environment for LATEX.

The interface (stored in the file `cjk-enc.el`) has the form of a Mule output encoding. This means that you load a file into a Mule buffer, change the name of the buffer to the target file name, select 'cjk-coding' as the output encoding and save the file.

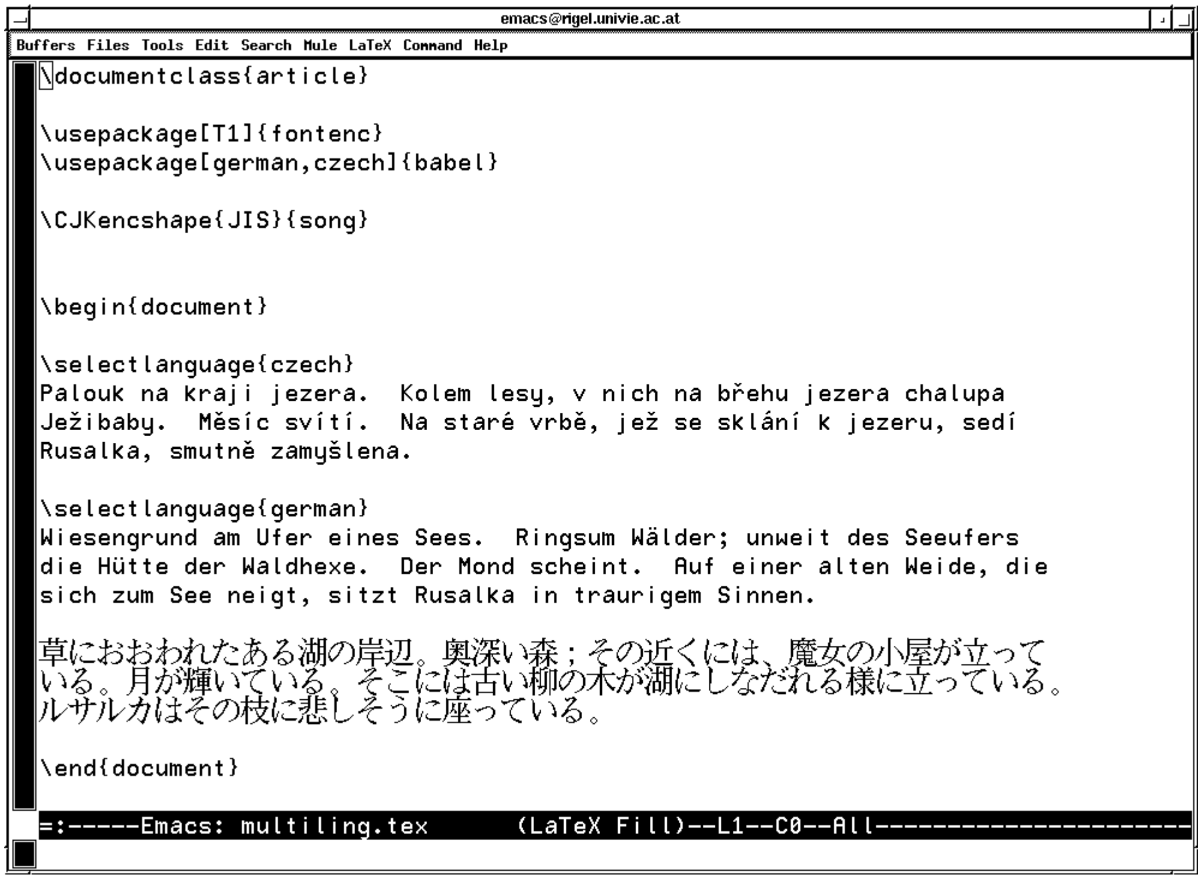
Look at figure 5. It shows a small multilingual example where Japanese is mixed with German and Czech.[14] As you can see, babel ([2]) is used for the two European languages; a CJK environment together with proper encoding switches is inserted automatically by the output encoding; accented characters are translated into LATEX macros also without any additional work.

It's not necessary to open a CJK environment inside of the document (it can even cause errors). Nor is it necessary to load the CJK package itself. The output filter uses `\RequirePackage` to load CJK and `\AtBeginDocument` to start a CJK environment with empty arguments; proper `\CJKenc` macros are inserted immediately before an encoding change. The most convenient way to specify CJK font shapes is then to use `\CJKencshape` in the preamble.

Now consider the word 'Dvořák' as an illustration of the hidden mechanism of the interface. Mule's internal representation of this word is '`Dvo ^^82^^f8^^82^^e1k`' (`^^xx` denoting real 8-bit values); `^^82` is a leading byte representing the Latin-2 *character set*, `^^f8` and `^^e1` are the 'ř' and 'á' in Latin-2 *encoding*.[15] After applying the 'cjk-coding' output encoding (as defined in `cjk-enc.el`), the Czech name looks like this: '`Dvo^^8051^^ffr^^ff ^^8020^^ffa^^ffk`'. The active character `^^80` has basically the following definition (in `MULEenc.sty`

[14] The text has been taken from a synchronoptical translation of the libretto of Antonín Dvořák's opera *Rusalka*. The left column is printed in Czech (the original language), the middle column in German, and the right column in Japanese.

[15] This is a bit sloppy. To speak correctly I had to say that the right-hand part of Latin Alphabet Nr. 2 (as defined in ISO 8859/2 and registrated as IR 101 in [4]) is mapped to the GR (Graphic Right, `0xA0` to `0xFF`) area. See [3] for a concise description of the terms necessary to understand Mule's internal and external code representations.



```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[german,czech]{babel}

\CJKencshape{JIS}{song}


\begin{document}

\selectlanguage{czech}
Palouk na kraji jezera.  Kolem lesy, v nich na břehu jezera chalupa
Ježibaby.  Měsíc svítí.  Na staré vrbě, jež se sklání k jezeru, sedí
Rusalka, smutně zamyšlena.

\selectlanguage{german}
Wiesengrund am Ufer eines Sees.  Ringsum Wälder; unweit des Seeufers
die Hütte der Waldhexe.  Der Mond scheint.  Auf einer alten Weide, die
sich zum See neigt, sitzt Rusalka in traurigem Sinnen.

草におおわれたある湖の岸辺。奥深い森；その近くには、魔女の小屋が立って
いる。月が輝いている。そこには古い柳の木が湖にしなだれる様に立っている。
ルサルカはその枝に悲しそうに座っている。

\end{document}

=:-----Emacs: multiling.tex      (LaTeX Fill)--L1--C0--All--------------------
```

Figure 5: A screen snapshot of `Mule`

which will always be `\input` at the very beginning of the output file):

```
\def^^80#1^^ff#2^^ff{...}
```

The first parameter is an index to an accent macro, the second is the letter (or character macro) to be modified. A number is used as the index; this has the advantage that it works in verbatim environments equally well as in case modifying commands. After expanding the macros `\mule@51` and `\mule@20` (defined with `\csname`) are called which finally expand to `\v` and `\'`. Great care has been taken to assure that really only expansion occurs to retain kerning.

Besides the common Latin and CJK character sets the `Mule` interface supports Vietnamese ([7]); it is planned to extend it to Thai and Russian soon.

Other tools

The pinyin package. This style file (which can be also used with plain TeX) enables the input of pīnyīn syllables with tones. An example of its usage was given on page 215.

Some additional notes:

- Mandarin Chinese has basically four tones (*sì shēng* 四聲) but sometimes it is referred to have a fifth, unstressed one. In the Chinese syllable script *zhùyīnfúhào* (注音符號) this fifth tone is indicated with a dot, but has no corresponding tone mark in pīnyīn. On the other hand, the first tone will be marked with a horizontal line in pīnyīn but remains unmarked in zhùyīnfúhào.

  With the pinyin package you can write e.g. '`\ma5`' to emphasize that you really mean an unstressed syllable — the result is equal to '`ma`'.

- In [16] a different approach to writing pīnyīn has been described; ligatures are used to compose vowels with tone marks, e.g. '`nu:v'e'r`' to get nǚ'ér[16] (女兒, the Chinese word for daughter). The advantage is the avoidance of any macros to produce the accented letters. The disadvantage is that you need virtual fonts to

[16] The quote character is used to denote the syllable boundary in ambiguous cases.

realize the ligatures which are not as easily installed as a macro package because you have to *create* them first if you want to use a new font.

The ruby package. To cite Martin Dürst[17] who wrote a proposal for ruby in HTML documents:

> Ruby are small characters used for annotations of a text, at the right side for vertical text, and atop for horizontal text, to indicate the reading (pronounciation) of ideographic characters.
>
> [...]
>
> The name *ruby* is the name of the 5.5 point type size in British terminology; this was the size most used for ruby.
>
> [...]
>
> Ruby are in most cases set at half the size of the main letters, resulting in a possible two ruby characters per main character, and taking up half of the width of the main characters. However, at least up to five ruby characters per main character are possible (an example is *u-ke-ta-ma-wa-ru* ($\overset{\text{うけたまわ}}{\text{承}}$る, to listen respectfully), and so various solutions, from leaving white space in the main text to having the ruby overlap the next characters of the main text, are possible (the latter is possible in Japanese especially because in many cases, the characters around an ideograph with ruby are syllabic, and therefore the assignment of ruby to main characters poses no problems for the reader).
>
> [...]
>
> Ruby are particularly frequent in Japanese, because of the way CJK ideographs are used in Japanese. Ideographs can have many different readings (pronounciations) because different readings were taken over from different regions of China and at different times when the characters where adopted in Japan. Also, these characters are used to write indigenous Japanese words, and many readings may be possible because the ideograph might cover many different concepts distinguished in the Japanese language. [...] The main use of ruby today is in magazines of all levels, and of course in educational material. Ruby are also used in educational material in China and Taiwan.
>
> In Japan, the term *furigana* (ふりがな) is also used instead of ruby. 'Furigana' is composed of the verb *furu* ($\overset{\text{ふ}}{\text{振}}$る, to attach, sprinkle, ...) and *gana* ($\overset{\text{が}}{\text{仮}}\overset{\text{な}}{\text{名}}$, either hiragana or katakana, one of the two Japanese syllabaries usually used for ruby).

[17] His email address is `mduerst@ifi.unizh.ch`.

The ruby $\overset{\text{うけたまわ}}{\text{承}}$ of the above citation has been input as `\ruby{承}{うけたまわ}`; the first parameter is the base character and the second the ruby itself.

To avoid lines sticking together the ruby package sets `\lineskiplimit` to 1 pt. It may be necessary to increase this value for larger font sizes.

Whether a ruby overlaps with the surrounding characters or not can be controlled with the overlap and nonoverlap options. There are a number of possibilities how ruby can interact with other CJK characters in both cases.

- The ruby has a smaller width than its base character: the behaviour is identical to an ordinary CJK character.
- The ruby has a greater width than its base character:
  - Overlapping ruby:
    - If the previous or next character is a CJK character (ordinary or punctuation), insert unbreakable glue between.
    - If the previous or next character is a ruby, handle both ruby as non-overlapping and insert unbreakable glue between.
    - A ruby at the beginning of a paragraph will be treated as if the nonoverlap option had been set. To force an overlapping ruby you have to start the paragraph with a `\leavevmode` command.
  - Non-overlapping ruby: if the previous or next character is a CJK character (ordinary or punctuation), insert unbreakable glue between.

`ruby.sty` introduces a third variation of a small kern (3 sp) to inform the next CJK or ruby macro that the previous character was an overlapping ruby with the ruby's width greater than its base character. The global variable `\ruby@width` then contains this width.

The interface to the koma-script package

One of the greatest deficiencies of the current implementation of the standard document classes is the inflexibility in handling captions which follow non-English conventions. But even English captions can cause trouble if they are non-standard. Consider "Chapter Two" vs. "Second Chapter": the former is supported, the latter isn't. With 'supported' I mean that it is not necessary to rewrite any internal LaTeX commands, and that even a novice user can change it. Besides this, many languages, especially in Asia, follow different conventions how to numerate sections, figures etc. A Chinese example: 'Chapter 5' is '第五章'. '第' (dì) is a prefix which converts the following Chinese number from a numeral into an ordinal, '第五' (dìwǔ) thus means 'the fifth'. '章' (zhāng) means chapter.[18]

Markus Kohm, the maintainer and developer of the koma-script package ([5]), has extended the package's document classes with a flexible hierarchical captioning model which consists of three levels, one level deeper than in standard LaTeX.

Level 1 are the well known standard macros `\figurename` etc. Language specific packages or options usually replace the English names with the right ones.

Level 2 is the modification of sectioning counters like `\thesection`. In the above example Chinese numbers should be used instead of Arabic digits.

Level 3 finally enables full control over the exact placement of spaces, counters, and other text in captions. All macros of this level have 'format' as postfix, e.g. `\chapterformat`; they are directly used by `\chapter`, `\section` etc.

A simplified definition for a Chinese chapter heading macro would be:

```
\newcommand\CJKnumber[1]{
  \ifcase#1\or
  一\or二\or三\or四\or五\or
  六\or七\or八\or九\or十\fi}

\newcommand\prechaptername{第}
\newcommand\postchaptername{章}
\renewcommand\thechapter{
  \prechaptername
  \CJKnumber{\value{chapter}}
  \postchaptername}
\renewcommand\chapterformat{\thechapter}
```

The CJK package supports this interface and provides caption files for Chinese, Japanese, and Korean. To activate, say `\CJKcaption{xxx}` inside of a CJK (or CJK*) environment; then the language module `xxx.cap` will be loaded. The names of the modules usually mirror the encoding, e.g., the Chinese caption file in Big 5 encoding is named `Bg5.cap`.

[18] Often it is written like '第 五 章' (第\ \ 五\ \ 章) in chapter headings, but the form without spaces is used in the table of contents.

Conclusion

The CJK package works best if you write a document in a *non*-CJK language as the main language. Many typographic features needed for native CJK script support can't be handled automatically due to limitations in TeX itself (the abovementioned shibuaki problem, vertical typesetting,[19] and others).

Another not yet mentioned problem is speed. Due to the many (sub)fonts you have to change the font for almost each character — if your document consists entirely of, say, Chinese, you have more than enough time to drink a cup of coffee to format 50 pages on a moderately fast computer.

The future for CJK multilingual text processing with LaTeX is definitely Ω, but until someone will have found time to provide CJK support (it is highly probable that this person is me again), it may not be the worst choice to use the CJK package meanwhile.

References

[1] Urs App, editor. *ZenBase CD 1.* International Research Institute for Zen Buddhism (IRIZ), Hanazono University (花園大学国際禅学研究所), Kyōtō, 1995. A CD-ROM with a large collection of Chinese Buddhist texts. Additionally it contains *KanjiBase for Windows*, the input tool for CEF, and a lot of other utilities useful for East Asian studies. Cf. `http://www.iijnet.or.jp/iriz/irizhtml/irizhome.htm`.

[2] Johannes Braams. An update on the babel system. *TUGboat*, 14(1):60–62, April 1993.

[3] European Computer Manufacturers' Association (ECMA). Standard ECMA-35. Character code structure and extension techniques. Available electronically from `ftp://ftp.ecma.ch` as the file `E035-PSC.EXE`, December 1994. This standard is completely identical to ISO-2022.

[4] International Organisation for Standardization (ISO). International register of coded character sets to be used with escape sequences, October 1994.

[5] Markus Kohm. The koma-script package. Available from CTAN, `macros/latex/contrib/supported/koma-script`, 1997.

[6] Werner Lemberg. Merging Babel and CJK under Ω. In *Proceedings of the First International Symposium on Multilingual Information Processing*, 1996. Hold March 25–26, 1996, in Tsukuba, Japan.

[19] Something which I've almost forgotten to say: CJK contains an experimental package for vertical typesetting with Big 5 encoded characters.

[7] Werner Lemberg. The vncmr package. Available from CTAN, `fonts/vietnamese/vncmr`, 1996.

[8] Werner Lemberg. New font tools for TEX. In *Proceedings of TUG 97*, July 1997.

[9] Ken Lunde. Creating fonts for the Unicode kanji set: Problems & solutions. In *Unicode Implementers' Workshop 6*. The Unicode Consortium, 1994. Hold September 8–9, 1994, in Santa Clara, California.

[10] Ken Lunde. Online companion to "understanding Japanese information processing". Available from `ftp://ftp.ora.com/pub/examples/nutshell/ujip/doc/cjk.inf`, 1996.

[11] Kresten Krab Thorup. GNU emacs as a front end to LATEX. *TUGboat*, 13(3):304–308, October 1992.

[12] Un Koaung-Hi (殷光熙). The HLATEX package. Available from CTAN, `language/korean`, 1997.

[13] The Unicode Consortium. *The Unicode Standard, Version 2.0.* Addison-Wesley, 1996. The latest versions of the various tables plus additional cross references can be found on `ftp.unicode.org`.

[14] Christian Wittern. The KanjiBase home page. Available from `http://www.gwdg.de/~cwitter`.

[15] Christian Wittern. The IRIZ KanjiBase. *The Electronic Bodhidharma (電子達摩)*, 4:58–62, June 1995. All articles in this journal are written both in Japanese and English.

[16] Wai Wong. Typesetting Chinese *pinyin* using virtual fonts. *TUGboat*, 14(1):8–11, April 1993.

DVIWindo [NEWTUGAD.DVI] page: 1

File Preferences Previous! Next! Unmagnify! Magnify! Fonts TeX

Y&Y TEX System

Powerful, fast, flexible TEX system for MicroSoft Windows

TeX System

Y&Y TEX System includes:

- DVIWindo previewer
- DVIPSONE PS driver
- Y&Y dynamic TEX
- Adobe Type Manager
- PFE Editor
- PostScript Type 1 fonts

Unique Features

- *Partial font downloading* for speed
- '*On the fly*' font reencoding
- *Dynamic memory allocation*
- Support for EPS and TIFF images
- Commercial grade, fully hinted fonts
- Windows 3.1, 95, NT 3.51, OS 2/2.1
- plain TEX, LATEX 2.09, LATEX 2ε etc.

Why Y&Y?

Mature products — years of experience with Windows, PostScript devices and scalable outline fonts. We understand and know how to *avoid* problems with Windows, 'clone' PostScript printers, ATM and 'difficult' fonts.

Y&Y — the experts in scalable outline fonts for TEX

®

```
WWW: http://www.YandY.com
email: sales-help@YandY.com
```

Y&Y, Inc. Tuttle's Livery, 45 Walden Street, Concord, MA 01742 USA — (508) 371-3286 — (508) 371-2004 (fax)