# TUGboat

Volume 26, Number 1 / 2005
Practical \TeX\ 2005 Conference Proceedings

## General Delivery

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Karl Berry / From the president</td>
</tr>
<tr>
<td>3</td>
<td>Barbara Beeton / Editorial comments</td>
</tr>
</tbody>
</table>

- Old TUGboat issues go electronic; CTAN announcement archives;
- Another \LaTeX\ manual— for word processor users;
- Create your own alphabet; Type design exhibition “Letras Latinas”;
- The cost of a bad proofreader;
- Looking at the same text in different ways: CSS on the web;
- Some comments on mathematical typesetting

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Barbara Beeton / Hyphenation exception log</td>
</tr>
</tbody>
</table>

## \LaTeX\n
<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Pedro Quaresma / Stacks in \TeX\</td>
</tr>
</tbody>
</table>

## Graphics

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Denis Roegel / Kissing circles: A French romance in MetaPost</td>
</tr>
</tbody>
</table>

## Software & Tools

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Tristan Miller / Using the RPM package manager for \LaTeX\ packages</td>
</tr>
</tbody>
</table>

## Practical \TeX\ 2005

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>Peter Flom and Tristan Miller / Impressions from \Prac\TeX\’05</td>
</tr>
</tbody>
</table>

## Keynote

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>Nelson Beebe / The design of \TeX\ and METAFONT: A retrospective</td>
</tr>
</tbody>
</table>

## Talks

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Peter Flom / \LaTeX\ fledgling struggles to take flight</td>
</tr>
<tr>
<td>56</td>
<td>Anita Schwartz / The art of \LaTeX\ problem solving</td>
</tr>
<tr>
<td>59</td>
<td>Klaus Höppner / Strategies for including graphics in \LaTeX\ documents</td>
</tr>
<tr>
<td>63</td>
<td>Joseph Hogg / Making a booklet</td>
</tr>
<tr>
<td>66</td>
<td>Peter Flynn / \LaTeX\ on the Web</td>
</tr>
<tr>
<td>68</td>
<td>Andrew Mertz and William Slough / Beamer by example</td>
</tr>
<tr>
<td>74</td>
<td>Kaveh Bazargan / Batch Commander: A graphical user interface for \TeX\</td>
</tr>
<tr>
<td>81</td>
<td>David Ignat / Word to \LaTeX\ for a large, multi-author scientific paper</td>
</tr>
<tr>
<td>85</td>
<td>Tristan Miller / Biblet: A portable Bib\TeX\ bibliography style for generating highly customizable XHTML</td>
</tr>
</tbody>
</table>

## News

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>99</td>
<td>Calendar</td>
</tr>
</tbody>
</table>

## TUG Business

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>Institutional members</td>
</tr>
</tbody>
</table>

## Advertisements

<table>
<thead>
<tr>
<th>Page</th>
<th>Author / Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>\TeX\ consulting and production services</td>
</tr>
<tr>
<td>101</td>
<td>Silmaril Consultants</td>
</tr>
<tr>
<td>101</td>
<td>Joe Hogg</td>
</tr>
<tr>
<td>101</td>
<td>Carleton Production Centre</td>
</tr>
<tr>
<td>102</td>
<td>Personal \TeX, Inc.</td>
</tr>
<tr>
<td>102</td>
<td>River Valley Technologies</td>
</tr>
<tr>
<td>102</td>
<td>Design Science, Inc.</td>
</tr>
<tr>
<td>102</td>
<td>The \LaTeX\ Companion, 2nd edition, by Frank Mittelbach et al.</td>
</tr>
<tr>
<td>103</td>
<td>Steve Peter</td>
</tr>
<tr>
<td>103</td>
<td>Cheryl Ponchin Training</td>
</tr>
<tr>
<td>103</td>
<td>MacKichan Software, Inc.</td>
</tr>
</tbody>
</table>
\TeX Users Group

\textit{TUGboat} (ISSN 0896-3207) is published by the \TeX Users Group.

Memberships and Subscriptions

2005 dues for individual members are as follows:
- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

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

\textit{TUGboat} subscriptions are available to organizations and others wishing to receive \textit{TUGboat} in a name other than that of an individual. Subscription rates: $85 a year, including air mail delivery.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both \TeX and the \TeX Users Group. For further information, contact the TUG office (office@tug.org) or see our web site.

\TeX is a trademark of the American Mathematical Society.

Copyright \textcopyright 2005 \TeX Users Group.

Copyright to individual articles within this publication remains with their authors, and may not be reproduced, distributed or translated without their permission.

For the editorial and other material not ascribed to a particular author, permission is granted to make and distribute verbatim copies without royalty, in any medium, provided the copyright notice and this permission notice are preserved.

Permission is also granted to make, copy and distribute translations of such editorial material into another language, except that the \TeX Users Group must approve translations of this permission notice itself. Lacking such approval, the original English permission notice must be included.

Printed in U.S.A.

\begin{center}
\textbf{Board of Directors}
\end{center}

Donald Knuth, \textit{Grand Wizard of \TeX-arcana}\textsuperscript{†}
Karl Berry, \textit{President}\textsuperscript{*}
Kaja Christiansen\textsuperscript{*}, \textit{Vice President}
Dave Walden\textsuperscript{*}, \textit{Treasurer}
Susan DeMeritt\textsuperscript{*}, \textit{Secretary}
Barbara Beeton
Lance Carnes
Steve Grathwohl
Jim Hefferon
Klaus Höppner
Ross Moore
Arthur Ogawa
Gerrée Pecht
Steve Peter
Cheryl Ponchin
Sam Rhoads
Philip Taylor
Raymond Goucher, \textit{Founding Executive Director}\textsuperscript{†}
Hermann Zapf, \textit{Wizard of Fonts}\textsuperscript{†}

\textsuperscript{*} member of executive committee
\textsuperscript{†} honorary

\begin{center}
\textbf{Addresses}
\end{center}

General correspondence, payments, etc.
\TeX Users Group
P.O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
\TeX Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

\begin{center}
\textbf{Electronic Mail (Internet)}
\end{center}

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

Submissions to \textit{TUGboat}, letters to the Editor:
TUGboat@tug.org

Technical support for \TeX users:
support@tug.org

Contact the Board of Directors:
board@tug.org

\begin{center}
\textbf{World Wide Web}
http://www.tug.org/
http://www.tug.org/TUGboat/
\end{center}

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

[printing date: August 2005]
TEX Users Group

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

Memberships and Subscriptions

2005 dues for individual members are as follows:
- Ordinary members: $75.
- Students/Seniors: $45.

The discounted rate of $45 is also available to citizens of countries with modest economies, as detailed on our web site.

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

TUGboat subscriptions are available to organizations and others wishing to receive TUGboat in a name other than that of an individual. Subscription rates: $85 a year, including air mail delivery.

Institutional Membership

Institutional Membership is a means of showing continuing interest in and support for both TEx and the TEx Users Group. For further information, contact the TUG office (office@tug.org) or see our web site.

TEx is a trademark of the American Mathematical Society.

Board of Directors

Donald Knuth, Grand Wizard of TEx-arcana†
Karl Berry, President∗
Kaja Christiansen*, Vice President
Dave Walden*, Treasurer
Susan DeMeritt*, Secretary
Barbara Beeton
Lance Carnes
Steve Grathwohl
Jim Hefferon
Klaus Höppner
Ross Moore
Arthur Ogawa
Gerree Pecht
Steve Peter
Cheryl Ponchin
Sam Rhoads
Philip Taylor
Raymond Goucher, Founding Executive Director†
Hermann Zapf, Wizard of Fonts†

∗member of executive committee
†honorary

Addresses
General correspondence, payments, etc.
TEx Users Group
P. O. Box 2311
Portland, OR 97208-2311
U.S.A.

Delivery services, parcels, visitors
TEx Users Group
1466 NW Naito Parkway
Suite 3141
Portland, OR 97209-2820
U.S.A.

Telephone
+1 503 223-9994

Fax
+1 503 223-3960

Electronic Mail

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

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

Technical support for TEx users:
support@tug.org

Contact the Board of Directors:
board@tug.org

World Wide Web

http://www.tug.org/
http://www.tug.org/TUGboat/

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

[printing date: August 2005]
This issue (Vol. 26, No. 1) combines regular articles and other material with the Practical \TeX 2005 conference proceedings.

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

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

\section*{Submitting Items for Publication}

Suggestions and proposals for \textit{TUGboat} articles are gratefully accepted and processed as received. We encourage submitting contributions by electronic mail to \texttt{TUGboat@tug.org}. Alternatively, please contact the TUG office.

The \textit{TUGboat} “style files”, for use with either \texttt{plain\TeX} or \texttt{LaT\TeX}, are available from CTAN and the \textit{TUGboat} web site above. We also accept submissions using ConT\TeXt.

As of this issue, submission of a new manuscript implies permission to publish the article, if accepted, on the \textit{TUGboat} web site, as well as in print. So, if you have any reservations about posting online, please notify the editors at the time of submission. (Background: until now, it has been \textit{TUGboat} policy to seek explicit permission for posting online, but we believe this has become unnecessary, leading primarily to articles never being posted, as well as being a time-consuming burden on \textit{TUGboat} staff. For several years, no author has refused permission to post online, so it seems reasonable to now assume this permission by default.)

\section*{\textit{TUGboat} Editorial Board}

Barbara Beeton, \textit{Editor-in-Chief}
Robin Laakso, \textit{Managing Editor}
Mimi Burbank, \textit{Production Manager}
Victor Eijkhout, \textit{Associate Editor, Macros}
Alan Hoenig, \textit{Associate Editor, Fonts}
Christina Thiele, \textit{Associate Editor, Topics in the Humanities}

\section*{Production Team}

William Adams, Barbara Beeton, Karl Berry, Mimi Burbank (Manager), Kaja Christiansen, Robin Fairbairns, Baden Hughes, Steve Peter, Michael Sofka, Christina Thiele

\section*{Other TUG Publications}

TUG is interested in considering additional manuscripts for publication. These might include manuals, instructional materials, documentation, or works on any other topic that might be useful to the \TeX community in general. Provision can be made for including macro packages or software in computer-readable form.

If you have any such items or know of any that you would like considered for publication, send the information to the attention of the Publications Committee at \texttt{tug-pub@tug.org}.

\section*{\textit{TUGboat} Advertising}

For information about advertising rates and options, write or call the TUG office, or see our web page \url{http://tug.org/TUGboat/advertising.html}.

\section*{Trademarks}

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

\texttt{METAFONT} is a trademark of Addison-Wesley Inc. \texttt{PostScript} is a trademark of Adobe Systems, Inc. \texttt{\LaT\TeX} and \texttt{\AMSTeX} are trademarks of the American Mathematical Society.
From the President

Karl Berry

For the first time since becoming president in summer 2003, I am writing this column not for a past year, but the present year! Which is to say, TUGboat is now just about up to date; if things go well, it will be fully caught up by the end of the year. A happy state of affairs, for which thanks are due to Barbara Beeton, Mimi Burbank, Robin Laakso, the rest of the TUGboat staff, and of course all the authors.

Also, because this issue is being published on time, it inaugurates the new policy, as announced in the last issue and elsewhere, of having TUGboat articles available online only to members for a year after publication (after that they will be open to all). The TUGboat information page (on the back of the title page) describes the policy in more detail.

In other news so far this year, perhaps the most visible item is that the TUG CTAN node, http://tug.ctan.org, has a new interface and visual design—check it out. The upgrade was possible because of new hardware that TUG was able to provide; so thanks to every member reading this. Also, many thanks to TUG director Jim Hefferon, who has maintained the site since 1999, his institution, St. Michael’s College in Vermont, for supporting Jim and providing the Internet connection, and of course the whole CTAN group for their continuing and amazing efforts.

The other technical news for TUG is that we upgraded the hardware for the tug.org server; again, thanks to member support. We expect the new machine to suffice for several years (just as the last one did). Ongoing thanks there go to TUG vice-president and system administrator Kaja Christiansen and her institution, Århus University in Denmark, for hosting it.

On the TUG administrative side, we’ve established a small working group to focus efforts on reversing the decline in TUG membership. The group is working on several projects, such as increased publicity for \TeX and TUG additional membership categories, and additional benefits. If you have any suggestions, or would like to help in this effort, please email tug-membership@tug.org, and thanks.

One small initiative along those lines is that as of this year, we increased the number of memberships available to institutional members, from seven to eight, thus providing a small discount. In the past, memberships were a way for institutions to “donate” to TUG, but times have changed.

We do have one new institutional member so far this year, MacKichan Software, Inc. (http://www.mackichan.com)—thank you! As always, a complete list of institutional members can be found elsewhere in this issue, and online at http://tug.org/instmem.html.

The last item regards Y&Y, a \TeX vendor of long standing, which had early and excellent support for outline fonts on the Windows platform, among many other features. As some of you may already know, the business was regrettably closed in 2003.

We were sorry to lose this significant part of the \TeX community.

However, Blenda Horn, the remaining principal of the company, has made a remarkably generous decision: to donate the Y&Y software to the scientific community. Thus, the product can continue, now as a volunteer effort. To that end, TUG will make the source files available under a free software license as soon as is practical.

Thank you, Blenda!

One additional note on this: Y&Y also distributed two font sets—Michael Spivak’s MathTime (full math complement to Times Roman), and Bigelow & Holmes’ Lucida. These fonts remain the property of their owners. You can obtain the MathTime fonts in various configurations through Personal \TeX, Inc. (http://www.pctex.com). For the Lucida fonts, TUG has negotiated the right to redistribute them with B&H, and we will make announcements when they are available.

As always, please don’t hesitate to contact me or the entire board (board@tug.org) with any issues or comments. Thanks for your support, and happy \TeXing.

Karl Berry

president@tug.org

Editorial Comments

Barbara Beeton

Old TUGboat issues go electronic

Regular posting of articles when published began with volume 16 (1995), so all issues after that should be present. However, owing to the wording of the copyright statement, for volumes 19–24 (1998–2003), only articles for which explicit permission has been received can be posted. Also, there will be a delay, beginning with this issue, of a year after printing, during which time only TUG members will have on-line access.

The missing issues will be scanned and posted as time permits.

If you find any problems with the posted material, or have not given permission to post an article from the “restricted” period, please notify the office (office@tug.org).

**CTAN announcement archives**

The archive of ctan-ann mail can be seen and the contents searched at [http://www.mail-archive.com/ctan-ann@dante.de/](http://www.mail-archive.com/ctan-ann@dante.de/). This collection begins with January 2005; earlier notices have been summarized in previous TUGboat issues in “The Treasure Chest”.

**Another \LaTeX{} manual — for word processor users**

Users of word processors often seem to have particular difficulty making the switch to \LaTeX{}, no matter how much they appreciate the better appearance of the output. A manual directed toward such potential users is available from CTAN. Written by Guido Gonzato, this manual presents the basics of preparing input with an editor, concentrating on structure rather than appearance, use of packages (including installation on one’s system), and many other topics. The manual is presented as a PDF file, with hyperlinks to many packages and tools mentioned in the text.

The manual can be found at [http://www.ctan.org/tex-archive/info/latex4wp/](http://www.ctan.org/tex-archive/info/latex4wp/).

**Create your own alphabet**

From the website [http://alphabet.tmema.org/](http://alphabet.tmema.org/):

“The Alphabet Synthesis Machine” is an interactive online artwork which allows one to create and evolve the possible writing systems of one’s own imaginary civilizations. The abstract alphabets produced by the Machine can be downloaded as PC-format TrueType fonts, and are entered into a comprehensive archive of user creations. The products of the Machine probe the liminal territories between familiarity and chaos, language and gesture.

The tools found here were created for the project “art:21”, “art in the twenty-first century” (a project of PBS), by Golan Levin, with Jonathan Feinberg and Cassidy Curtis. In addition to the downloadable software, the site contains example alphabets produced by visitors, an archive of user creations, and a bibliography of works on the history of writing, writing systems, and the (Latin) alphabet.

**Type design exhibition “Letras Latinas”**

The binennial type design exhibition “Latin letters” can be viewed online at [http://www.tipografica.com/letraslatinash/](http://www.tipografica.com/letraslatinash/). This event takes place simultaneously at several venues in Central and South America. The on-line exhibition includes fonts for both text and display. Although the text of the site is in Spanish, no translation is needed to appreciate and enjoy the samples shown. The exhibition was organized in Buenos Aires, Argentina, by Tipografía magazine. Font designers can register to submit their work for the next show via a form on the website.

**The cost of a bad proofreader**

In April, the French government was forced to destroy 162,000 copies of the EU constitution because the phrase “incoherent text” appeared on a page by mistake. (This occurred before the French referendum in May.) Proofreaders failed to notice this phrase in a footnote on a page which contained Article 1/33 of the constitution; it was apparently invisible on the screen when the document was read on line. A corrected version of the full 232-page text was printed at the cost of 74,000 euros. Whoever was responsible for inserting the text was not known.

The full report can be read at [http://news.bbc.co.uk/2/hi/europe/4421963.stm](http://news.bbc.co.uk/2/hi/europe/4421963.stm).

No matter how hard one tries, it seems that some typos always get through.

**Looking at the same text in different ways: CSS on the web**

Re-use or reformatting of the same text is a common theme in print. Here is a demonstration of what can be done using CSS to do the same for web pages: [http://www.csszengarden.com/](http://www.csszengarden.com/).

**Some comments on mathematical typesetting**

This quote, by Gottfried Leibniz, was contributed by Don Knuth, who found it in the library at the Institut Mittag-Leffler, near Stockholm.
From *Leibnizens mathematische Schriften*, edited by C. I. Gerhardt, Erste Abtheilung, Band III (1855), in a letter from Leibniz to Johann Bernoulli, 15 May 1696:

In notandis calculis ad usum typorum decrevi pro lineis vinculorum imponentur uti commatibus directis atque inversis in vim parenthesium: ita non interrupsum typorum series nec spatium ammittetur, et tamen omnia (ni fallor) accurate habeantur. Velim tamen prius Tuam audire sententiam. Exempli causa,

\[
\frac{a + b}{c} \quad \text{quod quinque typorum lineas minimum postulat, sic poterit scribi: } a + b : c \quad ; \quad e - f : g; \quad \text{possent tamen inversa commata omitti, scribique } a + b : c \quad ; \quad e - f : g \quad , \quad \text{quod et facere soleo et communiter sufficiere potest. Sed tamen designatio quasi par-
}

Another relevant note comes from *Acta Eruditorum* (Leipzig: 1708), 271; here is a translation from the Latin original, given by Florian Cajori on page 219 of his *History of Mathematical Notations*:

We hereby issue the reminder that in the future we shall use in these *Acta* the Leibnizian signs, where, when algebraic matters concern us, we do not choose the typographically troublesome and unnecessarily repugnant, and that we avoid ambiguity. Hence we shall prefer the parenthesis to the characters consisting of lines drawn above, and in multiplication by the parenthesis to the characters consisting of all means simply omit the comma; for example, in place of \(\sqrt{ab} + bb\) we write \(\sqrt{(ab + bb)}\) and for \(ab + bb \times c\) we take \(ab + bb, c\). Division we mark with two dots, unless indeed some peculiar circumstance directs adherence to the usual practice. Accordingly, we have \(a : b = \frac{a}{b}\). And it is not necessary to denote proportion by any special sign. For, if \(a\) is to \(b\) as \(c\) is to \(d\), we have \(a : b = c : d\). As regards powers, \(aa + bb\), we designate them by \((aa + bb)^m\); whence also \(\sqrt{aa + bb}\) becomes \(= (aa + bb)^{1/m}\) and \(\sqrt[\alpha]{aa + bb} = (aa + bb)^{\alpha/m}\). We do not doubt that all geometers who read the *Acta* will recognize the excellence of the Leibnizian symbols and will agree with us in this matter.

○ Barbara Beeton  
American Mathematical Society  
201 Charles Street  
Providence, RI 02904 USA  
bnb@ams.org

### Hyphenation Exception Log

Barbara Beeton

This is the periodic update of the list of words that *\TeX* fails to hyphenate properly. The full list last appeared in *TUGboat* 16, no. 1, starting on page 12, with updates in *TUGboat* 22, no. 1/2, pages 31–32, and *TUGboat* 23, no. 3/4, pages 247–248. This installment contains only exceptions identified since the last update.

In the list below, the first column gives results from *\TeX*’s `\showhyphens{...}`; entries in the second column are suitable for inclusion in a `\hyphenation{...}` list.

In most instances, inflected forms are not shown for nouns and verbs; note that all forms must be specified in a `\hyphenation{...}` list if they occur in your document. See the section below, “Converting this list into a list of hyphenation exceptions”.

Like the full list, this update has been subdivided into two parts: English words, and names and non-English words (including transliterations from Cyrillic and other non-Latin scripts) that occur in English texts.

Thanks to all who have submitted entries to the list. Since some suggestions have demonstrated a lack of familiarity with the rules of the hyphenation algorithm, here is a short reminder of the relevant idiosyncrasies. Hyphens will not be inserted before the number of letters specified by `\lefthyphenmin`, nor after the number of letters specified by `\righthyphenmin`. For U.S. English, `\lefthyphenmin=2` and `\righthyphenmin=3`; thus no word shorter than five letters will be hyphenated. (For the details, see *The \TeX*book, page 454.) This particular rule is violated in some of the words listed; however, if a word is hyphenated correctly by *\TeX* except for “missing” hyphens at the beginning or end, it has not been included here.
Some other permissible hyphens have been omitted for reasons of style or clarity. While this is at least partly a matter of personal taste, an author should think of the reader when deciding whether or not to permit just one more break-point in some obscure or confusing word. There really are times when a bit of rewriting is preferable.

One other warning: Some words can be more than one part of speech, depending on context, and have different hyphenations; for example, ‘analyses’ can be either a verb or a plural noun. If such a word appears in this list, hyphens are shown only for the portions of the word that would be hyphenated the same regardless of usage. These words are marked with a ‘*’; additional hyphenation points, if needed in your document, should be inserted with discretionary hyphens.

The reference used to check these hyphenations is Webster’s Third New International Dictionary, Unabridged.

Hyphenation for languages other than English
Patterns now exist for many languages other than English, including languages using accented alphabets. CTAN holds an extensive collection of patterns in tex-archive/language/hyphenation and its subdirectories.

Converting this list into a list of hyphenation exceptions
Werner Lemberg has created a script that will convert this article into a real hyphenation block that can be incorporated into a document either directly or by inputting a file. Many inflected forms will be included automatically, some evident in the printed version, but many included silently.

The script, hyphenex.sh, runs under Unix. It is posted on CTAN, along with its output for the current set of exceptions, in tex-archive/info/digests/tugboat/

The List — English words

<table>
<thead>
<tr>
<th>acronym</th>
<th>acronym</th>
</tr>
</thead>
<tbody>
<tr>
<td>au-ton-um-ber-ing</td>
<td>au-ton-um-ber-ing</td>
</tr>
<tr>
<td>au-tonomous</td>
<td>au-tonomous</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>buffer</td>
<td>buffer</td>
</tr>
<tr>
<td>chloromethane</td>
<td>chloromethane</td>
</tr>
<tr>
<td>core-la-tion(s)</td>
<td>core-la-tion(s)</td>
</tr>
<tr>
<td>core-li-gion-ist(s)</td>
<td>core-li-gion-ist(s)</td>
</tr>
<tr>
<td>core-op-sis</td>
<td>core-op-sis</td>
</tr>
<tr>
<td>core-spon-dent(s)</td>
<td>core-spon-dent(s)</td>
</tr>
<tr>
<td>coworker</td>
<td>coworker</td>
</tr>
<tr>
<td>crankcase</td>
<td>crank-case</td>
</tr>
<tr>
<td>crossover</td>
<td>crossover</td>
</tr>
<tr>
<td>cu-flink(s)</td>
<td>cur-flink(s)</td>
</tr>
<tr>
<td>cuz-tomiz-able</td>
<td>cuz-tomiz-able</td>
</tr>
<tr>
<td>cuz-tomize(s,d)</td>
<td>cuz-tomize(s,d)</td>
</tr>
<tr>
<td>dichloromethane</td>
<td>dichloromethane</td>
</tr>
<tr>
<td>ethane</td>
<td>ethane</td>
</tr>
<tr>
<td>flu-o-ro-car-bon</td>
<td>fluo-carbon</td>
</tr>
<tr>
<td>goether-mal</td>
<td>geo-ther-mal</td>
</tr>
<tr>
<td>grandun-cle</td>
<td>grand-uncle</td>
</tr>
<tr>
<td>hy-drother-mal</td>
<td>hy-dro-ther-mal</td>
</tr>
<tr>
<td>keynote</td>
<td>key-note</td>
</tr>
<tr>
<td>manslaugh-ter</td>
<td>manslaugh-ter</td>
</tr>
<tr>
<td>methane</td>
<td>methane</td>
</tr>
<tr>
<td>ni-tromethane</td>
<td>ni-tromethane</td>
</tr>
<tr>
<td>of-fline</td>
<td>of-line</td>
</tr>
<tr>
<td>of-flode(s,ed)</td>
<td>of-load(s,ed)</td>
</tr>
<tr>
<td>palette</td>
<td>palette</td>
</tr>
<tr>
<td>pipelin-ing</td>
<td>pipelin-ing</td>
</tr>
<tr>
<td>prewrap(ped)</td>
<td>pre-wrap(ped)</td>
</tr>
<tr>
<td>pro-gram-mable</td>
<td>pro-gram-mable</td>
</tr>
<tr>
<td>prom-i-next</td>
<td>prom-i-next</td>
</tr>
<tr>
<td>promis-cu-ous</td>
<td>promis-cu-ous</td>
</tr>
<tr>
<td>promis-sory</td>
<td>promis-sory</td>
</tr>
<tr>
<td>promise</td>
<td>promise</td>
</tr>
<tr>
<td>prowess</td>
<td>prowess</td>
</tr>
<tr>
<td>qu-a-sitriv-ial</td>
<td>qua-ui-sitriv-ial</td>
</tr>
<tr>
<td>rephrase(s,d)</td>
<td>rephrase(s,d)</td>
</tr>
<tr>
<td>rewrap(ped)</td>
<td>rewrap(ped)</td>
</tr>
<tr>
<td>subex-pres-sion</td>
<td>subex-pres-sion</td>
</tr>
<tr>
<td>taffrail</td>
<td>taffrail</td>
</tr>
<tr>
<td>tri-ethy-lamine</td>
<td>tri-ethyl-lamine</td>
</tr>
<tr>
<td>vi-sual(ly)</td>
<td>visual(ly)</td>
</tr>
</tbody>
</table>

Names and non-English words used in English text

<table>
<thead>
<tr>
<th>Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT-Pase</td>
<td>ATP-ase</td>
</tr>
<tr>
<td>Delaware</td>
<td>Del-a-ware</td>
</tr>
<tr>
<td>Du-ane</td>
<td>Duane</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>FreeBSD</td>
</tr>
<tr>
<td>Hadamard</td>
<td>Ha-da-mard</td>
</tr>
<tr>
<td>Har-alam-bous</td>
<td>Har-alam-bous</td>
</tr>
<tr>
<td>Jack-owski</td>
<td>Jac-kow-ski</td>
</tr>
<tr>
<td>Ma-cOS</td>
<td>Mac-OS</td>
</tr>
<tr>
<td>Math-SciNet</td>
<td>Math-Sci-Net</td>
</tr>
<tr>
<td>NetBSD</td>
<td>Net-BSD</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>Open-BSD</td>
</tr>
<tr>
<td>OpenOf-fice</td>
<td>Open-Office</td>
</tr>
<tr>
<td>PfaEdit</td>
<td>Pfa-Edit</td>
</tr>
<tr>
<td>Richard</td>
<td>Rich-ard</td>
</tr>
<tr>
<td>Southall</td>
<td>South-all</td>
</tr>
<tr>
<td>Tomaszewski</td>
<td>To-ma-szew-ski</td>
</tr>
<tr>
<td>VM-ware</td>
<td>VM-ware</td>
</tr>
<tr>
<td>Werner</td>
<td>Werner</td>
</tr>
<tr>
<td>WinEdt</td>
<td>Win-Edt</td>
</tr>
</tbody>
</table>

◊ Barbara Beeton

bnb@ams.org
Stacks in \TeX

Pedro Quaresma

Abstract

There are several situations where we need to “forward reference” something that it is not yet available. For example, when we say something like “as we will see in chapter . . . ” and when we make a bibliographic citation. Those situations are well treated in \LaTeX by the use of auxiliary files.

A different situation arises if we want to have a \LaTeX environment where one or more commands depend on the arguments given to other commands; that is, the values of the arguments of one command are taken from the arguments of another. We can also use an auxiliary file as a way of communication between commands but that implies that we have to process the document twice (at least) in order to complete the environment.

In this paper we describe an implementation of stacks in \TeX as a way to solve the problem just described. One command puts the information in the stack, the other command takes the information from the stack, and with this approach we manage to establish communication between commands while processing the document only once.

1 Introduction

In 1990 I was a PhD student in Computer Science and felt the need of a \LaTeX style file for producing diagrams, namely those used in Category Theory [4], e.g.

\[
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
\end{array}
\]

So, I created a style file whose first version used the \LaTeX picture environment as the graphical engine, and in a later version switched to \PCTeX because of its better capabilities. Since the first version the two main goals of the DCpic [5] package were:

- to have a \TeX-only format, in order to have good portability properties;
- to have a simple notation, a notation close to the graph notation where we describe a graph as a set of nodes (objects), and a set of arrows (morphisms) with each arrow having an initial and an end node.

So DCpic implements an environment \begin{dc} \end{dc}, with the command \( \text{\{obj\}(x,y)\{text\}} \) for the nodes, and the command \( \text{\{mor\}(x_1,y_1)(x_2,y_2)\{text\}} \) for the arrows. The diagram pictured above has the following specification:

\begin{verbatim}
\begin{dc}
\obj(1,1){$A$}
\obj(3,1){$B$}
\mor{(1,1)}{(3,1)}{f}
\end{dc}
\end{verbatim}

The syntax is one of the simplest, if not the simplest, among packages of this type [2], but it deviates from the graph notation because the arrows specification is done in absolute terms and not in relative terms, i.e., it does not state the initial and end node of each arrow, but rather their positions. Since version 2 we began to look for a specification syntax that would allow the following specification for the diagram:

\begin{verbatim}
\begin{dc}
\obj(1,1){$A$}
\obj(3,1){$B$}
\mor{A}{B}{f}
\end{dc}
\end{verbatim}

But this implies that the coordinates of objects \( A \) and \( B \) must be passed to the \text{\{mor\}} command. We also want to pass the dimensions of the (box) objects to be able to adjust the arrow length accordingly.

How to do this? I saw two possible solutions:

- the \text{\{obj\}} command writes all the information about its object in an auxiliary file, after which \text{\{mor\}} reads the information from that file.
- the \text{\{obj\}} command writes all the information about its object in an auxiliary structure kept in memory, after which \text{\{mor\}} reads the information from that structure.

The first solution seemed to me too complicated to the problem at hand; it is also inefficient because for a large diagram we have to open and close the auxiliary file too many times (or enforce a strict separation between objects and morphisms). Because of this we chose the second approach and decided to implement stacks.

Using stacks the solution is very simple: \text{\{obj\}} pushes the information onto the stack, and \text{\{mor\}} looks for it on the stack and whenever necessary pops it from the stack.

2 Stacks

To implement stacks we need a structure in which to place the elements and functions to operate on the stack. Using the \textit{Maude} [1, 3] syntax we have:

\begin{verbatim}
fmod STACK is
  sorts Elem NeStack Stack .
  subsorts Elem < NeStack < Stack .
  op empty : -> Stack .
  op push : Elem Stack -> NeStack .
  op pop : NeStack -> Stack .
  op top : NeStack -> Elem .
  op isempty : Stack -> Bool .
  var S : Stack .
endmod
\end{verbatim}

So, stacks in \TeX...
var E : Elem.
eq pop(push(E,S)) = S.
eq top(push(E,S)) = E.
eq isempty(empty) = true.
eq isempty(push(E,S)) = false.
endfm

That is, we have elements and stacks of elements; empty gives us the empty stack; push puts an element on top of the stack; pop deletes the element on top of the stack; top sees (and does not delete) the element on top of the stack; and isempty returns whether the stack is empty or not.

2.1 Stacks in \TeX

Not having a predefined type structure in \TeX that can support stacks, we decided to implement stacks as a macro. We began by defining (initializing) it as the empty stack, that is, a stack that contains only a sentinel, the “end-of-stack” element.

\def\emptystack{:}
\let\stack=\emptystack

The elements of the stacks will be other \TeX elements, e.g., we can put \$x\$ on the stack. The stack is now a structure that may contain almost anything.

The implementation of the the functions is now a matter of redefinition of the macro \stack.

The “push” function has one argument only, the element to be pushed; the result is the stack with that element on top of it.

\def\push#1{%
  \edef\stack{#1.\stack}%
}

(The dot serves as an element separator.)

The “topstack” function has no arguments; the result is the element that it is on the top of the stack. We use an auxiliary function and the “expandafter” command to control the expansion of the arguments.

\def\topaux#1.2.:{#1}
\def\topstack{\expandafter\topaux\stack}

The “pop” function also has no arguments; the result is the stack without the top element. It is very similar to \topstack.

\def\popaux#1.2.:{\def\stack{#2:}}
\def\pop{\expandafter\popaux\stack}

To implement the “isempty” predicate we need to define the appropriate “if”. In \DCpic we have opted for two “ifs”, one to test if the stack is empty, and the other to test if the stack is not empty.

\newif\ifisempty
\newif\ifnisempty

2.2 Using Stacks

The Polish Notation To illustrate the use of these stacks in \TeX, let’s pretend to calculate arithmetic expressions in Reverse Polish Notation (postfix notation). Our example will be this expression:

\$2\ +\ 3\ \times\$

Start by pushing all the elements into the stack:

\push{2}
\push{1}
\push{+}
\push{3}
\push{$\times$}

After this, the stack has this form:

Stack = \$\times.3.+.1.2.$

Loop until the stack is empty We can construct a cycle that will stop when the stack is empty:

\loop
\pop
\pop\quad Stack=stack
\nisempty\stack
\ifnisempty
\\endgraf
\repeat

Using the stack from the previous example:

\begin{verbatim}
Pop, Stack=\times.3.+.1.2.:Pop, Stack=3.+.1.2.:Pop,
Stack=+.1.2.:Pop, Stack=1.2.:Pop, Stack=2.:Pop, Stack=
\end{verbatim}

Stacks in the \DCpic package The use of stacks allows a very simple notation for the specification of our diagrams in \LaTeX. For example:

\begin{verbatim}
\def\isempty1{\%
  \let\arg=#1\relax
  \if:\arg\ \isemptytrue
  \else \isemptyfalse\fi}
\def\nisempty1{\%
  \let\arg=#1\relax
  \if:\arg\ \nisemptyfalse
  \else \nisemptytrue\fi}
\end{verbatim}
As you can see, the communication between \obj and \mor does not follow a strict first-in-last-out discipline; so what we do is to preserve the stack before a pop operation and recover the value afterwards. A list structure would be more appropriate for DCpic, but the simplicity of the stack implementation justifies, in our view, a little loss of efficiency.

3 Conclusions
Using an auxiliary structure like the stack gives us the possibility of organizing our \TeX\ programs with intercommunicating macros. The communication is established using the global variable \stack.

We decided not to deal with the error situations (e.g., a pop of the empty stack) in our implementation of stacks. The \mor command analyzes the stack to see whether it is empty; if so, then it writes an error message in the output and tries a naive error recovery using some default values.

The use of stacks in DCpic allows us to have a very simple notation for the graphs without a “visible” burden to the user. We are certain that this approach will be useful in other situations.

References
Kissing circles: A French romance
in METAPOST

Denis Roegel

Abstract
When circles meet, they kiss. If three of them kiss, others can try to join and kiss all of them at once. In this article, we look at this problem from the METAPOST point of view, and we try to tell circles how to kiss, no matter their position and size. Recursive kissing will also be attempted.

1 Introduction
Apollonius of Perga (3rd century BC) was a Greek geometer, author of, among other things, a treatise on conical sections. He is credited of having coined the terms ellipse, parabola and hyperbola. His book Tangencies, cited by Pappus, defines the tangents problem as the problem of finding a circle tangent to three other objects being any combination of points, lines or circles. Apollonius showed how to solve this problem with a compass and straightedge, and it is now known as Apollonius’ problem. When the three objects are circles, there are in general eight different solutions (Gisch and Ribando, 2004).

However, when the three circles are externally tangent to each other, these solutions reduce to only two non-trivial ones, namely the external and internal tangent circles, known as outer and inner Soddy circles (figure 1).

Descartes found a simple analytic solution. The curvatures $e_1 = 1/r_1$, $e_2 = 1/r_2$, $e_3 = 1/r_3$ of three circles kissing each other are related to the curvature $e_4$ of a Soddy circle through the equation

$$2(e_1^2 + e_2^2 + e_3^2 + e_4^2) = (e_1 + e_2 + e_3 + e_4)^2$$  (1)

In this equation, $e_1$, $e_2$, and $e_3$ being given, $e_4$ has two solutions. The positive solution corresponds to the inner Soddy circle and the negative solution to the outer Soddy circle (whose radius is then $-1/e_4$). The analytic solution can be used to iterate the construction, but one has to be careful to avoid overflows. The outer Soddy circle will always appear at the border, hence a small curvature value, whereas smaller and smaller circles will be packed on the border, hence larger and larger values for the other curvatures. The METAPOST language is not very well suited to handling very small or very large values, and a geometric construction with no calculations is better suited to this problem.

2 David Eppstein’s construction
In 2001, David Eppstein published a new construction for the inner and outer Soddy circles (Eppstein, 2001). Our purpose will not be to prove that this construction is indeed correct, but to see how best it can be implemented, and in particular in the most general way.

Eppstein’s construction goes as follows. Given three tangent circles $C_1$, $C_2$ and $C_3$ (figure 2), a triangle connecting their centers is drawn. From each center, a perpendicular line is dropped to the opposite side of the triangle. The intersections between the perpendiculars and the triangle sides are marked with discs having small holes. The perpendiculars intersect their originating circle at two points, marked with discs and circles. Now, each disc can be connected to the tangency point (vertical cross) of the two other circles. This line meets the first circle at another point than the one with a disc, and we mark it with a filled square. The three points with filled squares are the tangency points of the inner Soddy circle.

The same procedure applied to the circle points yields the square points which are the points of tangency of the outer Soddy circle.

This construction can be used to find the circles internally tangent to the outer Soddy circle and circles $C_1$ and $C_3$ for instance, and the procedure can be used to build the Apollonian gasket (figure 3).
3 Construction problems and METAPOST preliminaries

Finding the Soddy circles is rather straightforward, although special cases arise already at this stage. But the real problems are met when the construction is iterated, as the configurations of the circles change and there are several cases. The main source of difficulty is the duplicity of the circles. Eppstein’s construction gives six points, but we must take great care in grouping these six points into two sets, and we have to find out which set corresponds to the circle we want to draw.

We will start by building a number of robust macros for common tasks. Some of these macros can easily be reused in other applications.

3.1 Height of a triangle

Our first macro (figure 4) finds the height of a triangle $ABC$ starting at $A$. The height may not actually intersect the opposite side of the triangle, hence it is a good idea to use the whatever construction. Our macro states that the intersection $H$ is somewhere on $[B, C]$ and somewhere on $[A, D]$ where $D$ is a point constructed using $\overrightarrow{BC}$. The macro doesn’t return the intersection $H$, but a path going beyond $A$ and $H$ by at least $r$, which is a value provided to the macro.

The idea is that we will use these paths to find their intersection with the originating circles, hence they need to be extended by at least the radius of the circle, and actually a bit more, in order to be sure that there is an intersection. Two paths, having an intersection coinciding with the start of one of the paths, may actually have no intersection for METAPOST due to rounding errors.

3.2 Circles

Circles can be obtained with the fullcircle macro, and this macro will be used to find intersections. However, circles are often a problem in that their intersection with a line is usually not unique, and if only one intersection is wanted, it is necessary to ensure that we get the right one. Another related problem is the discontinuity within a circle. Although not visible, a circle has a beginning and an end for METAPOST. It is often a good idea to avoid the discontinuity, which is a source of problems.

One convenient way of influencing the intersection returned by using the intersectionpoint function with a circle is to rotate the circle around its center (figure 5). This may seem of no use, but actually the intersections are computed in such a...
way that the parameters of the paths are minimized. So, if we can make sure that the circle is in a position such that the intersection with the minimal parameter is the one we want, we will be rewarded.

We therefore define a macro for a circle of radius \( r \), centered at \( c \) and rotated by an angle \( a \).

### 3.3 Tangencies

We define a macro for the tangency point between two circles which are known to be tangent (figure 6). This macro needs to take care of the case where one circle is inside the other. This is the case when the distance between the two centers is smaller than one of the radii. Then, the circle with the smallest radius lies within the other. The tangency is obtained by extending the line connecting the circle centers. For instance, if \( C_a \) lies inside \( C_b \), the tangency point is

\[
C_a + r_a \times \frac{C_a C_b}{\|C_a C_b\|}.
\]

### 3.4 Angle between two lines

The angle between two lines is obtained using \texttt{angle} and is brought within the interval \([0, 180^\circ]\) (figure 7).

### 3.5 Circle-line intersection

Eppstein’s construction makes it necessary to find an intersection between a line and a circle, other than a given point. Devising a macro for that purpose which works in all cases is not trivial. One case that has to be taken into account is the case where the line is tangent to the circle, and possibly has no intersection at all with the circle due to rounding errors.

When writing such a macro (figure 8), we also have to ensure that it will not fail when the two intersections are too close. Given a point \( A \) on the circle, and another point \( B \), our solution is to first compute the angle between the radius at \( A \) and the vector \( \overrightarrow{AB} \). If this angle is between 89 and 91 degrees, we assume that the line \( (AB) \) is tangent to the circle. Even if it is not exactly tangent, the two intersections will be very close and can be confounded. In that case, we return \( A \).

If not, we find the second intersection by comparing the distance to the center of the circle of two points chosen in opposite directions from \( A \) on the line. If \( B' \) is such that \( \overrightarrow{AB} + \overrightarrow{AB'} = \overrightarrow{0} \), then the second intersection is on \( [A, E] \), where \( E \) is the closest of \( B \) and \( B' \) from the center of the circle. We slightly rotate the circle counterclockwise in order to avoid \( A \) being found by the intersection. In this way, \( A \) corresponds to a very large parameter and will be avoided in the computation.

### 3.6 Circle going through three points

The final operation in Eppstein’s construction takes three points and finds the circle going through these points (figure 9). We first obtain the center of this circle as the intersection of two medians. The macro \texttt{whatevermedian} returns an undefined point on a
def tangency(expr Ca, ra, Cb, rb) =
    (if (arclength(Ca--Cb) < rb) % a inside b
     or (arclength(Ca--Cb) < ra): % or b inside a
        if ra < rb: % a inside b
            Ca + ra * unitvector(Ca-Cb)
        else: % b inside a
            Cb + rb * unitvector(Cb-Ca)
        fi
    else: circle(Ca, ra, 0) intersectionpoint (Ca--Cb)
    fi)
enddef;

Figure 6: The three cases of tangencies. Notice that this macro will not fail if the circles have no intersection due to rounding errors.

vardef angleof(expr Va, Vb) =
    save a;
    hide(
        a = angle(Vb) - angle(Va);
        forever:
            if a >= 180: a := a - 180; fi;
            if a < 0: a := a + 180; fi;
            exitif ((a < 180) and (a >= 0));
        endfor;
    )
    a % angle
enddef;

Figure 7: Angle between two lines defined by vectors \( \overrightarrow{VA} \) and \( \overrightarrow{VB} \), brought on the interval \([0, 180^\circ]\).

line and is used in the circle_through macro. This macro defines the center and the radius of the circle.

3.7 Existence of a segment intersection
It will be useful to have a macro telling when two segments (and not lines) intersect, and when they do not. An easy way to achieve this is to provide a boolean version of the intersectionpoint macro and have it return true instead of the intersection and false instead of an error message, as follows:

secondarydef p intersectionpoint_b q =
    begingroup save x_, y_
        (x_, y_) = p intersectiontimes q;
        if x_ < 0: false
        else: true
        fi
    endgroup
enddef;

3.8 Innerness and outerness
Next, we define the following macro which will take four circle centers. Circles \( B \) and \( C \) are tangent externally and circle \( D \) is internally tangent to both. \( A \) is a circle externally tangent to \( B \) and \( C \), but internally tangent to \( D \). In other words, \( D \) is the outer Soddy circle of \( A, B \) and \( C \).
vardef intersection_circle_line(expr c,r,A,B)=
    save a,BP,I,uv;
    hide(
        pair BP,I,uv;
        a=abs(angleof(B-A,A-c)-90);
        if a<1: I=A;
        else:
            BP=A+(A-B);
            if arclength(c--B)<arclength(c--BP):
                uv=unitvector(A-B);
                I=circle(c,r,angle(A-c)+1) intersectionpoint ((B-2.1r*uv)--(1.1[B,A]));
            else:
                uv=unitvector(A-BP);
                I=circle(c,r,angle(A-c)+1) intersectionpoint ((BP-2.1r*uv)--(1.1[BP,A]));
            fi;
        fi;
    I % point returned
enddef;

Figure 8: Intersection between a line and a circle. The circle, its point $A$ and $B$ are given. We search the other intersection $I$. The shortest of $cB$ and $cB'$ indicates on which side of the line $(BB')$ is $I$, with respect to $A$.

Conversely, given $B$, $C$ and $D$, Eppstein’s construction gives two circles, one of them being $A$. It turns out that which points lead to which circle depends on the existence of an intersection between $[Ca,Cd]$ and $[Cb,Cc]$, where $Ca$, $Cb$, $Cc$, and $Cd$ are the circle centers. We call the two possible circles tangent externally to $B$ and $C$, but internally to $D$, inner when the aforementioned intersection exists and outer when it doesn’t. This is consistent with the outer Soddy circle center being such that there is no intersection for the previous segments.

In practice, of the two circles, the inner one will be the smaller of the two.

def is_inner(expr Ca,Cb,Cc,Cd)=
    ((Cb--Cc) intersectionpoint_b (Ca--Cd))
enddef;

3.9 Slope
Given a segment, the slope macro gives its slope as an angle. This will be a convenient macro when we need to rotate a circle in order to favor a certain intersection.

def slope(expr p)=
    angle((point 1 of p)-(point 0 of p))
enddef;

4 The main macro
The main macro takes four circle centers, as well as four radii, and an integer for the recursion depth. Initially, the first three circles are three tangent circles for which we find the Soddy circles. The fourth circle will be non-meaningful and assigned a negative radius.
Figure 9: Circle going through three points $A$, $B$ and $C$.

Later, when iterating the construction, there will be two cases. In the first case the three first circles are tangent externally, and an inner Soddy circle is to be found. In that case, the fourth circle is also non-meaningful.

The second case is a border case, where the first circle is the outer Soddy circle. The second and third circles are externally tangent, but internally tangent to the Soddy circle. And then the fourth circle is another circle externally tangent to the second and third, and internally tangent to the first. This fourth circle has already been drawn, but it is necessary to find out which one is the other circle not yet drawn at the border.

4.1 Computing the tangencies and triangle heights

The tangencies and triangle heights are obtained straightforwardly, given the previous definitions.

```plaintext
vardef tangent_circles(expr Ca,Cb,Cc,Co, ra,rb,rc,ro,n)=
  save C,T,r,ht,Ihc,Ilc;
  pair C[]; % centers
  pair T[]; % tangencies
  numeric r[]; % radii
  path ht[]; % heights

  C1=Ca;C2=Cb;C3=Cc;
  r1=ra;r2=rb;r3=rc;
  T[1][2]=tangency(C1,r1,C2,r2);
  T[2][3]=tangency(C2,r2,C3,r3);
  T[3][1]=tangency(C3,r3,C1,r1);
  ht1=triangle_height(C1,C2,C3,r1);
  ht2=triangle_height(C2,C1,C3,r2);
  ht3=triangle_height(C3,C1,C2,r3);

  In order to simplify certain expressions, we also define
  def next(expr i)=
    (if i+1<4:i+1 else: 1 fi)
  enddef;

4.2 Diameters and other intersections

The intersections between the heights and the circles are easy to obtain, but the key to success is to group them correctly. In our case, we first group the intersections which go towards the feet of the heights, by slightly rotating the circles clockwise. This gives the points $Ihc[i][1]$, which are marked with small circles.

Then, the circles and discs and square and filled squares.

```
if firststep: % case 1
  firststep:=false;
  % outer Soddy
  circle_through(Ilc1,Ilc2,Ilc3)
  (C4)(r4);
  % inner Soddy
  circle_through(Ilc4,Ilc5,Ilc6)
  (C5)(r5);
  % we recurse
  if n>0: % border cases
    tangent_circles(C4,C1,C2,C3,
                   r4,r1,r2,r3,n-1);
    tangent_circles(C4,C2,C3,C1,
                   r4,r2,r3,r1,n-1);
    tangent_circles(C4,C3,C1,C2,
                   r4,r3,r1,r2,n-1);
  fi;
else:
  if ro<0: % case 2
    circle_through(Ilc4,Ilc5,Ilc6)
    (C5)(r5)
  else: % case 3
    if is_inner(Co,C2,C3,C1):
      circle_through(Ilc1,Ilc2,Ilc3)
      (C4)(r4);
    else:
      circle_through(Ilc4,Ilc5,Ilc6)
      (C4)(r4);
  fi;
fi;

% we recurse
if n>0: % case 4
  if ro>0: % case 5
    tangent_circles(C1,C2,C4,C3,
                   r1,r2,r4,r3,n-1);
    tangent_circles(C1,C3,C4,C2,
                   r1,r3,r4,r2,n-1);
    tangent_circles(C2,C3,C4,origin,
                   r2,r3,r4,-1,n-1);
  else: % case 6
    tangent_circles(C1,C2,C5,origin,
                   r1,r2,r5,-1,n-1);
    tangent_circles(C1,C3,C5,origin,
                   r1,r3,r5,-1,n-1);
    tangent_circles(C2,C3,C5,origin,
                   r2,r3,r5,-1,n-1);
  fi;
fi;

Figure 10: The high-level structure of the recursion.

pair C[]; % centers
numeric r[];
numeric n; % depth
n=7;
C1=origin;
r1=4cm;
r2=3cm;
r3=6cm;
% we find the center C2:
C2-C1=(r1+r2,0);
% there are now two possibilities for C3,
% we keep only one
C3=site(C1,r1+r3,0)
  intersectionpoint circle(C2,r2+r3,0);
draw circle(C1,r1,0);
draw circle(C2,r2,0);
draw circle(C3,r3,0);
firststep:=true;
tangent_circles(C1,C2,C3,origin,
               r1,r2,r3,-1,n);

Figure 11: The driver of the construction. Three initial circles are defined and the general macro is called.

Once the outer Soddy circle $C_4$ has been obtained, the macro is again called on each border. In each case, we provide the outer Soddy circle as the first parameter, then two of the three inner circles, then the third inner circle. During the next call of the macro, the “case 3” part will be in effect, and the new circle to be squeezed between the two inner circles and the outer Soddy circle is found out using the innerness criterion mentioned above.

“Case 2” applies when the inner Soddy circle of three externally tangent circles has to be found.

But no matter what, when “case 4” is reached, we have found a circle and two new cases apply: either we have a border case with the original outer Soddy circle $C_1$ (case 5), or the circle is the inner Soddy circle of three externally tangent circles (case 6). Splitting case 5 leads to two new border cases and one inner case (the usual inner Soddy circle). Splitting case 6 leads to three new inner cases (the usual inner Soddy circles). Note that the parameter origin is irrelevant when the radius is negative.

The macro is started as shown in figure 11.

5 Conclusion

Our little journey through kissing circles has presented every detail of the METAPOST construction. We have followed Eppstein’s construction closely.
Figure 12: An Apollonian gasket of depth 7, with 6563 circles. The total number of circles is \(5 + 3 \times (3^n - 1)\) where \(n\) is the depth. For \(n = 0\), we have five circles, which are the three base circles and the two Soddy circles.

The resulting code is simple, but this simplicity was not achieved right away. It is supported by the robustness of each macro which tries to be as general as possible. The code produced is not tied to a particular set of circle radii and should work in all cases, provided METAPOST's capacities are not overflowed. We conclude with an Apollonian gasket of depth 7, built with our code (figure 12).

References


Using the RPM package manager for (L)\TeX\ packages

Tristan Miller

Abstract

RPM is a package management system which provides a uniform, automated way for users to install, upgrade, and uninstall programs. Because RPM is the default software distribution format for many operating systems (particularly GNU/Linux), users may find it useful to manage their library of \TeX\-related packages using RPM. This article explains how to produce RPM files for \TeX\ software, either for personal use or for public distribution. We also explain how a (L)\TeX\ user can find, install, and remove \TeX\-related RPM packages.

1 Background

1.1 The evolution of package management systems

In the first decade or two of personal computing, most software was distributed on and run directly from floppy disks. Users lucky enough to have a hard drive could copy the contents of these floppies into a directory on their hard disk and run it from there. When a user wanted to delete the program, he had to remember which directory he had copied it to, find it in his file system, and manually delete it, taking care to first preserve any data files he had created.

As programs and hard disk capacities grew in size, software was increasingly distributed on multiple floppy disks or (later) on a CD-ROM. Vendors would provide tools — usually simple shell scripts — to automate the process of creating a directory on the hard drive, copying the contents of each floppy to it, configuring the installed copy of the program, and then finally deleting it when the user requested that it be uninstalled.

These tools grew in sophistication along with the underlying operating systems (OSes), which by the 1990s had begun to provide standard hardware drivers and graphical interface toolkits for third-party software to use. Now software installation programs could not merely copy themselves to the hard drive; they also had to search for the presence and location of requisite system and third-party software, register themselves with the OS so that they too could be found by other programs, and create icons for the user in the system menu or desktop. Some of the better installers would also do sanity checks such as making sure the user didn’t install two copies...
of the same software package, or automatically detecting and upgrading older installed versions of the software. With the advent of dial-up bulletin board systems and eventually home Internet access, it became important for software to be downloadable in a single file rather than as dozens or hundreds of individual files as was the case with physical media.

Because each vendor wrote its own installation program, users often found themselves confused by different interfaces and lacking a single tool with which to install, upgrade, and remove software. To remedy this, each operating system developed its own standard software package management tool to be used by all users and vendors. Software developers can now distribute their programs in specially prepared packages containing the source code and/or binary executables for the software along with important metadata such as the software’s name, version number, vendor, and dependencies on other software. Packages might also include checksums or cryptographic signatures which the package management system can use to verify that they were not corrupted or tampered with during distribution.

Users install and remove these packages using standard system software, which keeps a database of all installed packages and makes sure that all dependencies are met — for example, by automatically fetching prerequisite packages, or warning the user if he is about to remove a package that other installed software depends on.

1.2 Package management and \TeX

Unfortunately for fans of quality typesetting, \TeX and friends are currently stuck in the Dark Ages of software package management. Though \TeX distributions now mostly conform to the \TeX Directory Structure [8], which specifies standard locations for the installation of certain types of files, there is currently no standard package format or associated tool for installing, upgrading, and removing macro packages, styles, classes, scripts, fonts, documentation, and other \TeX-related paraphernalia available on CTAN and elsewhere.

As a result, users who download new packages must themselves check for prerequisites, manually process `.ins` and `.dtx` files, create the appropriate directories in their `texmf` tree, copy the files in, and perform necessary post-installation configuration (such as running `texhash`). Worse yet, when a user wishes to uninstall a package, he must manually remove the files, often from multiple directories. This usually entails consulting the original package installation instructions to help remember what got installed where.

Fortunately, until such time as the \TeX community develops and settles on its own package management standard, users can avail themselves of their operating system’s native package management system for the maintenance of \TeX packages.\footnote{In this document “\TeX” refers to the entire \TeX system, including \LaTeX, \METAFONT, \BIBLATEX, and other components. Similarly, a “\TeX package” here means any set of related files distributed, installed, and maintained as a unit. This meaning includes but is not limited to \LaTeX\ε packages, which are style files supplementing a document class.} In this article, we describe how to do this using the RPM Package Manager, a packaging system originally developed by Red Hat and now in widespread use on several operating systems.

1.3 RPM versus other package managers

RPM has a number of good features to recommend itself to \TeX package management. Most important is its portability — RPM enjoys the status of being the official package format specified by the Linux Standard Base [4], meaning that any LSB-compliant GNU/Linux distribution can handle RPM packages. Distributions which use RPM by default include Aurox, Fedora Core, Lycoris, Mandriva (formerly Mandrake), PCLinuxOS, PLD, Red Flag, Red Hat, and SUSE. Distributions which use a different native package format but which can handle RPM by virtue of their LSB compliance include Debian, MEPIS, Slackware, and Ubuntu.

However, RPM is by no means limited to GNU/Linux operating systems. The RPM tools have been ported to Mac OS X, Novell Netware, and some commercial Unixes. Because the tools and file format specifications are released under a free license, it is possible to reimplement them on virtually any operating system. In fact, some work has already gone into porting RPM to Microsoft Windows, with some rudimentary tools available now.

It bears mention, however, that there are many alternatives to RPM the user may wish to consider. At least one \TeX distribution, MiK\TeX, for MS-Windows, provides its own packaging utility, `rpm` [6, §§3.2 and A.9], which has much the same functionality as RPM. However, `rpm` works only with the `MiK\TeX` distribution, and moreover is a network tool which fetches packages from some central repository. As far as the present author is able to determine, it is not possible for package authors to create and distribute their own `MiK\TeX` packages.

Another alternative is to use your operating system’s native package management system. In many cases, this requires purchasing the system’s official software development kit (SDK).
of Microsoft Windows, this SDK can be downloaded for free, though it requires a gigabyte of hard drive space and comes with a restrictive license.) Another disadvantage is that the distributed packages are often bundled as executable files, adding considerable overhead (possibly several megabytes) to the size of the package. (Consider that most \TeX{} packages are only a few kilobytes in size.) Users may also be wary of running executables for fear of viruses or spyware which the packager may have deliberately or unwittingly included.

Users of Mac OS X will be pleased to note that there exists an unofficial package management system, i-Installer,\footnote{http://ii2.sourceforge.net/}, which enjoys notable popularity among \TeX{} users on the Mac. The author of this tool provides i-Installer packages, or i-Packages, for a number of \TeX{} packages. Furthermore, the i-Installer distribution includes tools for users to create their own i-Packages.

1.4 About this article

This article makes liberal use of illustrative examples to help the reader understand how to use the RPM packaging tools. To help distinguish between various types of computer input and output, we employ the following typographical conventions:

• When depicting an interactive shell session, any text set in a \texttt{teletype font} marks that output by the computer, and \textbf{bold teletype} indicates text input by the user. At the beginning of a line, the \# character indicates the superuser (root) shell prompt, while the \$ character indicates the shell prompt for a normal user account.

• Other instances of computer input and output are rendered in a \texttt{teletype font}. Placeholders for arguments the user is expected to specify as appropriate are rendered \texttt{⟨like_this⟩}.

• The actual contents of configuration files created by the user are set in \texttt{small teletype text} surrounded by a box.

In §2, we give a brief overview of the RPM command-line interface and how it’s used to install, upgrade, and remove packages. It is aimed at novice users who simply want to know how to install or remove a \TeX{} RPM package they found on the Internet. Readers already familiar with installing RPM packages may wish to skip this section.

Section 3 describes how you can create RPM packages for existing \TeX{} packages; this information is likely to be of greatest interest to package developers and distributors, but also to advanced \TeX{} users who want to avoid the hassle of manual package management. This section assumes that you have a basic familiarity with downloading and manually installing \TeX{} packages.

Finally, §4 briefly touches on some advanced topics for RPM packagers and distributors.

2 RPM basics

2.1 RPM files and where to find them

Software for use with RPM is distributed in files known as RPM packages, which have the filename suffix \texttt{.rpm}. In order to distinguish between different versions of a package, a standard file naming scheme is employed which encodes the package name, its version and release number, and the computer architecture it is designed to work with. The syntax is illustrated with an example in Figure 1.

• The \texttt{package name} indicates the name of the software (or in our case, \TeX{} package) packaged in the RPM.

• The \texttt{version} is the version of the software to be installed.

• The \texttt{release} field is used to indicate revisions of the packaging of that particular version of the software. For instance, sometimes the person packaging the software will make an error, such as leaving out a particular file. Every time the software is repackaged to fix such an error, its release number is increased.

• The \texttt{architecture} field indicates the type of computer processor the software is designed to work with. For most executable programs, this field will have a value such as \texttt{i586} (Intel 586), \texttt{ppc} (PowerPC), or \texttt{sparc} (Sun SPARC). Most \TeX{} packages, however, do not depend on any particular computer processor and therefore have the value \texttt{noarch} in this field.

One important property of an RPM package which is not typically included in its filename is the operating system it is designed to work with. Different Unix and GNU/Linux distributions, such as SUSE and Fedora Core, may have slightly different conventions regarding how and where programs and documentation are installed. Therefore it is always important to install only those RPM packages which are meant for the OS distribution you are using.

\begin{verbatim}
Figure 1: A sample RPM filename
gnomovision-1.2-273.i586.rpm
package name version release architecture
\end{verbatim}
FTP (Usually whatever web page or FTP site you find the RPM on will indicate which distribution it is for.) This caveat is compounded by the fact that different TeX distributions may be available for the same operating system, and that these distributions may also have different conventions for how and where to install files, and may come with different default packages.

Therefore, when looking for TeX RPMs, you must ensure not only that they are specific to your operating system, but also to your TeX distribution. To help alleviate this problem, we recommend that packagers and distributors of RPMs prepend the name of the TeX distribution to the RPM package name. Thus, for example, an RPM package for the E\TeX{} package \texttt{breakurl} for use with the \texttt{tetex} distribution would have a filename such as \texttt{tetex-breakurl-0.04-1.noarch.rpm}

However, whether this RPM is meant for SUSE, Red Hat, or some other GNU/Linux distribution must be indicated separately.

Most developers writing programs for Unix-like systems will provide RPM packages of their software on their official website. Alternatively, there exist several Internet search engines, such as \texttt{rpmfind.net}, which index RPM files. Currently TeX packages are not widely available as RPM packages, though hopefully this article will go some way towards encouraging TeX package developers and distributors to remedy the situation. In the meantime, a number of E\TeX{} RPMs for the SUSE distribution of \texttt{te\TeX} have been made available on the present author’s home page at \url{http://www.nothingisreal.com/tetex}.

### 2.2 Installing, upgrading, and removing packages

The main utility for manipulating RPM packages is named, reasonably enough, \texttt{rpm}, and on Unix-like systems is usually located in the \texttt{/bin} directory.\footnote{This article will hereinafter assume that the user is working on a GNU/Linux or Unix system; however, most of what is presented is easily applicable to other operating systems which support RPM.} \texttt{rpm} is a command-line utility, and we describe its use in this section. Though it is not difficult to use, most operating systems provide a graphical interface to it, so installing or upgrading a package is often as simple as clicking on the RPM file in your file explorer.

To install an RPM package you have obtained, you issue the following command (substituting the actual filename) while logged in as the superuser (root):\footnote{To fit the formatting of this journal, we sometimes break lines in shell command examples by using a backslash (\) followed by a newline. In practice you can type the commands on a single line, omitting the backslash and newline.}

\begin{verbatim}
# rpm --install \n gnomovision-1.2-273.i586.rpm
\end{verbatim}

(For more verbose output and a progress meter, the \texttt{--verbose} and \texttt{--hash} options can also be specified.) On the other hand, if you haven’t yet downloaded the file but know its location on the Internet, you can tell \texttt{rpm} to fetch it for you via HTTP or FTP:

\begin{verbatim}
# rpm --install ftp://ftp.foo.de/\n gnomovision-1.2-273.i586.rpm
\end{verbatim}

\texttt{rpm} will then process the named file, make sure that all its dependencies are met and that no conflicts are caused, and install it. Once a TeX RPM is installed, no further work or setup should be needed; whatever TeX package it installed should be immediately available to your TeX installation. There should be no need to manually update TeX’s filename database (e.g., \texttt{texhash}).

If a certain RPM package is already installed on your system and you have downloaded a newer version, you can upgrade the existing installation as follows:

\begin{verbatim}
# rpm --upgrade \n gnomovision-1.2-273.i586.rpm
\end{verbatim}

The \texttt{--erase} option uninstalls an RPM package you have previously installed. Note that you do not specify the complete package filename; just the name of the software is used:

\begin{verbatim}
# rpm --erase gnomovision
\end{verbatim}

Conveniently, \texttt{rpm} will issue a warning if you try to remove a package which other installed packages require. You can then decide to remove those packages as well or abort the process.

### 2.3 Getting information on packages

\texttt{rpm} also provides the \texttt{--query} option for listing and getting information on installed packages. Used by itself and a package name, this option simply prints out the version and revision number of the package if it is installed. Alternatively, \texttt{--query} can be used with auxiliary options to perform various useful tasks. For example,

\begin{verbatim}
# rpm --query --info gnomovision
\end{verbatim}

\texttt{rpm} displays the details of the \texttt{gnomovision} package, including its size, packaging date, installation date, as well as its purpose and functionality. Adding the \texttt{--list} option will also show a list of each file the
This package provides a command much like hyperref's \url that typesets a URL using a typewriter-like font. However, if the dvips driver is being used, the original \url doesn't allow line breaks in the middle of the created link: the link comes in one atomic piece. This package allows such line breaks in the generated links.

Note that this package is intended only for those using the dvips driver. Users of the pdflatex driver already have this feature.

Distribution: SuSE 9.0 (i586)
/usr/local/share/texmf/doc/latex/breakurl/README
/usr/local/share/texmf/doc/latex/breakurl/breakurl.dvi
/usr/local/share/texmf/doc/latex/breakurl/breakurl.pdf
/usr/local/share/texmf/tex/latex/breakurl/breakurl.sty

Figure 2: Output of rpm --query --info --list --package tetex-breakurl-0.04-1.rpm

package will install. (See Figure 2 for sample output of a more realistic package — our example in the next section, in fact.) Note that by default, the --query option searches only the database of installed packages. To use it on a RPM file you have downloaded, you must use it in conjunction with the --package option and the filename.\footnote{Some pagers and file viewers, such as less, understand the RPM file format; using them to view RPM files will result in output similar to that of rpm --query --info --list --package.} For example:

```
# rpm --query --info --package \gnomovision-1.2-273.i586.rpm

Another useful command with --query is

```
# rpm --query --all
```

which lists all RPM packages installed on the system.

For most ordinary users, the above commands are all that is required to effectively use rpm. For advanced operations, consult the rpm man page, or use whatever graphical interface your system provides.

3 Creating RPM packages

So, you're a developer who has created a new \TeX package, or perhaps you're just an ordinary user who has downloaded something from CTAN and wants to package it as an RPM. Before you can begin creating RPM packages, though, you first need to set up a few things; these need be done only once.

3.1 First-time setup

The program used to create RPM packages is named rpmbuild. Before you can use it, however, you need to create a workspace for its use. You can do this with the following shell command:

```
$ mkdir -p ~/rpm/{BUILD,SOURCES,\SPECs,SRPMS,RPMS/noarch}

It's OK to specify a directory other than ~/rpm if you wish.

Next, you need to create in your home directory a configuration file named .rpmmacros which provides some default information to be used when building packages; Listing 1 shows a sample. The %packager line should specify your name and e-mail address, formatted as shown, so that people can contact you to report bugs or problems with your package. The %topdir line should correspond to the workspace directory you created previously. (If you are unsure of the full path to your home directory, the pwd shell command can tell you what it is.)

```
%packager Tristan Miller <Tristan.Miller@dfki.de>
%topdir /home/miller.rpm

Listing 1: A sample ~/.rpmmacros file
3.2 Preparing the TeX package source

With the above one-time setup steps complete, you are now ready to begin building RPM packages. The first thing to do is to fetch the source to the TeX package for which you want to create an RPM. If you are a package developer, we assume you already have all the files; for those of you creating RPMs for others’ TeX packages, you will have to download the files from the author’s web page or from CTAN. Normally these will be available as a tar.gz or zip archive.

Let’s assume that we are installing the package breakurl, which is available at http://ctan.org/tex-archive/macros/latex/contrib/breakurl. Follow the “get this entire directory” link, specify a mirror that supports directory archives, and download the package into a temporary directory on your machine, such as /tmp. Then decompress the archive using unzip or tar as appropriate:

```bash
$ cd /tmp
$ tar xzvf breakurl.tar.gz
breakurl/
breakurl/README
breakurl/breakurl.dtx
breakurl/breakurl.ins
breakurl/breakurl.pdf
```

3.3 Writing the spec file

Next you must prepare a spec file, which is a set of commands instructing rpmbuild how to compile the source files and where to install them. spec files are generally stored in the SPECS subdirectory of the workspace you created in §3.1, and are composed of a number of sections, or stanzas:

- the Header stanza, which defines custom macros and gives basic information about the package;
- the Prep stanza, which unpacks the package and prepares it for compilation;
- the Build stanza, which provides instructions for compiling the package;
- the Install stanza, which provides instructions for installing the package;
- the Files stanza, which lists all the files to be included in the package distribution;
- the Scripts stanza, which specify programs to be run before and after installation or uninstallation of the package; and
- the Changelog stanza, which contains a record of changes made to the RPM package.

In the following subsections we continue with our breakurl example by building its spec file, named `/rpm/SPECS/breakurl.spec`. We show the various stanzas as they are being built; the completed spec file is presented at the end in Listing 9.

3.3.1 The Header stanza

The Header stanza appears, unlabelled, at the beginning of the spec file and typically contains two kinds of information: macro definitions, and fields containing important metadata about the package.

Macros. A number of macros are predefined by your RPM distribution. For example, the macro `%tmppath` is predefined to some temporary directory in your file system, such as /tmp or /var/tmp. Other macros are automatically defined by rpmbuild as it processes the spec file, using information from the fields you specify. For example, rpmbuild assigns to the `%name` and `%version` macros the same values you specify for the Name and Version fields (see below) so that you can use these values later on in your spec file.

In addition to these predefined macros, you can create and use your own custom macros. A macro definition looks like this:

```
%define (macro_name) (macro_value)
```

Once a macro has been defined, you can reference it later with the following syntax:

```
%(macro_name)
```

It is permissible to use a macro in the definition of a new macro.\(^6\)

One useful macro we should define here is the root of our local TDS tree — that is, where new TeX packages should be installed on the system [8, §2.3].\(^7\) The exact location of this directory varies with both your OS and TeX distribution, so you will need to consult the appropriate documentation. The TeX distribution on SUSE 9.0, for example, uses `/usr/local/share/texmf`, so in that case we would define a macro as follows:

\(^6\) Observant readers will note that what we entered into our `/rpm/macros` file in §3.1 were actually macro definitions.

\(^7\) Why install to the local tree rather than the main texmf tree? Consider the case where the TeX distribution includes version 1.0 of a certain package foo. Say we then produce an RPM package of version 1.1 of foo which installs to the main texmf tree rather than the local tree. If the user installs this RPM and then later decides that version 1.1 is buggy and removes it, he will be unable to revert to version 1.0 without reinstalling his TeX distribution. Furthermore, if he does reinstall his TeX distribution, any other RPM packages that happened to install themselves in the root texmf tree will likely be overwritten. Installing new and upgraded versions of packages in the local tree avoids this problem; new TeX packages can be installed and removed while preserving older versions in the root tree. (When two versions of a package exist, most TeX distributions are configured to prefer the local-tree version over the root-tree version.)
%define texmf /usr/local/share/texmf

Fields. Fields are defined with the following syntax:

\( \langle field\_name \rangle : \langle field\_value \rangle \)

The most commonly specified fields are as follows:

Name The name of the RPM package. As explained in §2.1, we recommend forming the RPM package name by combining your \TeX{} distribution name with the name of the \TeX{} package. For example, a \texttt{breakurl} RPM for \TeX{} would be called \texttt{tetex-breakurl}.

Summary A concise, one-line summary of the \TeX{} package.

Version The version number of the \TeX{} package. Normally this will be found in a \texttt{README} file or in the package's documentation, though in some cases you may need to examine the package source code. Sometimes a package will have a date but no formal version number; in these cases you should use the date, in the format \( \langle YYYYMMDD \rangle \), as the version number.

Release The release number of the RPM package. Initially, this should be 1; every time you rebuild the RPM package (say, to fix an error in the \texttt{spec} file), this number should be incremented. Release numbers are specific to each version of the \TeX{} package, so whenever you prepare a \texttt{spec} file for a new version of the same \TeX{} package, the release number should be reset to 1.

License The license under which the \TeX{} package is released. Normally this information will be found in a file named \texttt{README} or \texttt{COPYING}, or in the package documentation. Typically the value for the \texttt{License} field will be \texttt{LPPL} (\LaTeX{} Project Public License), though some packages are released other ways, such as under the GNU Public License (GPL) or as public domain. If the package is released under a custom or unusual license with no common abbreviation, then it's best to write here something like \texttt{Other} or \texttt{See package docs}.

Group The category to which this package belongs. Different OS distributions have different categorization schemes, so you will need to consult your distribution's documentation. Table 1 lists the groups where \TeX{} packages go for some common GNU/Linux distributions.

URL The home page of the \TeX{} package. Normally this will be the package's location on CTAN.

Requires Any software or other RPM package required for this package to work. At a minimum, this field should contain the name of the \TeX{} distribution you are using. You can also specify that a certain minimum (or exact) version of a package is required—for example:

\texttt{Requires: tetex \textasciitilde 2.0.2}

You can use as many \texttt{Requires} fields as there are prerequisites for your package, or you can use a single \texttt{Requires} field and separate the values with commas.

Distribution The name and version of the OS distribution for which this RPM is intended. This information is used by RPM search engines to properly categorize your package. It is possible to build RPMs for a distribution other than the one you are currently running, though this will require some knowledge of where it expects the local TDS tree to be rooted.

Source The source archive used to build the package. This basically corresponds to the \texttt{zip} or \texttt{tar.gz} file you downloaded from CTAN. However, it is generally expected that the source be archived as a \texttt{tar.bz2} file and given a standard name of the form \texttt{(name)-(version).tar.bz2}, where \texttt{(name)} and \texttt{(version)} are the name and version, respectively, of the \TeX{} package.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|l|}
\hline
Distribution & Group & Reference \\
\hline
Fedora Core  & Applications/Publishing & [1, §13.2.2] \\
PLD Linux    & Applications/Publishing/\TeX{} & [1, §13.2.2] \\
Red Hat      & Applications/Publishing       & [7, §2.5]   \\
SUSE         & Productivity/Publishing/\TeX{}/Base  & & \\
Yellow Dog   & Applications/Publishing       & & \\
\hline
\end{tabular}
\caption{Groups for \TeX{} packages by GNU/Linux distribution}
\end{table}

8 If you are planning on making your RPM package available to the public, be sure to first check that the license allows it. Most free software licenses, including the LPPL and GPL, permit this, though some other licenses may stipulate that the software cannot be repackaged or redistributed.

9 This is especially true if you intend to distribute "source" RPMs as well as binary RPMs—see §3.4.
<table>
<thead>
<tr>
<th>Name: tetex-breakurl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary: An extension to hyperref for line-breakable urls in DVIs</td>
</tr>
<tr>
<td>Version: 0.04</td>
</tr>
<tr>
<td>Release: 1</td>
</tr>
<tr>
<td>License: LPPL</td>
</tr>
<tr>
<td>Group: Productivity/Publishing/TeX/Base</td>
</tr>
<tr>
<td>URL: <a href="http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/">http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/</a></td>
</tr>
<tr>
<td>Requires: tetex</td>
</tr>
<tr>
<td>Distribution: SuSE 9.0 (i586)</td>
</tr>
<tr>
<td>Source: %{name}-%{version}.tar.bz2</td>
</tr>
<tr>
<td>BuildRoot: %{_tmppath}/%{name}-%{version}-root</td>
</tr>
<tr>
<td>BuildArch: noarch</td>
</tr>
</tbody>
</table>

%define texmf /usr/local/share/texmf

%description
This package provides a command much like hyperref's \url that typesets a URL using a typewriter-like font. However, if the dvips driver is being used, the original \url doesn't allow line breaks in the middle of the created link: the link comes in one atomic piece. This package allows such line breaks in the generated links.

Note that this package is intended only for those using the dvips driver. Users of the pdflatex driver already have this feature.

Listing 2: The Header stanza for breakurl.spec

Assuming we are working with version 0.04 of breakurl, the contents of this field would be tetex-breakurl-0.04.tar.bz2. If we want to use rpmbuild's pregenerated macros, it would be %{name}-%{version}.tar.bz2. Since this file must actually be found by rpmbuild, you will also have to recompress and rename the archive from CTAN, and then move it into the SOURCES subdirectory of your workspace:

```bash
$ cd /tmp
$ gunzip breakurl.tar.gz
$ mv breakurl.tar \ texet-breakurl-0.04.tar
$ bzip2 -9 tetex-breakurl-0.04.tar
$ mv tetex-breakurl-\ 0.04.tar.bz2 ~/rpm/SOURCES
```

BuildArch The computer architecture the package is intended to run on. Since most T\TeX\ packages do not contain any binary computer code, a value of noarch will suffice in most cases.

BuildRoot A temporary directory in which to test installing the package. Normally this will be declared as %{_tmppath}/%{name}-%{version} -root.

%description A detailed description of the T\TeX\ package, possibly several paragraphs in length. Often this information can be copied from the package's README file. (Strictly speaking, the %description is not a field, since it is followed by a newline rather than a colon, and is terminated by the beginning of the Prep stanza.)

These fields can be specified in any order, except that the %description field must come last.

Listing 2 shows a complete Header stanza for breakurl.spec.

3.3.2 The Prep stanza

The Prep stanza, which always begins with the line

```bash
%prep
```

contains macros and/or shell commands which prepare the package for compilation. For T\TeX\ packages, this typically involves merely unpacking the source code archive (specified by the Header stanza's Source field) into the temporary build directory.

The RPM system provides the macro %setup for this purpose; it is usually used with the -q (quiet) option to suppress unwanted output. The %setup macro assumes that the source archive unpacks into a directory named (name)-(version) (the actual name and version being retrieved from the Header stanza) and will cd into it in preparation for the next stanza. If your tarball unpacks into a different directory, then the option -n (dirname) can be used to specify an alternative directory name.

A sample Prep stanza for breakurl.spec appears in Listing 3. Note that we specify the option -n breakurl to %setup. That's because, as we saw in §3.2, the breakurl tarball we downloaded from CTAN unpacked into a directory called simply breakurl.
3.3.3 The Build stanza

The Build stanza begins as follows:

```
%build
```

Following this token you should type whatever shell commands are necessary to build the \TeX package. For most \LaTeX packages, this will involve running `latex` on any `ins` and `dtx` files; it may also involve running `bibtex`, `makeindex`, `dvips`, and other commands. On the other hand, some \LaTeX packages come prebuilt as ready-to-install `sty` and/or `cls` files; in such cases the Build stanza will be empty.

You should consult the package's build instructions to find out which commands, if any, you need to run on which files, and manually try them out yourself on the temporary copy of the package source you unpacked in §3.2. This way you can find out, for example, how many times you need to run `latex` before all references are resolved, and thus include the appropriate number of calls to it in the Build stanza.

A sample Build stanza for `breakurl.spec` appears in Listing 4. Note that the tarball already contained a PDF version of the package documentation, making our processing of `breakurl.dtx` to create the DVI documentation somewhat unnecessary. However, let’s assume for illustrative purposes that we wish to include both PDF and DVI versions of the documentation in our RPM, thus necessitating the generation of the latter.

```
%build
latex breakurl.ins
latex breakurl.dtx
latex breakurl.dtx
```

Listing 4: The Build stanza for `breakurl.spec`

3.3.4 The Install stanza

After the Build stanza comes the Install stanza. Its beginning is denoted by the following token:

```
%install
```

Like the Build stanza, the Install stanza consists of shell commands, though this time the purpose is to copy the generated files into their correct places in the \TeX directory structure (TDS). However, rather than installing the files into the system TDS (which can have disastrous effects if there is an error in the `spec` file), we instead simulate or “practice” installing them into a temporary copy of the file system. This temporary copy was specified by the `BuildRoot` field of the Header stanza, the value of which can be accessed here with the shell variable `$RPM_BUILD_ROOT`.

Another component of the Install stanza is a small script to remove the contents of the `BuildRoot` directory after a build. This script begins with

```
%clean
```

and usually consists of the command

```
rm -rf $RPM_BUILD_ROOT
```

Many packagers prefer to include a copy of this command as the first command after `%install`, just to make sure the `BuildRoot` is empty.

A sample Install stanza for `breakurl.spec` appears in Listing 5.

```
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p %{texmf}/tex/latex/breakurl
cp breakurl.sty %{texmf}/tex/latex/breakurl
mkdir -p %{texmf}/doc/latex/breakurl
cp README %{texmf}/doc/latex/breakurl
cp breakurl.dvi %{texmf}/doc/latex/breakurl
cp breakurl.pdf %{texmf}/doc/latex/breakurl
%clean
rm -rf $RPM_BUILD_ROOT
```

Listing 5: The Install stanza for `breakurl.spec`

3.3.5 The Files stanza

The Files stanza, which begins with the line

```
%files
```

lists all the files and directories to be included in the RPM package. This is important, as during the build process a number of temporary files (e.g., `aux`, `log`, `bbl`) may be created, and `rpmbuild` needs to know that these can be safely discarded.

The main part of the Files stanza is relatively simple; it simply consists of a list of files and directories, relative to the `BuildRoot` directory where they were temporarily installed in the Install stanza. Normally one file or directory per line is specified, though it is permissible to use wildcards. For example, the lines

```
%{texmf}/tex/latex/mypackage/*.sty
%{texmf}/tex/latex/mypackage/*.cls
```

specify including all the \LaTeX style and class files in the directory `$RPM_BUILD_ROOT/%{texmf}/tex/latex/mypackage`. 
Be careful when specifying a directory name, because that indicates to rpmbuild that it should package all files in that directory. If you just want to indicate that a particular directory but none of its files should be packaged, precede the name of the directory with the %dir macro:

%dir %{texmf}/tex/latex/mypackage/tmp

There are other macros you can use before a filename to give important information about the file. The %doc macro indicates that a file is documentation. (Marking this is important because some users might want to save space by not installing the package’s documentation. The rpm --install command has an auxiliary option, --excludedocs, to suppress installation of documentation.) Alternatively, the %config macro can be used to mark a file as being a user-modifiable configuration file. When upgrading a package, rpm will be careful not to overwrite any such file the user may have painstakingly modified. Here are some examples:

%doc %{texmf}/doc/latex/foo/guide.dvi
%doc %{texmf}/doc/latex/foo/README
%config %{texmf}/tex/latex/foo/foo.cfg

Finally, it is important to set the ownership and permissions for the files to be installed. This can be done collectively for all files by issuing the %defattr macro at the beginning of the Files stanza, which has this syntax (on two lines only for this presentation; the actual source must be all on one line):

%defattr(file_permissions),<owner>,
   (group),(directory_permissions)

Both <owner> and <group> will normally be root. You should specify the file and directory permissions in the standard three-digit octal format used by chmod. For example, one might specify the directory permissions as 755, which corresponds to rwxr-xr-x (i.e., everyone can read and execute the directory, but only its owner can write to it). Instead of specifying an octal value, you can use the hyphen, -, to tell rpmbuild to package the files and directories with the same permissions it has in the BuildRoot tree. To override the default permissions or ownership on a particular file, you can prefix it with the %attr macro, which uses the same syntax as %defattr.

A sample Files stanza for breakurl.spec appears in Listing 6.

3.3.6 The Scripts stanza

Sometimes, certain system commands need to be executed before or after software is installed or uninstalled. This is also true of (un)installing packages on most TeX distributions, usually because TeX has to update its file index. On TeX and other TeX distributions which use the Kpathsea path searching library, for example, the command texhash or mktexslr must be run whenever a package is installed or uninstalled. Such commands can be specified in the Scripts stanza of the spec file.

Unlike most of the previous stanzas, the beginning of the Scripts stanzas is not marked by a token. Rather, scripts to be executed before installation, after installation, before uninstallation, and after uninstallation can be specified following the %pre, %post, %preun, and %postun tokens, respectively. Listing 7 shows the Script stanza for our breakurl.spec file.

3.3.7 The Changelog stanza

The Changelog stanza is where you should keep a human-readable log of changes to the spec file. This stanza begins with the line

%changelog

and contains a chronological list of entries (most recent first) in this format:

* <date> <name> <email> (version)-(release)
- <change made>
- <other change made>...

The <date> field must be in the following format:

Tue Jul 05 2005

Such a date can be produced with the command

$ date +"%a %b %d %Y"

The initial Changelog stanza for breakurl.spec is illustrated in Listing 8.
3.4 Building the RPM

Now that you’ve written the spec and moved the source tarball (recompressed with bzip2) into your “~/rpm/SOURCES” directory, you are finally ready to build the RPM. Simply cd into the directory where “breakurl.spec” resides (“~/rpm/SPRCS”) and run the following command (logged into your regular account—not as root!):

```
$ rpmbuild -ba breakurl.spec
```

`rpmbuild` will first scrutinize the `spec` file for any syntax errors and abort with an informative message if it finds any. If not, you should see the commands you’ve specified in the Build and Install stanzas being executed as if you had typed them yourself. You’ll probably get several pages of BtEx output plus various other diagnostic messages from `rpmbuild` itself. The resulting RPM package will be written to the file

```
~/rpm/RPMS/centos/tetex-breakurl-0.04-1.rpm
```

Go ahead and examine the file with `less`, or with `rpm --query --info --list --package`, to make sure that all the package data is correct and that all files are set to install in the correct places. (The data should look similar to what is shown in Figure 2.) If not, you’ll have to figure out where the error is, re-edit the `spec` file, increment its `Release` number, and try rebuilding.\(^\text{10}\)

Another file produced by `rpmbuild`,

```
~/rpm/SRPMS/tetex-breakurl-0.04-1.src.rpm
```

is what is known as a source RPM or SRPM. Unlike the RPM package, which contains only the precompiled, ready-to-install BtEx package, the SRPM contains the original source tarball and your `spec` file. SRPMs are useful for users who want to modify the BtEx package before it is compiled, or for those who wish to create a new RPM of the same BtEx package for a different OS or BtEx distribution and don’t want to go to the trouble of creating their own `spec` file from scratch.

Note that `rpmbuild` does not, by default, install the RPM package; it only creates one. To actually install the RPMs you create, you need to invoke the `rpm` command as outlined in §2.2.

Once you have created and tested your RPM, it might be a good idea to save other users the trouble you’ve gone to by publishing it on the web or on a local FTP server. Eventually it will probably be spidered by an RPM search engine such as `rpmfind.net` so that others can find it. It is also possible that CTAN may accept submissions of RPM packages, either now or at some point in the future.

4 Advanced topics

The information presented in this article is sufficient for making basic RPMs for most BtEx packages, though there are many other topics which are not addressed here. Most importantly, this article has assumed that the BtEx package you are packaging is something like a font or a BtEx package which contains no executable code. Utilities written in programming languages such as Perl, Python, or C, or those which make use Makefiles or the GNU Auto tools must be handled slightly differently: though the process isn’t necessarily more complicated, there are too many different cases to cover in an article of this scope. The reader is therefore referred to more general-purpose documents on RPM [1, 2, 3] for dealing with such packages.

Another topic worthy of mention is the use of cryptographic tools such as PGP and GNUPG to digitally sign and verify RPMs. RPM has built-in support for this, though because there are many more applications for digital signatures in the world of BtExing, we will reserve treatment of it for a future article on using GNUPG with BtEx.

5 Bibliography


---

<table>
<thead>
<tr>
<th>Name: tetex-breakurl</th>
<th>Tristan Miller</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary: An extension to hyperref for line-breakable urls in DVIs</td>
<td>German Research Center for Artificial Intelligence (DFKI GmbH)</td>
</tr>
<tr>
<td>Version: 0.04</td>
<td>Postfach 20 80</td>
</tr>
<tr>
<td>Release: 2</td>
<td>67608 Kaiserslautern, Germany</td>
</tr>
<tr>
<td>License: LPPL</td>
<td><a href="mailto:Tristan.Miller@dfki.de">Tristan.Miller@dfki.de</a></td>
</tr>
<tr>
<td>Group: Productivity/Publishing/\TeX/Base</td>
<td><a href="http://www.dfki.uni-kl.de/~miller/">http://www.dfki.uni-kl.de/~miller/</a></td>
</tr>
<tr>
<td>URL: <a href="http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/">http://www.ctan.org/tex-archive/macros/latex/contrib/breakurl/</a></td>
<td></td>
</tr>
<tr>
<td>Requires: tetex</td>
<td></td>
</tr>
<tr>
<td>Distribution: SuSE 9.0 (i586)</td>
<td></td>
</tr>
<tr>
<td>Source: %{name}-%{version}.tar.bz2</td>
<td></td>
</tr>
<tr>
<td>BuildRoot: %_tmppath%/%{name}-%{version}-root</td>
<td></td>
</tr>
<tr>
<td>BuildArch: noarch</td>
<td></td>
</tr>
<tr>
<td>%define texmf /usr/local/share/texmf</td>
<td></td>
</tr>
<tr>
<td>%description</td>
<td></td>
</tr>
<tr>
<td>This package provides a command much like hyperref’s \url that typesets a URL using a typewriter-like font. However, if the dvips driver is being used, the original \url doesn’t allow line breaks in the middle of the created link: the link comes in one atomic piece. This package allows such line breaks in the generated links.</td>
<td></td>
</tr>
<tr>
<td>Note that this package is intended only for those using the dvips driver. Users of the pdflatex driver already have this feature.</td>
<td></td>
</tr>
</tbody>
</table>

%prep
%setup -q -n breakurl
%build
latex breakurl.ins
latex breakurl.dtx
latex breakurl.dtx
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%(texmf)/tex/latex/breakurl
cp breakurl.sty $RPM_BUILD_ROOT/%(texmf)/tex/latex/breakurl
mkdir -p $RPM_BUILD_ROOT/%(texmf)/doc/latex/breakurl
cp README breakurl.(dvi,pdf) $RPM_BUILD_ROOT/%(texmf)/doc/latex/breakurl
%clean
rm -rf $RPM_BUILD_ROOT
%files
%defattr(-,root,root,-)
%(texmf)/tex/latex/breakurl/breakurl.sty
%doc %(texmf)/doc/latex/breakurl/README
%doc %(texmf)/doc/latex/breakurl/breakurl.dvi
%doc %(texmf)/doc/latex/breakurl/breakurl.pdf
%changelog
* Mon Jul 4 2005 Tristan Miller <psychonaut@nothingisreal.com> 0.04-1
  - Initial build.

Listing 9: The complete breakurl.spec file
Acknowledgements

Special thanks to all the speakers and teachers, without whom there would be no conference, and also:

- Chapel Hill Visitor’s Bureau, for providing local assistance
- Nelson Beebe, for agreeing to be the keynote speaker
- Duane Bibby, for the always excellent drawings.
- Joseph Hogg, for his extra donations to the raffle
- Wendy McKay, for organizing the Mac OS X BOF

Conference committee

Karl Berry ■ Lance Carnes ■ Sue DeMeritt ■ Steve Grathwohl ■
Robin Laakso ■ Steve Peter ■ Cheryl Ponchin

Participants

David Allen, University of Kentucky
Tim Arnold, SAS
Kapila Attele, Chicago State University
Kaveh Bazargan, River Valley Technologies
Nelson Beebe, University of Utah
Karl Berry, TeX Users Group
John Burt, Brandeis University
Lance Carnes, Personal TeX Inc.
Rialine Cruywagen, Unisa, South Africa
Em Van Deventer, Unisa, South Africa
Trinette JeAnne Evert, Unisa, South Africa
Ronald Feld, Centers for Disease Control and Prevention
Frances Felhuca, INFORMS
Peter Flom, National Development and Research Institutes
Peter Flynn, Simlaril Consultants
August Gering, Duke University Press
Steve Grathwohl, Duke University Press & TUG
Eitan Gurari, Ohio State University
Hans Hagen, Pragma ADE & NTG
Joseph Hogg, Los Angeles, CA
Leslie Holbrook, Oak Ridge National Laboratory
Klaus Höppner, DANTE e.V. & TUG
David Ignat, International Atomic Energy Agency
Calvin Jackson, Caltech
Mirko Jane, INFORMS
Jonathan Kew, SIL International

Richard Koch, University of Oregon
Martha Kummerer, University of Notre Dame
Robin Laakso, TeX Users Group
Jenny Levine, Duke University Press
Barbara Mastrian, Rutgers University
Wendy McKay, Caltech
Marisa Meredith, University of North Carolina
Andrew Mertz, Eastern Illinois University
Tristan Miller, German Research Center for Artificial Intelligence
Jaime Moore, Pacific Northwest National Laboratory
Stephen Moye, American Mathematical Society
Liz Pennington, University of North Carolina
Steve Peter, Beech Stave Press & TUG
Cheryl Ponchin, Center for Communications Research, Princeton, NJ
John Rorem, Duke University Press
Volker R.W. Schaa, DANTE e.V.
Anita Schwartz, University of Delaware
Heidi Sestrich, Carnegie-Mellon University
William Slough, Eastern Illinois University
Alistair Smith, Sunrise Setting Ltd
Terri Spence, Duke University Press
Chris Swanepeol, Unisa, South Africa
Lee Trimble, University of North Carolina
Gary Tucker, Averett University
| Tuesday  
| June 14 | **courses** | 9 am - 4:30 pm |
|         | Peter Flynn | Practical TeX on the web |
|         | Steve Peter | Introduction to ConTeXt |
|         | Cheryl Ponchin | Beginning and intermediate \LaTeX |
| 8-9 am | registration |
| 10:30 am | break |
| 12:30 pm | lunch (until 1:30 pm) |
| 3 pm | break |
| 5-7 pm | registration & reception |

| Wednesday  
| June 15 | **registration** |
| 8-9 am | Karl Berry, TeX Users Group |
| 9 am | Nelson Beebe, University of Utah |
| 10:15 am | break |
| 11 am | Anita Schwartz, University of Delaware |
| 11:45 am | Kaveh Bazargan, River Valley Tech. |
| 12:30 pm | lunch |
| 1:30 pm | David Ignat, IAEA |
| 2 pm | Steve Grathwohl, Duke University Press |
| 2:30 pm | Ronald Fehd, CDC |
| 3 pm | break |
| 3:15 pm | Jonathan Kew, SIL International |
| 4 pm | Q&A |
| 4:30 pm | *Birds of a Feather* (see following page) |

| Thursday  
| June 16 | 9 am | Eitan Gurari, Ohio State University |
| 9:45 am | John Burt, Brandeis University |
| 10:30 am | break |
| 10:45 am | Joseph Hogg, Los Angeles |
| 11:30 am | Klaus Höppner, DANTE e.V. & TUG |
| 12:30 pm | lunch |
| 1:30 pm | Tristan Miller, DFKI |
| 2:15 pm | David Allen, University of Kentucky |
| 3 pm | break |
| 3:15 pm | Andrew Mertz & William Slough, Eastern Illinois University |
| 4 pm | Q&A |
| 4:30 pm | *TUG members meeting* |
| 7:30 pm | banquet (see following page) |

| Friday  
| June 17 | 9 am | Tristan Miller |
| 9:45 am | Volker R.W. Schaa, DANTE e.V. |
| 10:30 am | break |
| 10:45 am | Hans Hagen, Pragma ADE & NTG |
| 11:30 am | lunch |
| 12:30 pm | Steve Peter |
| 1:30 pm | Mirko Janc, INFORMS |
| 2:15 pm | Peter Flynn, Silmaril Consultants |
| 3 pm | break |
| 3:15 pm | panel: Digital Publishing |
| ≈ 4 pm | end |

*Welcome keynote address: The design of TeX and METAFONT: A retrospective*  

*Word to \LaTeX for a large, multi-author scientific paper*  

*Dynamic presentations using \TeXpower and PSTricks*  

*Biblet: A portable \BibTeX bibliography style for generating highly customizable XHTML*  

*XML workflows and the Euro\TeX 2005 proceedings*  

*TEX font installation and usage*  

 moderator: *Steve Grathwohl; Kaveh Bazargan, Nelson Beebe, Lance Carnes, Peter Flynn, Hans Hagen, Mirko Janc*
Impressions from PracT\TeX’05

Compiled by Peter Flom and Tristan Miller

The recent PracT\TeX 2005 conference in Chapel Hill, North Carolina was a great success, thanks to the presenters, the attendees, and the local organizer, Steve Grathwohl of Duke University Press.

Below are a few photos and comments from some of the attendees. More information and photos are on the conference web pages (http://tug.org/practicaltex2005).

Cheryl Ponchin, Heidi Sestrich, Kaveh Bazargan, Chris Swaepoel, Em Van Deventer, Trinette Jeanne Evert, Alistair Smith, and Rialine Cruywagen

John Burt (Brandeis University, USA)
What a wonderful conference that was! I found it incredibly informative, and incredibly fun. The papers were very exciting to me, particularly those that brought up aspects of \TeX that were new to me, such as the papers on HA-prosper and Beamer, the paper on the graphical interface, and the paper on Xe\TeX. I enjoyed also the course on Con\TeXt a great deal, and was persuaded by it to give it a whirl.

One of the best things about the conference was the social side of it — not just the delicious lunches and the even more delicious conversations, and the banquet, but the informal getting-together over dinner. I can’t think of a conference I have enjoyed more!

Ron Fehd (Centers for Disease Control and Prevention, Atlanta, USA)
I came to Prac\TeX for the first time in San Francisco in 2004 and thoroughly enjoyed it. This year at Prac\TeX in Chapel Hill, NC, I am pleased to report that I got my money’s worth each and every day. Let me restate that: I would have paid the full conference fee for the knowledge I gained on any of the four days!

I have been using SAS (Statistical Analysis System) for almost 20 years and for the past four years I have been using \TeX to write a book about my SAS programming expertise. As a result of attending Prac\TeX conferences I have learned new techniques that will make this a better book.

Based on my return on investment and continued use of \TeX: see you next year, for sure.

Peter Flom (National Development and Research Institutes, New York, USA)
I attended the recent Prac\TeX conference and had a great time. I gave a talk, “A \TeX fledgling struggles to take flight”, which was well-received. I learned a lot from the \TeX class I took, the presentations I attended, and from conversations with various people. Despite my newbie status (I’ve been using \TeX for all of seven months), and despite the fact that many of the people had clearly known each other for years, people could not have been more welcoming. Several people made deliberate efforts to include me in conversations and extra-conference activities. I’m just sorry I had to leave early.

I encourage other beginning users of \TeX to attend future conferences; it’s a great way to learn more about the program, to meet a good group of people, and to put faces to the names.

Lance Carnes and Cheryl Ponchin

Joe Hogg (Los Angeles, USA)
I’ve been using \TeX for about a year for booklets, a couple of leaflets and business correspondence. It’s been rewarding and many persons who have seen
these documents have liked them. Like many individuals using \TeX{}, and probably most new users, I’ve worked alone using \TeX{} resources in print and at various web sites.

The Practical \TeX{} 2005 conference in Chapel Hill gave me a better sense of the variety of applications being created and the expertise of individuals using this software. The attendees ranged from computer scientists to end-users like me, individuals from private and university presses, academic type-setters, typesetting-system developers and a nice mix of electronic publishing practitioners involved with both web and print venues.

Besides its conviviality, I value the conference for the new areas it gave me to explore and how what I’ve learned will change the way I use \TeX{}. Since returning home, I’ve been looking at Con\TeX{}t and thinking about how I can integrate XML, the web and \TeX{} for two projects I’ve wanted to tackle for months. I’ll also read \textit{TUGboat} and \textit{The Prac\TeX{} Journal} with particular interest since I met many of the authors whose articles appear there. Very worthwhile conference.

\textbf{Tristan Miller} (German Research Center for Artificial Intelligence, Germany)

Though I have been using \LaTeX{} for a number of years now, I joined TUG only last November, and Practical \TeX{} 2005 was the first \TeX{}-related conference I have ever attended. The thing that impressed me most about the conference was the sheer diversity of its attendees — there were computer scientists, mathematicians, statisticians, physicists, linguists, publishers, writers, programmers, and even a volunteer zoo worker! The consequence of this variety was that we were all pretty much on equal footing — though some of us had more \TeX{} experience than others, we all seemed to have our own unique uses for it and therefore had something new and valuable to share with the others. I was also delighted to see how friendly and open everyone was with each other. Even though I was a newcomer and had never met the other participants before, I was received warmly and sensed a universal collegiality that is often lacking at larger academic conferences. Having been made to feel so welcome and having learned so much from the other participants, I am eagerly looking forward to attending the next Practical \TeX{} and other TUG conferences.
The design of $\TeX$ and $\mathsf{METAFONT}$: A retrospective

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148
Internet: beebe@math.utah.edu, beebe@acm.org, beebe@computer.org

Abstract

This article looks back at the design of $\TeX$ and $\mathsf{METAFONT}$, and analyzes how they were affected by architectures, operating systems, programming languages, and resource limits of the computing world at the time of their creation by a remarkable programmer and human being, Donald E. Knuth. This paper is dedicated to him, with deep gratitude for the continued inspiration and learning that I've received from his software, his scientific writing, and our occasional personal encounters over the last 25+ years.

1 Introduction

More than a quarter century has elapsed since Donald Knuth took his sabbatical year of 1977–78 from Stanford University to tackle the problem of improving the quality of computer-based typesetting of his famous book series, *The Art of Computer Programming* [53–58, 60, 65–67].

When the first volume appeared in 1968, most typesetting was still done by the hot-lead process, and expert human typographers with decades of experience handled line breaking, page breaking, and page layout. By the mid 1970s, proprietary computer-based typesetting systems had entered the market, and in the view of Donald Knuth, had seriously degraded quality. When the first page proofs of part of the second edition of Volume 2 arrived, he was so disappointed that he wrote [68, p. 5]:

> I didn't know what to do. I had spent 15 years writing those books, but if they were going to look awful I didn't want to write any more. How could I be proud of such a product?

A few months later, he learned of some new devices that used digital techniques to create letter images, and the close connection to the 0s and 1s of computer science led him to think about how he himself might design systems to place characters on a page, and draw the individual characters as a matrix of black and white dots. The sabbatical-year project produced working prototypes of two software programs for that purpose that were described in the book *$\TeX$ and $\mathsf{METAFONT}$: New Directions in Typesetting* [59].

The rest is of course history [6] ... the digital typesetting project lasted about a decade, produced several more books [64, 68–73], Ph.D. degrees for Frank Liang [79, 80], John Hobby [36], Michael Plass [88], Lynn Ruggles [92], and Ignacio Zabala Salelles [110], and had spinoffs in the commercial document-formatting industry and in the first laser printers. $\TeX$, and the $\LaTeX$ system built on top of it [20–22, 76, 77, 83], became the standard markup and typesetting system in the computer science, mathematics, and physics communities, and have been widely used in many other fields.

---

1 Introduction

More than a quarter century has elapsed since Donald Knuth took his sabbatical year of 1977–78 from Stanford University to tackle the problem of improving the quality of computer-based typesetting of his famous book series, *The Art of Computer Programming* [53–58, 60, 65–67].

When the first volume appeared in 1968, most typesetting was still done by the hot-lead process, and expert human typographers with decades of experience handled line breaking, page breaking, and page layout. By the mid 1970s, proprietary computer-based typesetting systems had entered the market, and in the view of Donald Knuth, had seriously degraded quality. When the first page proofs of part of the second edition of Volume 2 arrived, he was so disappointed that he wrote [68, p. 5]:

> I didn't know what to do. I had spent 15 years writing those books, but if they were going to look awful I didn't want to write any more. How could I be proud of such a product?

A few months later, he learned of some new devices that used digital techniques to create letter images, and the close connection to the 0s and 1s of computer science led him to think about how he himself might design systems to place characters on a page, and draw the individual characters as a matrix of black and white dots. The sabbatical-year project produced working prototypes of two software programs for that purpose that were described in the book *$\TeX$ and $\mathsf{METAFONT}$: New Directions in Typesetting* [59].

The rest is of course history [6] ... the digital typesetting project lasted about a decade, produced several more books [64, 68–73], Ph.D. degrees for Frank Liang [79, 80], John Hobby [36], Michael Plass [88], Lynn Ruggles [92], and Ignacio Zabala Salelles [110], and had spinoffs in the commercial document-formatting industry and in the first laser printers. $\TeX$, and the $\LaTeX$ system built on top of it [20–22, 76, 77, 83], became the standard markup and typesetting system in the computer science, mathematics, and physics communities, and have been widely used in many other fields.
The purpose of this article is to look back at \TeX{} and METAFONT and examine how they were shaped by the attitudes and computing environment of the time.

2 Computers and people

Now that computers are widely available throughout much of the developed world, and when embedded systems are counted, are more numerous than humans, it is probably difficult for younger people to imagine a world without computers readily at hand. Yet not so long ago, this was not the case.

Until the desktop computers of the 1980s, a ‘computer’ usually meant a large expensive box, at least as long as an automobile, residing in a climate-controlled machine room with raised flooring, and fed electricity by power cables as thick as your wrist. At many universities, these systems had their own buildings, or at least entire building floors, called Computer Centers. The hardware usually cost hundreds of thousands to millions of dollars (where according to the US Consumer Price Index, a million dollars in 1968 is roughly the same as five million in 2000), and required a full-time professional staff of managers, systems programmers, and operators.

At most computer installations, the costs were passed on to users in the form of charges, such as the US$1500 per hour for CPU time and US$0.50 to open a file that I suffered with as a graduate student earning US$1.50 per hour. At my site, there weren’t any disk-storage charges, because it was forbidden to store files on disk: they had to reside either on punched cards, or on reels of magnetic tape. A couple of years ago, I came across a bill from the early 1980s for a 200MB disk: the device was the size of a washing machine, and cost US$15 000. Today, that amount of storage is about fifty thousand times cheaper, and disk-storage costs are likely to continue to drop.

I have cited these costs to show that, until desktop computers became widespread, it was people who worked for computers, not the reverse. When a two-hour run cost as much as your year’s salary, you had to spend a lot of time thinking about your programs, instead of just running them to see if they worked.

When I came to Utah in 1978, the College of Science that I joined had just purchased a DECsystem 20, a medium-sized timesharing computer based on the DEC PDP-10 processor, and the Department of Computer Science bought one too on the same order. Ours ultimately cost about $750 000, and supplied many of the computing needs of the College of Science for more than a dozen years, often supporting 50–100 interactive login sessions. Its total physical memory was just over three megabytes, but we called it three quarters of a megaword. We started in 1978 with 400MB of disk storage, and ended in 1990 with 1.8GB for the entire College. Although computer time was still a chargeable item, we managed to recover costs by getting each Department to contribute a yearly portion of the expenses as a flat fee. The operating system’s class scheduler guaranteed departmental users a share of the machine in proportion to their fraction of the budget. Thus, most individual users didn’t worry about computer charges.

3 The DEC PDP-10

The PDP-10, first released in 1967, ran at least ten or eleven different operating systems:

- BBN TENEX,
- Compuserve modified 4S72,
- DEC TOPS-10 (sometimes humorously called BOTTOMS-10 by TOPS-20 users), and just called the MONITOR before it was trademarked,
- DEC TOPS-20 (a modified TENEX affectionately called TWENEX by some users),
- MIT ITS (Incompatible Timesharing System),
- Carnegie-Mellon University (CMU) modified TOPS-10,
- On-Line Systems’ OLS-10,
- Stanford WAITS (Westcoast Alternative to ITS),
- Tymshare AUGUST (a modified TENEX),
- Tymshare TYMCOM-X, and on the smaller DECSYSTEM 20/20 model, TYMCOM-XX.

Although the operating systems differed, it was usually possible to move source-code programs among them with few if any changes, and some binaries compiled on TOPS-10 in 1975 still run just fine on TOPS-20 three decades later (see Section 3).

Our machines at Utah both used TOPS-20, but Donald Knuth’s work on \TeX{} and METAFONT was done on WAITS. That system was a research operating system, with frequent changes that resulted in bugs, causing many crashes and much downtime. Don told me earlier this year that the O/S was aptly named, since he wrote much of the draft of The \TeX{}book while he was waiting in the Computer Center for WAITS to come back up. By contrast, apart from hardware-maintenance sessions in a four-hour block each week, the Utah TOPS-20 systems were rarely down.

For about a decade, PDP-10 computers formed the backbone of the Arpanet, which began with
just five nodes, at the University of California campuses at Berkeley, Los Angeles, and Santa Barbara, plus SRI (Stanford Research Institute) and Utah, and later evolved into the world-wide Internet [24, p. 48]. PDP-10 machines were adopted by major computer-science departments, and hosted or contributed to many important developments, including at least these:

- Bob Metcalf's Ethernet [Xerox PARC, Intel, and DEC];
- Vinton Cerf's and Robert Kahn's invention of the Transmission Control Protocol and the Internet Protocol (TCP/IP);
- the MACSYMA [MIT], REDUCE [Utah] and MAPLE [Waterloo] symbolic-algebra languages;
- several dialects of LISP, including MACLISP [MIT] and PSL (Portable Standard Lisp) [Utah];
- the systems-programming language BLISS [DEC and CMU];
- the shell-scripting and systems-programming language PCL (Programmable Command Language) [DEC, CMU, and FUNDP] [94];
- Dan Swinehart's and Bob Sproull's SAIL (Stanford Artificial Intelligence Language) Algol-family programming language in which \( \TeX \) and METAFONT were first implemented;
- an excellent compiler for PASCAL [Hamburg/Rutgers/Sandia], the language in which \( \TeX \) and METAFONT were next implemented;
- Larry Tesler's PUB document formatting system [101] [PUB was written in SAIL, and had a macro language based on a SAIL subset];
- Brian Reid's document-formating and bibliographic system, SCRIBE [89, 90] [CMU], that heavily influenced the design of LATEX and BiLATEX [although SAIL co-architect Bob Sproull was Brian's thesis advisor, Brian wrote SCRIBE in the locally-developed BLISS language];
- Richard Stallman's extensible and customizable text editor, emacs [MIT];
- Jay Lepreau's port, pcc20 [Utah], of Steve Johnson's Portable C Compiler, pcc [Bell Labs];
- Kok Chen's and Ken Harrenstien's kcc20 native C compiler [SRI];
- Ralph Gorin's spell, one of the first sophisticated interactive spelling checkers [Stanford];
- Mark Crispin's mail client, mm, still one of the best around [Stanford];
- Will Crowther's adventure, Don Daglow's baseball and dungeon, Walter Bright's empire, and University of Utah student Nolan Bushnell's pong, all developed on PDP-10s, were some of the earliest computer games [Bushnell went on to found Atari, Inc., and computer games are now a multi-billion-dollar world-wide business driving the computer-chip industry to ever-higher performance];
- part of the 1982 DISNEY science-fiction film TRON was rendered on a PDP-10 clone [curiously, that architecture has a TRON instruction (Test Right-halfword Ones and skip if Not masked) with the numeric operation code 666, leading some to suggest a connection with the name of the film, or the significance of that number in the occult];
- Frank da Cruz's transport- and platform-independent interactive and scriptable communications software kermit [Columbia];
- Gary Kildall's [105] CP/M, the first commercial operating system for the Intel 8080, was developed using Intel's 8080 simulator on the PDP-10 at the Naval Postgraduate School in Monterey, California;
- Harvard University student Paul Allen's Intel 8080 simulator on the PDP-10 was used by fellow student Bill Gates to develop a BASIC-language interpreter before Intel hardware was available to them. [Both had worked on PDP-10 systems in Seattle and Portland in the late 1960s and early 1970s while they were still in school. They later founded Microsoft Corporation, and borrowed ideas from a subset of Kildall's CP/M for their MS-DOS. While IBM initially planned to offer both systems on its personal computer that was introduced in August 1981, pricing differences soon led to its dropping CP/M.]

Notably absent from this list is the Bell Laboratories project that led to the creation of the UNIX operating system: they wanted to buy or lease a PDP-10, but couldn't get the funding [93, Chapter 5]. The PDP-10 and its operating systems is mentioned in about 170 of the now nearly 4000 Request for Comments (RFC) documents that informally define the protocols and behavior of the Internet.

The PDP-10 had compilers for ALGOL 60, BASIC, BLISS, C, COBOL 74, FORTH, FORTRAN 66, FORTRAN 77, LISP, PASCAL, SAIL, SIMULA 67, and SNOBOL, plus three assemblers called MACRO, MIDAS, and FAIL (fast one-pass assembler). A lot of programming was done in assembly code, including that for most of the operating systems. Indeed, the abstract of the FAIL manual [108] notes:
Although \texttt{FAIL} uses substantially more main memory than \texttt{MACRO-10}, it assembles typical programs about five times faster. \texttt{FAIL} assembles the entire Stanford time-sharing operating system (two million characters) in less than four minutes of CPU time on a \texttt{KA-10} processor.

The \texttt{KA-10} was one of the early \texttt{PDP-10} models, so such performance was quite impressive. The high-level \texttt{BLISS} language \cite{9,10,109} might have been preferred for such work, but it was comparatively expensive to license, and few sites had it. Anyway, Ralph Gorin\textquotesingle s book on assembly language and systems programming \cite{23} provided an outstanding resource for programmers.

Given the complexity of most assembly languages, it is instructive to look at the short example in Figure 1 that helps to illustrate why the \texttt{PDP-10} assembly language was so popular among its users.

![Figure 1: MACRO-10 assembly language for the PDP-10 and its high-level pseudo-language equivalent, adapted from \cite{15}](image)

You can understand the assembly code once you know the instruction mnemonics: \texttt{CAML} (Compare Accumulator with Memory and skip if Low) handles the conditional, \texttt{HRRROI} (Half word Right to Right, Ones, Immediate) constructs a 7-bit byte pointer in an 18-bit address space, \texttt{SETOM} (Set to Ones Memory) stores a negative integer one, \texttt{SETZM} (Set to Zeros Memory) stores a zero, \texttt{AOS} (Add One to Sel) increments the stack pointer (P), \texttt{PUSHJ} and \texttt{POPJ} handle stack-based call and return, and \texttt{PSOUT} is a system call to print a string. Brackets delimit remote code and data blocks.

The prevalence of instructions that manipulate 18-bit addresses makes it hard to generalize assembly code for 30-bit extended addressing, but tricks with 18-bit memory segments alleviated this somewhat.

Document formatting was provided by \texttt{runoff}, which shared a common ancestor roff with UNIX troff, and by \texttt{PUB}. Later, \texttt{SCRIBE} became commercially available, but required an annual license fee, and ran only on the \texttt{PDP-10}, so it too had limited availability, and I refused to use it for that reason.

The \texttt{PDP-10} had 36-bit words, with five seven-bit ASCII characters stored in each word. This left the low-order (rightmost) bit unused. It was normally zero, but when set to one, indicated that the preceding five characters were a line number that some editors used, and compilers could report in diagnostics.

Although seven-bit ASCII was the usual \texttt{PDP-10} text representation, the hardware instruction set had general byte-pointer instructions that could reference bytes of any size from 1 to 36 bits, and the \texttt{kcc20} compiler provided easy access to them in C. For interfacing with 32-bit UNIX and VMS systems, 8-bit bytes were used, with four bits wasted at the low end of each word.

The \texttt{PDP-10} filesystems recorded the byte count and byte size for every file, so in principle, text-processing software at least could have handled both 7-bit and 8-bit byte sizes. Indeed, Mark Crispin proposed that Unicode could be nicely handled in 9-bit UTF-9 and 18-bit UTF-18 encodings \cite{13}. Alas, most \texttt{PDP-10} systems were retired before this generality could be widely implemented.

One convenient feature of the \texttt{PDP-10} operating systems was the ability to define directory search paths as values of logical names. For example, in \texttt{TOPS-20}, the command

\begin{verbatim}
@define TEXINPUTS: TEXINPUTS:, ps:<tex.inputs>,
ps:<jones.tex.inputs>
\end{verbatim}

would add a user\textquotesingle s personal subdirectory to the end of the system-wide definition of the search path. The @ character was the normal prompt from the \texttt{EXEC} command interpreter. A subsequent reference to texinputs:myfile.tex was all that it took to locate the file in the search path.

Since the directory search was handled inside the operating system, it was trivially available to all programs, no matter what language they were written in, unlike other operating systems where such searching has to be implemented by each program that requires it. In this respect, and many others, to paraphrase ACM Turing Award laureate Tony Hoare\textquotesingle s famous remark about \texttt{ALGOL} 60 \cite{31}, \texttt{TOPS-20} \textquoteleft was so far ahead of its time that it was not only an improvement on its predecessors, but also on nearly all its successors\textquoteright.\footnote{\texttt{ALGOL} 60 is a higher-level programming language developed in the 1950s by a committee of European and American computer scientists. It was named after Ada Lovelace, a mathematician and the daughter of the poet Lord Byron. It was influential in the development of modern computer programming languages.}

In addition, a manager could readily change the system-wide definition by a single privileged command:

\begin{verbatim}
$^E$define TEXINPUTS: ps:<tex.inputs>,
ps:<tex.new>
\end{verbatim}

The new definition was immediately available to all users, including those who had included the name...
TEXINPUTS: in their own search paths. The $ was the EXEC prompt when a suitably-privileged user had enabled management capabilities.

The great convenience of this facility encouraged those who ported \TeX\ and \METAfont\ to provide something similar. Today, users of the \TeX\ Live distributions are familiar with the kpathsea library, which provides an even more powerful, and customizable, mechanism for path searching.

The original PDP-10 instruction set had an 18-bit address field, giving a memory space of $2^{18}=262,144$ words, or about 1.25MB. Later designs extended the address space to 30 bits (5GB), but only 23 were ever implemented in DEC hardware, giving a practical limit of 40MB. That was still much more than most customers could afford in 1983 when the PDP-10 product line was terminated, and VAX VMS became the DEC flagship architecture and operating system.

The next generation of the PDP-10 was announced to be about ten to fifteen times faster than existing models, but early in 1983, rumors of trouble at DEC had reached the PDP-10 user community. At the Fall 1983 DECUS (DEC User Society) Symposium in Las Vegas, Nevada, that I attended, several PDP-10 devotees sported T-shirts emblazoned with

\begin{verbatim}
I don’t care what they say, 36 bits are here to stay!
\end{verbatim}

They were not entirely wrong, as we shall see.

DEC had products based on the KA-10, KI-10, and KL-10 versions of the PDP-10 processor. Later, other companies produced competing systems that ran one or more of the existing operating systems: Foonly (F1, F2, and F3), Systems Concepts (SC-40), Xerox PARC (MAXC) [16], and XKL Systems Corporation (TOAD-1 for Ten On A Desk). Some of these implemented up to 30 address bits (1GW, or 4.5GB). XKL even made a major porting effort of GNU and \LaTeX\ utilities, and got the X11 WINDOW SYSTEM running. Ultimately, none enjoyed continued commercial success.

The PDP-10 lives on among hobbyists, thanks to Ken Harrenstien’s superb KLIH10 simulator [30] with 23-bit addressing, and the vendor’s generosity in providing the operating system, compilers, documentation, and utilities for noncommercial use. On a fast modern desktop workstation, TOPS-20 runs several times faster than the original hardware ever did. It has been fun revisiting this environment that was such a leap forward from its predecessors, and I now generally have a TOPS-20 window or two open on my UNIX workstation. I even carried this virtual PDP-10 in a laptop to the Practical \TeX\ 2005 conference, and it fits nicely in a memory stick the size of a pocket knife.

4 Resource limits

The limited memory of the PDP-10 forced many economizations in the design of \TeX\ and \METAfont. In order to facilitate possible reimplementation in other languages, all memory management is handled by the programs themselves, and sizes of internal tables are fixed at compile time. Table 1 shows the sizes of those tables, then and now. To further economize, many data structures were stored compactly with redundant information elided. For example, while \TeX\ fonts could have up to 128 characters (later increased to 256), there are only 16 different widths and heights allowed, and one of those 16 is required to be zero. Also, although hundreds of text fonts are allowed, only 16 mathematical families are supported. Ulrik Vieth has provided a good summary of this topic [103].

\begin{table}[h]
\centering
\caption{\TeX\ table sizes on TOPS-20 in 1984 and in \TeX\ Live on UNIX in 2004, as reported in the \trip\ test.}
\begin{tabular}{|l|l|l|l|}
\hline
\textbf{Table} & \textbf{1984} & \textbf{2004} & \textbf{Growth} \\
\hline
strings & 1819 & 98002 & 53.9 \\
string characters & 9287 & 1221682 & 131.5 \\
memory words & 3001 & 1500022 & 499.8 \\
control sequences & 2100 & 60000 & 28.6 \\
font info words & 20000 & 1000000 & 50.0 \\
fops & 75 & 2000 & 26.7 \\
hyphen. exceptions & 307 & 1000 & 3.3 \\
stack positions (i) & 200 & 5000 & 25.0 \\
stack positions (n) & 40 & 500 & 12.5 \\
stack positions (p) & 60 & 600 & 100.0 \\
stack positions (b) & 500 & 200000 & 400.0 \\
stack positions (s) & 600 & 40000 & 66.7 \\
\hline
\end{tabular}
\end{table}

Instead of supporting scores of accented characters, \TeX\ expected to compose them dynamically from an accent positioned on a base letter. That in turn meant that words with accented letters could not be hyphenated automatically, an intolerable situation for many European languages. That restriction was finally removed in late 1989 [63] with the release of \TeX\ version 3.0 and \METAfont\ version 2.0, when those programs were extended to fully support 8-bit characters, and provide up to 256 hyphenation tables to handle multilingual documents. Examination of source-code difference listings shows that about 7% of \TeX\ was changed in this essential upgrade.

The \TeX\ DVI and \METAfont\ GF and TFM files were designed to be compact binary files that re-
quire special software tools to process. Recall from p. 34 that disk storage cost around US$100 per MB, so file compactness mattered! In contrast, in UNIX troff, the corresponding files are generally simple, albeit compact and cryptic, text files to facilitate use of filters in data-processing pipelines. Indeed, the UNIX approach of small-is-beautiful encouraged the use of separate tools for typesetting mathematics [43], pictures [41], and tables [39], each filtering the troff input stream, instead of the monolithic approach that \TeX uses.

In any computer program, when things go awry, before the problem can be fixed, it is essential to know where the failure occurred. The same applies when a change in program behavior is called for: you first have to find the code that must be modified.

In either case, to better understand what is happening, it is very helpful to have a traceback of the routine calls that led to the failure or point of change, and a report of the source-code location where every step in the call history is defined. Unfortunately, PDP-10 memory limitations prevented \TeX and \METAFONT from recording the provenance of every built-in operator and run-time macro, yet every programmer who has written code for these systems has often asked: where is that macro defined, and why is it behaving that way? Although both programs offer several levels of execution tracing, the output trace is often voluminous and opaque, and no macro-level debugger exists for either program.

The need for a record of source-code provenance is particularly felt in the \LaTeX world, where it is common for documents to depend on dozens of complex macro packages collectively containing many tens of thousands of lines of code, and sometimes redefining macros that other loaded packages expect to redefine differently for their own purposes. During the course of writing this article, I discovered, tracked down, and fixed three errors in the underlying \LaTeX style files for the \TeX User Group conference proceedings. Each time, the repairs took much longer than should have been necessary, because I could not find the faulty code quickly.

Finally, error diagnostics and error recovery reflect past technology and resource limits. Robin Fairbairns remarked in a May 2005 \TeXhax list posting:

> Any \TeX-based errors are pretty ghastly. This is characteristic of the age in which it was developed, and of the fiendishly feeble machines we had to play with back then. But they’re a lot better than the first ALGOL 68 compiler I played with, which had a single syntax diagnostic “not a program!”

## 5 Choosing a programming language

When Donald Knuth began to think about the problem of designing and implementing a typesetting system and a companion font-creation system, he was faced with the need to select a programming language for the task. We have already summarized what was available on the PDP-10.

**COBOL** was too horrid to contemplate: imagine writing code in a language with hundreds of reserved words, and such verbose syntax that a simple arithmetic operation and assignment \( c = a \times b \)

becomes

\[ \text{MULTIPLY A BY B GIVING C}. \]

More complex expressions require every subexpression to be given a name and assigned to.

**FORTRAN** 66 was the only language with any hope of portability to many other systems. However, its omission of recursion, absence of data structures beyond arrays, lack of memory management, deficient control structures, record-oriented I/O, primitive Hollerith strings (12HHELLO, 12HWORLD) that could be used only in \DATA and \FORMAT statements and as routine arguments, and its restriction to six-character variable names, made it distinctly unsuitable. Nevertheless, \METAFONT was later translated to **FORTRAN** elsewhere for a port to Harris computers [85].

**PASCAL** only became available on the PDP-10 in late 1978, more than a year after Don began his sabbatical year. We shall return to it in Section 6.

**BLISS** was an expensive commercial product that was available only on DEC PDP-10, PDP-11, and later, VAX, computers. Although DEC subsequently defined **COMMON BLISS** to be used across those very different 16-bit, 32-bit, and 36-bit systems, in practice, BLISS exposed too much of the underlying architecture, and the compilers were neither portable [9, 10] nor freely available. Brian Reid commented [90, p. 106]:

> BLISS proved to be an extremely difficult language in which to get started on such a project \[\S\CTRIBE], since it has utterly no low-level support for any data types besides scalar words and stack-allocated vectors.

I began an implementation on the PDP-10 in September 1976, spending the first six months building a programming environment in which the rest of the development could take place. This programming environment included runtime and diagnostic sup-


---

Nelson H. F. Beebe

---

`TUGboat, Volume 26 (2005), No. 1 — Proceedings of the Practical \TeX 2005 Conference`
port for strings, lists, and heap-allocated vectors, as well as an operating-system interface intended to be portable across machines.

Inside DEC, later absorbed by Compaq and then by Hewlett-Packard, BLISS was ported to 32-bit and 64-bit Alpha in the early 1990s, to Intel IA-32 in 1995, and recently, to IA-64 [10], but has remained largely unknown and unused outside those corporate environments.

LISP would have been attractive and powerful, and in retrospect, would have made \TeX{} and \metafont far more extensible than they are, because any part of them could have been rewritten in LISP, and they would not have needed to have macro languages at all! Unfortunately, until the advent of Common LISP in 1984 [96, 97], and for some time after, the LISP world suffered from having about as many dialects as there were LISP programmers, making it impossible to select a language flavor that worked everywhere.

The only viable approach would have been to write a LISP compiler or interpreter, bringing one back to the original problem of picking a language to write \textit{that} in. The one point in favor of this approach is that LISP is syntactically the simplest of all programming languages, so workable interpreters could be done in a few hundred lines, instead of the 10K to 100K lines that were needed for languages like Pascal and Fortran. However, we have to remember that computer use cost a lot of money, and comparatively few people outside computer-science departments had the luxury of ignoring the substantial run-time costs of interpreted languages. A typesetting system is expected to receive heavy use, and efficiency and fast turnaround are essential.

PDP-10 assembly language had been used for many other programming projects, including the operating systems and the three assemblers themselves. However, Don had worked on several different machines since 1959, and he knew that all computers eventually get replaced, often by new ones with radically-different instruction sets, operating systems, and programming languages. Thus, this avenue was not attractive either, since he had to be able to use his typesetting program for all of his future writing.

There was only one viable choice left, and that was SAIL. That language was developed at Stanford, and that is probably one of the reasons why Don chose it over Simula 67, its Norwegian cousin, despite his own Norwegian heritage; both languages are descendents of Algol 60. Simula 67 did however strongly influence Bjarne Stroustrup’s design of C++ [98, Chapter 1]. Although SAIL had an offspring, MAINSAIL (Machine Independent SAIL), that might have been more attractive, that language was not born until 1979, two years after the sabbatical-year project. Figure 2 shows a small sample of SAIL, taken from the \metafont source file mntprp.sai. A detailed description of the language can be found in the first good book on computer graphics [86, Appendix IV], co-written by one of SAIL’s architects.

\begin{figure}[h]
\begin{verbatim}
internal saf string array fname[0:2]
    # file name, extension, and directory;

internal simp procedure scanfilename
    # sets up fname[0:2];
    begin integer j,c;
        fname[0].fname[1].fname[2].null;
        j_0;
        while curbuf and chartype[curbuf]=space
            do c_lop(curbuf);
                loop case c of begin
                    [pnt] j_1;
                    [lbrack] j_2;
                    [comma][wxy][rbrack][digit][letter];
                    else done
                        end;
                fname[j].fname[j]&lop(curbuf);
            end;

Figure 2: Filename scanning in SAIL, formatted as originally written by Donald Knuth, except for the movement of comments to separate lines. The square-bracketed names are symbolic integer constants declared earlier in the program.
\end{verbatim}
\end{figure}

The underscore operator in SAIL source-code assignments printed as a left arrow in the Stanford variant of ASCII, but PDP-10 sites elsewhere just saw it as a plain underscore. However, its use as the assignment operator meant that it could not be used as an extended letter to make compound names more readable, as is now common in many other programming languages.

The left arrow in the Stanford variant of ASCII was not the only unusual character. Table 2 shows graphics assigned to the normally glyphless control characters. The existence of seven Greek letters in the control-character region may explain why \TeX{}’s default text-font layout packs Greek letters into the first ten slots.

Besides being a high-level language with good control and data structures, and recursion, SAIL...
Table 2: The Stanford extended ASCII character set, with table positions in octal. This table from RFC 698 [84] disagrees in a few slots with a similar table in the first book about \TeX{} [59, p. 169]. CMU, MIT, and the University of Southern California also had their own incompatible modified versions of ASCII.

Although ASCII was first standardized in 1963, got lowercase letters in 1965, and reached its current form in 1967, the character set Babel has lasted far too long, with hundreds of variants of 7-bit and 8-bit sets still in use around the world. See Mackenzie’s book [81] for a comprehensive history up to 1980, and the Unicode Standard [102] for what the future may look like.

```
<table>
<thead>
<tr>
<th>ASCII</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>↓</td>
</tr>
<tr>
<td>002</td>
<td>α</td>
</tr>
<tr>
<td>003</td>
<td>β</td>
</tr>
<tr>
<td>004</td>
<td>∧</td>
</tr>
<tr>
<td>005</td>
<td>¬</td>
</tr>
<tr>
<td>006</td>
<td>δ</td>
</tr>
<tr>
<td>007</td>
<td>κ</td>
</tr>
<tr>
<td>010</td>
<td>λ</td>
</tr>
<tr>
<td>011</td>
<td>γ</td>
</tr>
<tr>
<td>012</td>
<td>δ</td>
</tr>
<tr>
<td>013</td>
<td>ς</td>
</tr>
<tr>
<td>014</td>
<td>±</td>
</tr>
<tr>
<td>015</td>
<td>⊕</td>
</tr>
<tr>
<td>016</td>
<td>∞</td>
</tr>
<tr>
<td>017</td>
<td>∇</td>
</tr>
<tr>
<td>020</td>
<td>⊂</td>
</tr>
<tr>
<td>021</td>
<td>⊃</td>
</tr>
<tr>
<td>022</td>
<td>∩</td>
</tr>
<tr>
<td>023</td>
<td>∪</td>
</tr>
<tr>
<td>024</td>
<td>∀</td>
</tr>
<tr>
<td>025</td>
<td>∃</td>
</tr>
<tr>
<td>026</td>
<td>⊗</td>
</tr>
<tr>
<td>027</td>
<td>↔</td>
</tr>
<tr>
<td>030</td>
<td>→</td>
</tr>
<tr>
<td>031</td>
<td>~</td>
</tr>
<tr>
<td>032</td>
<td>≠</td>
</tr>
<tr>
<td>033</td>
<td>≤</td>
</tr>
<tr>
<td>034</td>
<td>≥</td>
</tr>
<tr>
<td>035</td>
<td>≡</td>
</tr>
<tr>
<td>036</td>
<td>≅</td>
</tr>
<tr>
<td>037</td>
<td>χ</td>
</tr>
<tr>
<td>040–135</td>
<td>in standard ASCII</td>
</tr>
<tr>
<td>136</td>
<td>↑</td>
</tr>
<tr>
<td>137</td>
<td>←</td>
</tr>
<tr>
<td>140–174</td>
<td>in standard ASCII</td>
</tr>
<tr>
<td>175</td>
<td>°</td>
</tr>
<tr>
<td>176</td>
<td>}</td>
</tr>
<tr>
<td>177</td>
<td>^</td>
</tr>
</tbody>
</table>
```

had the advantage of having a good debugger. Symbolic debuggers are common today, sometimes even with fancy GUI front ends that some users like. In 1977, window systems had mostly not yet made it out of Xerox PARC, and the few interactive debuggers available generally worked at the level of assembly language. Figure 3 shows a small example of a session with the low-level Dynamic Debugging Tool/Technique, ddt, that otherwise would have been necessary for debugging most programming languages other than \SAIL (\texttt{ALGOL}, \texttt{COBOL}, and \texttt{FORTRAN}, and later, \texttt{PASCAL}, also had source-level debuggers).

\SAIL had a useful conditional compilation feature, allowing Don to write the keyword definitions shown in Figure 4, and inject a bit of humor into the code.

A scan of the SAIL source code for \texttt{METAFONT} shows several other instances of how the implementation language and host computer affected the \texttt{METAFONT} code:

- 19 buffers for disk files;
- no more than 150 input characters/line;
- initialization handled by a separate program module to save memory;
- bias of 4 added to case statement index to avoid illegal negative cases;
- character raster allocated dynamically to avoid 128K-word limit on core image;
- magic \texttt{TENEX}-dependent code to allocate buffers between the \texttt{METAFONT} code and the \SAIL disk buffers because, as Don wrote in a comment in the code, \textit{there is all this nifty core sitting up in the high seg ... that is just begging to be used}.

Another feature of the PDP-10 that strongly influenced the design of \TeX{} and \texttt{METAFONT} was the way the loader worked. On most other operating systems, the linker or loader reads object files, finds the required libraries, patches unresolved references, and writes an executable image to a disk file. The PDP-10 loader left the program image in memory, relegating the job of copying the memory image to disk to the save command. If the image was not required again, the user could simply start the program without saving it. If the program was

```
@type hello.pas
program hello(output);
begin
  writeln('Hello, world')
end.

@load hello
PASCAL: HELLO
LINK: Loading
@ddt
DDT
hello$b hello+62$b $g
$1B>>HELLO/ TDZA 0 $x
  0/ 0 0/ 0
<SKIP>
HELLO+2/ MOVEM %CCLSW $x
  0/ 0 %CCLSW/ 0
HELLO+3/ MOVE %CCLDN $x
  0/ 0 %CCLDN/ 0
HELLO+4/ JUMPN HELLO+11 $x
  0/ 0 HELLO+11
HELLO+5/ MOVEM 1,%RNNAM $p

OUTPUT : tty:
$2B>>HELLO+62/ JRST .MAIN. $x
Hello, world
```

Figure 3: Debugging a \texttt{PASCAL} program with ddt. The at signs are the default TOPS-20 command prompt. The dollar signs are the echo of ASCII \texttt{ESC APE} characters. Breakpoints ($b$) are set at the start of the program, and just before the call to the runtime-library file initialization. Execution starts with $g$, proceeds after a breakpoint with $p$, steps single instructions with $x$, and steps until the next breakpoint with $$x.
started, but then interrupted at a quiescent point, such as waiting for input, the memory image could be saved to disk.

Since some of the features of \TeX{} and METAFONT are implemented in their own programming languages, they each need to read that code on every execution. For \LaTeX{}, the startup code can amount to tens of thousands of lines. Thus, for small user input files, the startup actions may be significantly more costly than the work needed for the user files. Don therefore divided both programs into two parts: the first parts, called initex and inimf, read the startup code and write their internal tables to a special compact binary file called a format file. The second parts, called virtex and virmf, can then read those format files at high speed. If they are then interrupted when they are ready for user input, they can be saved to disk as programs that can later be run with all of this startup processing already done.\textsuperscript{[72, \S 1203], [70, \S 1331]}. While this sounds complex, in practice, it takes just six lines of user input, shown in Figure 5. This normally only needs to be done by a system manager when new versions of the startup files are installed. It is worth noting that installers of both PDP-10 emacs and modern GNU emacs do a very similar preparation of a dumped-memory image to reduce program-startup cost.

On most other architectures, the two-part split is preserved, but the virtex and virmf programs are then wrapped in scripts that act as the tex and mf programs. On UNIX systems, the script wrappers are not needed: instead, virtex, tex, and latex are filesystem links to the same file, and the name of the program is used internally to determine what format file needs to be automatically loaded. Modern systems are fast enough that the extra economization of preloading the format file into the executable program is relatively unimportant: the fastest systems can now typeset the \TeX{}book at nearly 1100 pages/sec, compared to several seconds per page when \TeX{} was first written. In any event, preloading is difficult to accomplish outside the PDP-10 world. It can be done portably, but much less flexibly, if the preloaded tables are written out as source-code data initializers, and then compiled into the program, as the GNU bc and dc calculators do for their library code.

\LaTeX{} and METAFONT distributions come with the devious \texttt{trip} and \texttt{trap} torture tests that Don devised to test whether the programs are behaving properly. One of the drawbacks of the two-part split is that these tests are run with initex and inimf respectively, rather than with the separately-compiled virtex and virmf, which are the programs that users run as tex and mf. I have encountered at least one system where the torture tests passed, yet virtex aborted at run time because of a compiler code-generation error. Fortunately, the error was eliminated when virtex was recompiled with a different optimization level.

Although \LaTeX{} and METAFONT were designed with great care and attention to detail, and programmed to give identical line-breaking and page-breaking decisions on all platforms, it would have

\begin{figure}
\begin{verbatim}
# changed to ^P^Q when debugging METAFONT;
define DEBUGONLY = ^Pcomment^Q

# used when an array is believed to require
# no bounds checks;
define saf = ^Psafe^Q

# when debugging, belief turns to disbelief;
DEBUGONLY redefine saf = ^P^Q

# and simplicity dies too;
DEBUGONLY redefine simp = ^P^Q
\end{verbatim}
\caption{SAIL conditional compilation for generating additional debugging support. The two control characters displayed as \texttt{\&} and \texttt{\_} at Stanford (octal values 020 and 021 in Table 2).}
\end{figure}

The design of \LaTeX{} and METAFONT: A retrospective

\begin{verbatim}
@initex lplain
*\dump
@virtex &lplain
*^C
@save latex
@rename lplain.fmt texformats:
\end{verbatim}

\begin{figure}
\caption{Creating a preloaded \LaTeX{} executable on TOPS-20.}
The initex stage reads lplain.tex and dumps the precompiled result to lplain.fmt. The leading ampersand in the virtex stage requests reading of the binary format file, instead of a normal \TeX{} text file. The keyboard interrupt suspends the process, and the next command saves latex.exe.

The final command moves the format file to its standard location where it can be found should it be needed again. On TOPS-20, it normally is not read again unless a user wishes to preload further customizations to create another executable program.

The procedure for METAFONT is essentially the same; only the filenames have to be changed.
been better if their user communities had collaborated on development of a much more extensive test suite, designed with the help of test-coverage analyzers to ensure that as much of the source code as possible is exercised. These compiler-based tools instrument software in such a way that program execution produces a data file that leads to a report of the number of times that each line of code is reached. This identifies the hot spots in the code, but it also reveals the unused, and therefore, untested, parts of the program.

When I did such an analysis of runs with the trip and trap tests, I was surprised to find that just under 49% of all lines of code were executed. I reported these results to the \TeX{} Live mailing list on 18 March 2004, in the hope of initiating a project to use the test-coverage feedback to devise additional tests that will exercise most of the other half of the code. It will never be possible to test all of it: there are more than 50 locations in the \TeX{} and METAFONT source code where there is a test for a supposedly-impossible situation, at which point section 95 of \TeX{} (section 90 in METAFONT) is invoked to issue a message prefixed with This can’t happen and terminate execution.

6 Switching programming languages

Donald Knuth initially expected that \TeX{} and METAFONT would be useful primarily for his own books and papers, but other people were soon clamoring for access, and many of them did not have a PDP-10 computer to run those programs on. The American Mathematical Society was interested in evaluating \TeX{} and METAFONT for its own extensive mathematical-publishing activities, but it could make an investment in switching from the proprietary commercial typesetting system that it was then using only if it could be satisfied with the quality, the longevity, and the portability of these new programs.

Researchers at Xerox PARC had translated the SAIL version of \TeX{} to MESA, but that language ran only on Xerox workstations, which, while full of great ideas, were too expensive ever to make any significant market penetration.

It was clear that keeping \TeX{} and METAFONT tied to SAIL and the PDP-10 would ultimately doom them to oblivion. It was also evident that some of the program-design decisions, and the early versions of the Computer Modern fonts, did not produce the high quality that their author demanded of himself.

A new implementation language, and new program designs, were needed, and in 1979–1980, when Don and Ignacio produced prototype code for the new design, there was really only one possibility: PASCAL. However, before you rise to this provocation, why not C instead, since it has become the lingua franca for writing portable software?

UNIX had reached the 16-bit DEC PDP-11 computers at the University of California at Berkeley in 1974. By 1977, researchers there had it running on the new 32-bit DEC VAX, but the C language in which much of UNIX is written was only rarely available outside that environment. Jay Lepreau’s pcc20 work was going on in the Computer Science Department at Utah in 1981–82, but it wasn’t until about 1983 that TOPS-20 users elsewhere began to get access to it. Our filesystem archives show my first major porting attempt of a C-language UNIX utility to TOPS-20 on 11 February 1983.

PASCAL, a descendant of ALGOL 60 [5], was designed by Niklaus Wirth at ETH in Zürich, Switzerland in 1968. His first attempt at writing a compiler for it in FORTRAN failed, but he then wrote a compiler for a subset of PASCAL in that subset, translated it by hand to assembly language, and was finally able to bootstrap the compiler by getting it to compile itself [106].

Urs Ammann later wrote a completely new compiler [2] in PASCAL for the PASCAL language on the 60-bit CDC 6600 at ETH, a machine class that I myself worked extensively and productively on for nearly four years. That compiler generated machine code directly, instead of producing assembly code, and ran faster, and produced faster code, than Wirth’s original bootstrap compiler. Ammann’s compiler was the parent of several others, including the one on the PDP-10.

PASCAL is a small language intended for teaching introductory computer-programming skills, and Wirth’s book with the great title Algorithms + Data Structures = Programs [107] is a classic that is still worthy of study. However, PASCAL is not a language that is suitable for larger projects. A fragment of the language is shown in Figure 6, and much more can be seen in the source code for \TeX{} [70] and METAFONT [72].

PASCAL’s flaws are well chronicled in a famous article by Brian Kernighan [40, 42]. That paper was written to record the pain that PASCAL caused in implementing a moderate-sized, but influential, programming project [44]. He wrote in his article:

PASCAL, at least in its standard form, is just plain not suitable for serious programming.

... This botch [confusion of size and type] is the biggest single problem in PASCAL. ...

I feel that it is a mistake to use PASCAL for
Figure 6: Filename scanning in PASCAL, after manual prettyprinting. The statements beginname and endname are calls to procedures without arguments. The magic constants 32, 37, and 59 would normally have been given symbolic names, but this code is output by the tangle preprocessor which already replaced those names by their numeric values. The lack of statements to exit loops and return from procedures forces programmers to resort to predeclared numeric labels in PASCAL.

...
is called a \texttt{WEB}, and a fragment of \TeX{}'s own \texttt{WEB} code is illustrated in Figure 7. The output of the two utilities is shown in Figures 8 and 9, and the typeset output for the programmer is given in Figure 10.

In order to keep a stable source-code base, the \texttt{WEB} files are \textit{never} edited directly when the code is ported to a new platform. Instead, tangle and \texttt{weave} accept simple \textit{change files} with input blocks
\begin{verbatim}
@x old code
@y new code
@z
\end{verbatim}

where the old-code sections must match their order in the \texttt{WEB} file. For \TeX{} and METAFONT, these change files are typically of the order of $5\%$ of the size of the \texttt{WEB} files, and the changes are almost exclusively in the system-dependent parts of those programs, and in the handling of command-line and startup files.

\begin{verbatim}
@ The \{scan_optional_equals\} routine looks for an optional `\'.=\' sign preceded by optional spaces; `\.(\{relax\})' is not ignored here.

@p procedure scan_optional_equals;
begi
@<Get the next non-blank non-call token@>;
if cur_tok<>other_token+.= then back_input;
end;
\end{verbatim}

\textbf{Figure 7}: Fragment of \texttt{tex.web} corresponding to section 405 of \TeX: The Program [70, p. 167]. The vertical bars are a \texttt{WEB} shorthand that requests indexing of the enclosed text. The prose description begins with the command @p, and the \texttt{PASCAL} code begins with the command @p. The text @<...> represents a block of code that is defined elsewhere.

Because \texttt{PASCAL} permits only one source-code file per program, \texttt{WEB} files are also monolithic. However, to reduce the size of the typeset program listing, change files normally include a statement \texttt{\let \maybe = \iffalse} near the beginning to disable DVI output of unmodified code sections. Having a single source file simplified building the programs on the PDP-10, which didn't have a \texttt{UNIX}-like make utility until I wrote one in 1988. Figure 11 shows how \texttt{index} was built on TOPS-20.

In the early 1980s, few users had terminals capable of on-screen display of typeset output, so one of the system-dependent changes that was made in the PDP-10 implementations of \TeX{} was the generation of a candidate command for printing the

\begin{verbatim}
\&{end};
\&{then}\5 \&{back_input};\2
\&{if}
\1\&{cur_tok}+\{other_token\}+\.=\$\1\&{then}\5 \&{back_input};\2
\&{end};\par \fi
\end{verbatim}

\textbf{Figure 8}: \texttt{PASCAL} code produced from the \texttt{WEB} fragment in Figure 7 by tangle. All superfluous spaces are eliminated on the assumption that humans never need to read the code, even though that may occasionally be necessary during development. Without postprocessing by a \texttt{PASCAL} prettyprinter, such as \texttt{pform}, it is nearly impossible for a human to make sense of the dense run-together \texttt{PASCAL} code from a large \texttt{WEB} file, or to set sensible debugger breakpoints.

To conform to the original definition of \texttt{PASCAL}, and adapt to limitations of various compilers, all identifiers are uppercased, stripped of underscores, and truncated to 12 characters, of which the first 7 must be unambiguous.

Notice that the remote code from the @<...> input fragment has been inserted, and that symbolic constants have been expanded to their numeric values. The braced comments indicate sectional cross references, and no other comments survive in the output \texttt{PASCAL} code.

\begin{verbatim}
\&{begin} \37\&{get the next non-blank non-call token\X406};\2\6
\&{if}
\$\&{cur_tok}+\{other_token\}+\.=\$\1\&{then}\5 \&{back_input};\2\6
\&{end};\par \fi
\end{verbatim}

\textbf{Figure 9}: \TeX{} typesetter input produced from the \texttt{WEB} fragment in Figure 7 by \texttt{weave}.

\begin{verbatim}
\&{begin} \37\&{get the next non-blank non-call token 406};\2\6
\&{if}
\$\&{cur_tok}+\{other_token\}+\.=\$\1\&{then}\5 \&{back_input};\2\6
\&{end};\par \fi
\end{verbatim}

\textbf{Figure 10}: Typeset output from \TeX{} for the \texttt{weave} fragment in Figure 9. Notice that the remote code block is referenced by name, with a trailing section number that indicates its location in the output listing. Not shown here is the mini-index that is typeset in a footnote, showing the locations elsewhere in the program of variables and procedures mentioned on this output page.
The design of \TeX{} and METAFONT: A retrospective

@tex hello.tex
This is \TeX{}, Tops-20 Version 2.991
(preloaded format=plain 5.1.14)
(PS:<BEEBE>HELLO.TEX.1 [1])
Output written on PS:<BEEBE>HELLO.DVI.1
(1 page, 212 bytes).
Transcript written on PS:<BEEBE>HELLO.LST.1.
@TeXspool: PS:<BEEBE>HELLO.DVI.1

Figure 12: A \TeX{} run on TOPS-20. The user typed only
the first command, and in interactive use, \TeX{} provided
the second command, leaving the cursor at the end of
the line, so the user could then type a carriage return
to accept the command, or a Ctl-U or Ctl-C interrupt
character to erase or cancel it.

This feature was implemented via a TOPS-20 system
call that allowed a program to simulate terminal input.
\TeX{} thereby saved humans some keystrokes, and users
could predefine the logical name \TeX{}spool with a suit-
able value to select their preferred DVI translator. This
shortcut is probably infeasible on most other operating
systems.

operate. The 16-bit computers were particularly
challenging because of their limited address space,
and it was a remarkable achievement when Lance
Carnes announced \TeX{} on the HP3000 in 1981
[11], followed not long after by his port to the
IBM PC with the wretched 64KB memory segments
of the Intel 8086 processor. He later founded
a company, Personal \TeX{}, Inc. About the same
time, David Fuchs completed an independent port
to the IBM PC, and that effort was briefly available
commercially. David Kellerman and Barry Smith left
Oregon Software, where they worked on \PASCAL
compilers, to found the company Kellerman & Smith
to support \TeX{} in the VAX VMS environment. Barry
later started Blue Sky Research to support \TeX{} on the
Apple MACINTOSH, and David founded Northlake
Software to continue support of \TeX{} on VMS.

7 Switching languages, again

Because of compiler problems, UNIX users suffered
a delay in getting \TeX{} and METAFONT. Pavel Curtis
and Howard Trickey first announced a port in 1983,
and lamented [14]:

Unhappily, the pc [\PASCAL] compiler has
more deficiencies than one might wish.

Their project at the University of California, Berke-
ley, took several months, and ultimately, they had
to make several changes and extensions to the UNIX
\PASCAL compiler.

In 1986–1987, Pat Monardo, also at Berkeley,
did the UNIX community a great service when he un-
took a translation, partly machine assisted, and

output. A typical run then looked like the sample
in Figure 12.

Because \PASCAL had mainly been used for small
programs, few compilers for that language were
prepared to handle programs as large and complex
as \TeX{} and METAFONT. Their \PASCAL source code
produced by \tangle to tangle amounts to about 20000 lines
each when prettyprinted. A dozen or so supporting
tools amount to another 20000 lines of code, the
largest of which is weave.

Ports of \TeX{} and METAFONT to new systems
frequently uncovered compiler bugs or resource
limits that had to be fixed before the programs could

Figure 11: Building and installing initex on TOPS-20.
A similar procedure handled virtex: only the filenames
change, and in both cases, the procedure was encap-
slated in a command file that allowed a one-line command
to do the entire job.

The last command shows a wonderful feature of TOPS-
20: deleted files could be undeleted at any time until they
were expunged from the filesystem.

Comments from 1986 in the command file noted that
on the fastest DEC PDP-10 model, tangle took 102 sec-
onds, and \PASCAL compilation, 80 seconds.

When this build was repeated using the KLH10 simu-
lator running on a 2.4GHz AMD64 processor, tangle took
only 5 seconds, and \PASCAL only 2.6 seconds.

For comparison with a modern \TeX{} build on GNU/
\LINUX, I used the same AMD64 system for a fresh build.
\PASCAL generation with tangle took 0.09 seconds, the \WEB-
to-C conversion (see Section 7) took 0.08 seconds, and
compilation of the 14 C-code files took 2.24 seconds. The
KLH10 simulator times are clearly outstanding.

The change file on the PDP-10 inserted special compi-
ler directives in a leading comment to select extended
addressing. The memory footprint of \TeX{} after typesetting
its own source code is 614 pages of 512 words each, or
just 1.4MB.

On GNU/\LINUX on AMD64 with the 2004 \TeX{} Live
release, \TeX{} needs 11MB of memory to typeset itself,
although of course its tables are much larger, as shown
in Table 1.

TUGboat, Volume 26 (2005), No. 1 — Proceedings of the Practical \TeX{} 2005 Conference 45
partly manual, of Ti\TeX{} from P\ASCAL{} to C, the result of which he called COMMON Ti\TeX{}. That work ultimately led to the WEB2C project to which many people have contributed, and today, virtually all UNIX installations, and indeed, the entire Ti\TeX{} Live distribution for UNIX, Apple MAC OS, and Microsoft WINDOWS, is based on the completely-automated translation of the master source files of all Ti\TeX{}ware and META\FONT{}ware from the WEB sources to P\ASCAL{} and then to C.

8 Ti\TeX{}'s progeny

The limitations that stem from the resources and technologies that were available when Ti\TeX{} was developed have since been addressed in various ways. As we showed in Table 1, some of the internal table sizes are relatively easy to expand, as long as the host platform has enough addressable memory.

Growing tables whose indexes are limited to a small number of bits requires deeper changes, and combined with the addition of a small number of new primitives, and several useful extensions, resulted in e-Ti\TeX{} [100]. Its change file is about a quarter the size of tex.web.

Ti\TeX{} has been extended beyond the limitations of eight-bit characters in significant projects for typesetting with the UNICODE character set: OMEGA (Ω) [87, 99], ALEPH (N) [7], and XeTi\TeX{} [45, 46]. Each is implemented with change files for the Ti\TeX{} or e-Ti\TeX{} WEB sources. For OMEGA, the change files are about as large as tex.web itself, reflecting modification of about half of Ti\TeX{}, and suggesting that a new baseline, or a complete rewrite, may be desirable.

With few exceptions other than GNU groff (a reimplemented of UNIX troff), Ti\TeX{}'s DVI file format is not widely known outside the Ti\TeX{} world. Indeed, commercial vendors usurped the DVI acronym to mean Digital Video Interactive and Digital Visual Interface. Today, electronic representation of typeset documents as page images in PDF format [1] is common. While this format is readily reachable from Ti\TeX{} with translation from DVI to POSTSCRIPT to PDF, or directly to PDF, there are some advantages to being able to access advanced features of PDF such as hypertext links and transparency from within Ti\TeX{} itself. Hàn Thế Thành’s pdfTi\TeX{} [28] is therefore an important extension of Ti\TeX{} that provides PDF output directly, and allows fine control of typography with new features like dynamic font scaling and margin kerning [27, 29]. The change file for pdfTi\TeX{} is about a third the size of tex.web.

It is worth noting that yet another program-

ming language has since been used to reimplement Ti\TeX{}: Karel Skoupy’s work with JAVA [25]. One of the goals of this project was to remove most of the interdependence of the internals of Ti\TeX{} to make it easier to produce Ti\TeX{}-like variants for experiments with new ideas in typography.

Another interesting project is Achim Blumensath’s ANT: A Typesetting System [8], where the recursive acronym means AN\TeX{}T is not Ti\TeX{}. The first version was done in the modern LISP dialect SCHEME, and the current version is in OCAML. Input is very similar to Ti\TeX{} markup, and output can be DVI, POSTSCRIPT, or PDF.

Hong Feng’s NeoTi\TeX{} is a recent development in Wuhan, China, of a typesetting system based on the algorithms of Ti\TeX{}, but completely rewritten in SCHEME, and outputting PDF. Perhaps this work will bring Ti\TeX{} back to its origins, allowing it to be reborn in a truly extensible language.

Although most users view Ti\TeX{} as a document compiler, Jonathan Fine has shown how, with small modifications, Ti\TeX{} can be turned into a daemon [17]: a permanently-running program that responds to service requests, providing typesetting-on-demand for other programs. At Apple [3], IBM [38], Microsoft [82], SIL [12], and elsewhere, rendering of UNICODE strings is being developed as a common library layer available to all software. These designers have recognized that typesetting is indeed a core service, and many programmers would prefer it to be standardized and made universally available on all computers.

9 META\FONT{}'s progeny

Unlike Ti\TeX{}, META\FONT{} has so far had only one significant offspring: META\POST{} (or META\POST{}), written by Don’s doctoral student John Hobby [36], to whom META\FONT{}: The Program is dedicated. META\POST{} is derived from META\FONT{}, and like that program, is written as a P\ASCAL{} WEB. META\POST{} normally produces pictures, although it can also generate data for outline font files, and it supports direct output in POSTSCRIPT. META\POST{} is described in its manuals [32–35] and parts of two books [22, Chapter 3], [37, Chapter 13].

Although META\FONT{}, META\POST{}, and POSTSCRIPT offer only a two-dimensional drawing model, the 3DLDLD program developed by Laurence Finston [18] and the FEATPOST program written by Luís Nobre Gonçalves [19] provide three-dimensional drawing front ends that use META\POST{} at the back end. Denis Roegel’s 3d.mp package [91] offers a similar extension using the META\POST{} programming language.
The recent Asymptote program [26] credits inspiration from \textsc{metapost}, but is a completely independent package for creating high-quality technical drawings, with an input language similar to that of \textsc{metapost}.

\section{Wrapping up}

In this article, I have described how architecture, operating systems, programming languages, and resource limits influenced the design of \TeX and \textsc{metafont}, and then briefly summarized what has been done in their descendants to expand their capabilities. This analysis is in no way intended to be critical, but instead, to offer a historical retrospective that is, I believe, helpful to think about for other widely-used software packages as well.

\tex and \textsc{metafont}, and the literate programming system in which they are written, are truly remarkable projects in software engineering. Their flexibility, power, reliability, and stability, and their unfettered availability, have allowed them to be widely used and relied upon in academia, industry, and government. Donald Knuth expects to use them for the rest of his career, and so do many others, including this author. Don's willingness to expose his programs to public scrutiny by publishing them as books [70, 72, 74], to further admit to errors in them [61, 62] in order to learn how we might become better programmers, and then to pay monetary rewards (doubled annually for several years) for the report of each new bug, are traits too seldom found in others.

\section{Bibliography}


[26] Andy Hammerlindl, John Bowman, and Tom Prince. ASYMPTOTE: a script-based vector graphics language. Faculty of Science, University of Alberta, Edmonton, AB, Canada, 2004. URL http://asympote.sourceforge.net/. ASYMPTOTE is a powerful script-based vector graphics language for technical drawing, inspired by METAPOST but with an improved C++-like syntax. ASYMPTOTE provides for figures the same high-quality level of typesetting that \latex does for scientific text.


This obituary nicely describes the very many accomplishments of this industry pioneer.


A LaTeX fledgling struggles to take flight

Peter L. Flom
National Development and Research Institutes, Inc.
71 West 23rd St., 8th floor
New York, NY 10010

A little about this article
I work as a statistical consultant and data analyst at a nonprofit research company. I also work as an independent statistical consultant, mostly to graduate students in the social and behavioral sciences. I’ve done almost no computer programming. (I did have one course in computer programming, but it was so long ago that we used punch cards and waited a day or more for our programs to run on the mainframe that took up most of the basement; I also write some very simple programs in R.)

When I read the first issue of *The PracTEX Journal*, I was thrilled. Finally, someone was writing a journal for beginners. So, I wrote a very enthusiastic ‘Thank you’ to the editor (Lance Carnes), and he wrote back, thanking me for the feedback, and asking me to write an article. I said OK. And here it is.

I’m writing with two groups in mind: Beginners, and people who write for beginners. I’d like to offer both groups some perspective from someone who is just a little way along the path. I’d like to let the true beginners know that it is possible to learn LaTeX; after only a few months of intermittent use, I can do a lot — I have written entire articles in LaTeX, some of them with quite complicated organizational structure and with fairly intimidating formulas; I’ve also started doing some presentations in LaTeX, using the Beamer package. If I can do it, you can too. I’d like to give the teachers the perspective of a recent beginner, so that their efforts can have maximum reward; when I consider that so many people contribute to LaTeX, often without any monetary reward, I imagine that those people would like to have their efforts help as many people as possible to use LaTeX easily and well. This article is in three sections:

1. Introduction
2. Some suggestions for teaching beginners
3. Some hints for beginners

Work on this project was supported by NIDA grant P30 DA11041; I’d like to thank the editor and the reviewers for their helpful comments and encouragement.

Editor’s note: This article was originally published in *The PracTEX Journal*, issue 2005-2, http://tug.org/pracjourn.

I hope, however, that both teachers and learners will read all the sections — the division of material is not rigid.

How I started using LaTeX
Long ago, I used Nota Bene. This was a very nice word processing program, designed for use by scholars. But no one I knew used it, so . . . I then became a dissatisfied user of Microsoft Word for years. But it came with my computer, everyone else used it, journal editors liked it, so, I used it. Then, at the recommendation of a friend and colleague, I started using WinEdt to write R files (R is a language and environment for statistical computing and graphics). It’s great for that purpose, but I noticed that it kept mentioning LaTeX. I looked into it a little, but it looked really hard, so I didn’t do much.

Then, I saw on the R-help list that someone was writing a book on R for beginners. I asked if he wanted some help from a beginner. He said he did; but the files were in LaTeX. He expressed amazement that I didn’t use it. But it looked really hard, so I didn’t do much.

Then I wrote a grant proposal that included a lot of formulas. A consultant on the grant did not have Word on his machine. He recommended LaTeX; but my co-investigator wanted Word files. So, I started looking more into LaTeX, and into programs to convert Word into LaTeX and vice versa. The deadline was looming, so I wrote that grant in Word (using Math Edit), and wrote files out as rtf files, which my consultant could read. Still, some formulas didn’t print right; or printed differently on different computers; it was a mess. So, I resolved to learn LaTeX. I’ve been using it more and more over the last 6 months or so, and now really prefer it to Word, for virtually everything. Maybe after reading another issue of this journal, I’ll prefer it for absolutely everything.

Some suggestions for teaching beginners

Ease of use
LaTeX looks hard. When I first saw a .tex file, I wondered how anyone could ever learn to write such
stuff. There are reasons for this: \LaTeX{} was (naturally) written and extended by computer scientists (Donald Knuth for \TeX{}, Leslie Lamport for \LaTeX{}, and many others), and that’s probably why it looks like a programming language.\footnote{Reviewers pointed out that most all document markup languages developed in the pre-GUI (graphical user interface) era looked like this.} When you are really expert at something, it’s hard to remember what it was like to not be expert; when you are really talented at something, it’s hard to empathize with the less talented — this is not to criticize the people who write for beginners, it’s just the way people are.

Well, I am neither experienced nor talented at programming, so I can empathize; even moderately complex \LaTeX{} files look indecipherable to true beginners (at least, they did to me). Part of this is due to how people are first exposed to \LaTeX{}. The first \texttt{.tex} file I saw was one which was going to be a book on a statistical programming language. I think that many people who start using \LaTeX{} do so because of the limitations of Word or Word Perfect, or some other program. Thus, the first things we want to write are complicated files. Also, for the people who write documentation, it’s easy to get into tricky stuff quickly, and this makes sense — there’s not much point in having pages and pages of very simple documents.

One way of making the learning curve a little less steep is to provide annotated programs. Another might be to provide more exercises and treat an introductory book more as a text.

So, if you’re writing for true beginners, emphasize ease of use. And, as \LaTeX{} becomes used by more people who are not and never were programmers, try to remember that we don’t think the way you think. If you’re a programmer who doesn’t like statistics, maybe thinking about how you would like to learn statistics would help in how people like me like to learn things like \LaTeX{}.

Distributions

Everything I see on \LaTeX{} mentions several (or more than several) different distributions. This just confused the heck out of me. Is there a difference? (I still don’t know.) Is one better than the other? (I still don’t know.) Some are free, some are commercial — what advantages do the commercial programs have? (They must have \texttt{some} or the companies would go out of business.) I’ve heard about LyX, which is a WYSIWYG version of \LaTeX{} — this seems nice, but what are the drawbacks? I wound up using \TeX{} Live, more or less by chance. Now I use pro\TeX{}t, because that’s what I got sent as a member of the \TeX{} Users Group (TUG). It would be good if some documentation could list the various distributions and what their strengths are, or state that there are no real differences.

Writing in \LaTeX{} is not like writing in Word

In Word (and probably in other word processors) when you don’t get what you want, it’s often because the program is illogical. It does some things automatically, some (most?) of which make no sense. In \LaTeX{}, though, when you don’t get what you want it’s often because you messed up. When I started writing things that were a little complex, I often got errors. This still happens. At first, this really annoyed me. It almost made me stop using \LaTeX{}.

Then I realized I should look on this more like a programming problem: Debugging is often necessary, and this doesn’t mean you’re stupid. I got this from the minimal programming I’ve done in R, but others who have never done any programming at all may not get this attitude, and I didn’t see it in any of what I’ve read. Programmers may be so used to this way of thinking that they don’t think to mention it.

Adding packages

I find this very confusing.\footnote{According to one reviewer, this may not be as difficult as I think it is — there are, apparently, tools for doing this that I am unaware of; I am just writing about what I know.} I’ve read various help files on how to do this; I’m sure they’re all correct, I know they’re all written by experts. It seems to me, as a nonprogrammer, that they contradict each other. I know they really don’t, because then they wouldn’t all work. So, it must be that I am even more confused than I thought, which is saying something. I don’t fully understand \textit{why} this has to be so hard (as I said, I am no programmer).

The other free software I use a lot is R, which also runs on lots of platforms, and also has lots of additional packages written by lots of different people, but there, when you add a package, it does all the background work for you; you just find the package you want, click on it, and you’re done. If it can’t be made easy, then I would strongly urge recommending that beginners install everything—all the available packages—at once. Disk space is cheap, writing the files takes a while, but it only needs to be done once. That’s what I wound up doing (by uninstalling all the files, and then reinstalling everything I could get all at once) and this worked perfectly.

To a large extent, these problems have been solved by pro\TeX{}t, which automates a lot of this. But, as far as I know, it is only for Windows, and
thus \LaTeX{} users using other systems may still have the type of question outlined above.

**Annotated programs**

All the books and other material on learning \LaTeX{} include numerous examples of \LaTeX{} files, which is good. One of the best ways of learning is by example. But one way to make these examples even more useful would be to include extensive annotations, either in the margins, in footnotes, or in text immediately below the program. What I have in mind is something like the way many editions of Shakespeare have notes explaining terms and references that are unfamiliar. The first few times a command is used, it would be useful to include a note. Kopka and Daly \cite{Kopka:1999} do a nice job of this in their “Sample \LaTeX{} file” on pages 16–19; I’d like to see more examples like this.

**Debugging and error messages**

Whenever I do anything complicated in \LaTeX{} (and sometimes when I do something simple) I get errors. The messages accompanying these are sometimes helpful, but often rather obscure, at least to non-programmers such as myself. It would be great to have a source that explains some of these error messages in ordinary English. It would also be great to have some reference on debugging.\footnote{I have since found that Kopka and Daly \cite{Kopka:1999} do include a list of some error messages in an appendix.}

**Some hints for beginners**

\LaTeX{} has to be **learned**

Word is designed not to be learned. It’s supposed to function right out of the box (whether it does or not is another matter): if you are used to Word, then you may think that you should be able to use \LaTeX{} right out of the box. Well, maybe some people can. I couldn’t. On the other hand, as you learn \LaTeX{}, you get more and more control over how your document looks.

**Some resources**

There are a lot of free resources available for \LaTeX{} (see the CTAN website). A lot of these are wonderful, and some are intended for beginners. I know some people find these resources to be enough for them to use \LaTeX{} very well. Personally, I like books. I keep three close at hand: Math into \LaTeX{} \cite{Gratzer:1999} is on my desk, and Guide to \LaTeX{} \cite{Kopka:1999}, and The \LaTeX{} Companion \cite{TUG:2000} are on my bookshelf. I like books (as opposed to web-based material) in general because:

1. they have extensive tables of contents and indexes;
2. they are already bound and thus easy to flip through;
3. I am just old-fashioned enough to like being able to page through a book, and keep it open on my desk while I work on something complex.

I like the three books mentioned above for different things. The \LaTeX{} Companion \cite{TUG:2000} is a great book, but not for beginners. It’s intimidating. It’s too big. It assumes knowledge. I think it should be the 3rd or 4th book a \LaTeX{} user buys; it’s a great reference, but it still kind of intimidates me.

Kopka and Daly’s Guide to \LaTeX{} \cite{Kopka:1999} is the best introduction to \LaTeX{} that I’ve seen. The book I use most now is George Grätzer’s Math into \LaTeX{} \cite{Gratzer:1999} (it’s open on my desk as I write this, I just looked up how to type the author’s accented name). I use this all the time, partly because one of the main reasons I started using \LaTeX{} was to typeset some complex mathematical formulas. All three of these books are very well organized and comprehensible, given their depth. Your taste in particular books may vary. Try out a few. Even if you buy a bunch of books before finding one or two you really like, it’s not that much money (after all, the software is free).

Another resource I find very helpful is the general mailing list for \TeX{} users, texhax@tug.org; for more information, see http://tug.org/mailman/listinfo/texhax.

Figure out what you need to know, and when you need to know it

\LaTeX{} is huge. It does all kinds of things, plus a lot that I am sure I am unaware of. What you need from it depends on what kind of work you do. For instance, I need to do a lot with tables, equations, bibliographies and imported graphics; I had to learn these first. But I don’t have as much need to make my own drawings—I’ll wait. Learning about some different fonts would be fun, but not urgent (for me—this may be very urgent for you). I will probably never learn to typeset Sanskrit or musical notation. But just figuring out what is available can be a challenge.

One thing to do, after you can write basic documents, is to browse through various sources, including books and the CTAN website; Jim Hefferon wrote a good introduction to the website in the first issue of The PracTEx Journal \cite{Hefferon:1999}. Try to follow the discussions on the mailing list. TPJ is very helpful; and then there’s TUGboat, which also contains more advanced material (sometimes, I don’t even understand the titles!).
Run files often
Run your file through \LaTeX{} a lot. Each time you do something even a little interesting, where you have any doubt at all about whether what you are doing will work correctly, typeset the file. If you’ve only made one or a few changes since you last ran the file, then it will be easier to find your error. In the editor I use (WinEdt) you can also typeset a small part of your document (hit ctrl+shift+c). This saves a lot of time.

On a related note, make backups often, and give them names you will understand and remember later. In particular, if you’ve gotten something complicated to work reasonably well, but still want to tweak it a little, save the file that works before you forget how you got it to work. (For me, this happens most often with complex, multiline equations and with tables that have complicated structures.)

Look at examples
All the books I listed have lots of examples. Try to figure out how they work and how they could be changed. Fool around; see what happens.

Make a default preamble
As you learn more \LaTeX{}, you will (probably) find that there are certain packages that you always want loaded. It’s hard (at least for me) to remember which ones I want, so I made a default file:

\begin{verbatim}
\documentclass{article}
\usepackage{graphicx}
\usepackage{amsmath, amsxtra, latexsym, amsthm}
\usepackage{escale, mathrsfs}
\usepackage{caption2, float, chapterbib, natbib}
\usepackage[section]{placeins}
\usepackage{fancyhdr}
\usepackage{geometry}
\usepackage[symbol, perpage]{footmisc}

\theoremstyle{plain}
\newtheorem{theorem}{Theorem}
\theoremstyle{definition}
\newtheorem{definition}{Definition}

\begin{document}
\title{Put title here}
\author{Peter L. Flom}
\maketitle
Sample text
\bibliographystyle{amsplain}
\bibliography{file name}
\end{document}
\end{verbatim}

\footnote{One of the reviewers commented that it would be better to make a .sty file; I, however, do not know how to do this.}

Summary
As I get more and more used to \LaTeX{}, I find it more and more useful. I am gradually using it for more and more documents. For me, the best things about using \LaTeX{}, as opposed to Word, are

1. \LaTeX{} directs my attention to things that need attention. It takes care of section formatting, typography, and so on; but it forces my attention to things like complicated mathematical formulas and complex tables.

2. The ability to typeset complex mathematical equations and know they will appear correctly on other people’s computers and on printout.

3. The naturalness of section formatting (that is, with \texttt{section} and related commands)

4. The ease of cross-referencing to different sections of a document (using \texttt{\label} and \texttt{\ref}).

5. The helpfulness of the \LaTeX{} community in finding solutions.

The biggest barriers to using \LaTeX{} are

1. Working with co-authors and editors who insist on Word files.

2. Formatting complex tables.

3. Learning to use my editor (WinEdt) more efficiently.

4. Remembering that getting an error message is not the computer telling me that I am stupid (careless, ignorant, forgetful . . . but not stupid).

I look forward to learning more, and to becoming more expert, and to finding ways to spread my \LaTeX{} wings. Certainly writing this article helped me do so, I hope reading it helped you, as well.

References


Abstract

Have you ever been stuck using \LaTeX? What does this really mean to you? “Stuck” may be anywhere from solving some esoteric error message while \LaTeX-ing to trying to find a solution to a specific, not so obvious, formatting issue. There is a huge \TeX community with a plethora of information where many problems have been solved by a highly knowledgeable group of volunteers. This article will attempt to lead you in the right direction to make the most out of the resources available during your \LaTeX adventure. I will attempt to explain common errors and provide solutions with \LaTeX and variations such as pdf\LaTeX.

Introduction

Back in September 1992, I was invited to present a paper about supporting \TeX and \LaTeX at Euro-\TeX'92 in Prague, Czechoslovakia. At that time, I had only been working at the University of Delaware for five years. Now thirteen years later, I have been supporting the \TeX and \LaTeX community for a total of eighteen years. I use the term community, because I have helped many \TeX and \LaTeX users around the world, not just at the University of Delaware. In the article, The Key to Successful Support: Knowing Your \TeX and \LaTeX Users, I wrote...

...Shortly after getting involved, I realized that this was not going to be a short-term project, but one that would last forever. What I mean by this is that from week to week, I would learn a new macro or style file, new previewer, new printer driver, new utility, new user and kept wondering how I was possibly going to stay above water and good support in such a changing environment.

These statements have proven to be so true. The foundation I laid eighteen years ago has paid off. \TeX and \LaTeX users at the University of Delaware have transitioned and transformed into (mostly) \LaTeX \epsilon users over the years, as have I also, as the primary support person on campus. The training once offered several times every semester has dwindled to one customized class per year or as new technology emerges. The basic documentation provided over the years has been updated with samples and pointers to books and resources online. The two guidelines:

- don’t reinvent the wheel, and
- email or call before getting too frustrated

have been my primary philosophy and strategy enabling users to connect with the best tools making it easier to master the art of \LaTeX problem solving and truly appreciate \LaTeX’s beauty as a powerful tool. Of course, like all art, it is subject to interpretation, so each user will have a different level of tolerance and appreciation of their final product or artwork.

Oiling the squeaky wheel

For me, training and support relies heavily on others to solve common problems. I have built a support network within the University of Delaware community encouraging users to share their experiences and examples. It has been the only way I have remained sane all these years supporting \TeX and \LaTeX. In the article \LaTeX/\TeX User: A Typist, or Typesetter?, I mentioned several important points to remember when using macros created by others

1. Having the macros does not mean that the user does not have to pay attention to the original specifications or guidelines. It is important that the user check the document for correctness. Macros are developed with the intention of being correct, but errors do happen.

2. Users need to be reminded that the macros have been defined to meet certain specifications, and as a result the macros should not be changed. I hear complaints such as, “I don’t like the way the document looks.” The point is that it does not matter how they think it should look, and
altering the macros means the document no longer conforms to the specifications.

3. There needs to be good documentation on how to use the macros. References on which macros fulfill which specifications are important.

4. Examples should be provided whenever possible. Example documents of the input and output are easy ways of showing the organization of the document, how to use the macros, and what they will produce.

Resources

Interestingly enough, while technology for accessing network resources has changed over the years, such as your favorite flavor of search engines, most of our past resources are still available and relevant today in their new and improved forums such as the TeX user groups, CTAN, (L)TeX newsgroups and FAQs. I must admit that www.tug.org and Google have become my best friends for connecting users with available network resources.

In addition, there have been many new books over the years and many have been updated to make them even more useful. Although more and more local versions of “Getting Started with (L)TeX” or “How do I do XXX in (L)TeX” are showing up all over the web, I believe the best working environment for users requires a set of TeX and/or (L)TeX books on hand.

A collection or complete system of tools is a must when using (L)TeX. It will make your (L)TeX experience so much easier and more enjoyable. Many users find this task overwhelming and confusing, because there are multiple choices just to get started. Some are free and some are not; what should you use? The answer to this question depends on your computer operating system and your comfort level with computers in general. Again, www.tug.org provides a wealth of information about both free implementations and commercial/shareware TeX systems, with links to the www.ams.org web page about commercial TeX-related software.

Common gotchas

Using a complex software package like (L)TeX gets easier with experience and time. There is a steep learning curve, especially if you are interested in trying to solve errors or debug your (L)TeX documents. If you always take an example or template and don’t deviate from it, then you are good to go.

While this is a very important part of learning (L)TeX, it won’t necessarily help you if you need to make any changes. The art of (L)TeX problem solving is understanding where to begin when you run into an error with (L)TeX. Below are some common gotchas that might getcha during your (L)TeX adventures.

- Preamble errors
- Missing or incorrect placement of }
- Blank lines or other spacing issues in math mode
- Forgetting about special characters, like $, %, & and quotation marks
- Protecting in moving arguments
- Misspelled environment or macro names
- Incorrect use of options or improper structure for an environment or macro
- Incorrect reference for numbering
- Mismatching braces, environments, “whatever”
- Changing size and style in text and math
- Figures and Tables
- Graphics

The largest numbers of complaints I receive are about the “meaningless error messages” in (L)TeX. However, it is probably better to describe the error messages as meaningful to experts. Of course this doesn’t help a beginner and to most this is by far what makes debugging (L)TeX so difficult. However, the art of debugging (L)TeX is the ability to divide and conquer around the error to see if you can produce a more meaningful error message or not get an error at all.

In my experience, most errors occur as a result of a missing or misplaced end environment or }. In addition, most users have a working document and then make changes which introduce an error. My suggestions are to always keep one revision behind or eliminate parts of the changes made and try to see if you can get back to a working document. Once you have a working document again, you can gradually add back changes so each small change will allow you to determine what may have caused the error in the first place.

Conclusion

Every user has a different learning style, but every user must be willing to learn (L)TeX, if they plan to succeed and feel comfortable with it. Be on the lookout for updates for all your resources and keep up with the technology. Don’t get intimidated and don’t exceed your tolerance threshold of frustration before getting help. Doing so will allow you to transition and transform into a (L)TeX problem solving art lover.
References


Strategies for including graphics in \LaTeX{} documents

Klaus Höppner  
Nieder-Ramstädt Str. 47  
64283 Darmstadt  
Germany  
klaus.hoeppner@gmx.de

Abstract

This talk presents strategies for including graphics into \LaTeX{} documents. It shows the usage of the standard graphics packages of \LaTeX{} as well as an introduction to different graphics formats. Some external tools for converting graphics formats are discussed.

Overview of graphics formats

In general, there exist two kinds of graphics formats: vector and bitmap graphics. For bitmaps, there exist different flavors: no compression (which can make your files truly huge, dependent on resolution and color depth, so I won’t cover them from here on), compression methods which completely preserve the image quality while reducing the data size, and “lossy” compression methods which cause a consequent reduction in image quality.

So let’s go more into detail:

Vector graphics are set up by drawing or filling geometrical objects such as lines, Bézier curves, polygons, circles and so on. The properties of these objects are stored mathematically. Vector graphics are in general device independent. It is easy to scale or rotate them without loss of quality, since the job of rasterizing them into actual pixels is done by the printer or printer driver.

Bitmaps without lossy compression store the image information as pixels, each pixel of a given color. In principle, the quality of a bitmap becomes better with increased resolution and color depth (e.g., GIF files use a color depth of 8 bits, leading to 256 different indexed colors while a bitmap with 24 bit color depth can have about 16 million colors). Scaling and rotating bitmap images will yield a loss of quality, and printing bitmaps to a device with a different resolution can produce bad results. Fig. 1 shows the difference between a scaled image as vector and bitmap graphics.

Figure 1: Zoomed view into a sample image as vector graphics (left) and bitmap (right).

Bitmaps with lossy compression use the fact that the human eye is fairly good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. For this reason, components in the high frequency region can be reduced, leading to smaller file sizes. This works well for photographs that usually contain smooth transitions in color, but for graphics with a sharp border, artifacts can occur, as shown in fig. 2. The most prominent graphics format using lossy compression is JPEG.

Figure 2: A low quality JPEG image showing some artifacts at the transition between black and white.

Graphics formats in practice

There exist very many graphics formats, so I will concentrate on a few of those most often used:

EPS is the encapsulated PostScript format. It is mostly used for vector graphics but can also contain bitmaps.
PNG is the portable network graphics format. It was introduced due to the problem that Unisys claimed a patent for the compression algorithm used in GIF format. For this reason, it is often used nowadays on web pages. PNG is a bitmap format that supports compression both with and without loss of image quality.

JPEG is a bitmap format with lossy compression and is often used for photographs (e.g. most digital cameras produce JPEG files).

TIFF is a bitmap format sometimes used for high quality pictures—in part because it supports the CMYK color space important especially for commercial printing.

Now the question is: What format shall I use for what purpose? Though there is no one true answer to this question, my advice is as follows:

1. For drawings (e.g. technical drawings or data plots) use vector graphics. It gives you maximum freedom to manipulate the image when including it into a document where you often need to scale the image to fit into your layout. Additionally, it is independent of the output device, and thus you can zoom into the image in your document viewer without seeing single pixels.

   Drawing tools offered by \TeX{} distributions—notably PSTricks and \METAPost—can usually produce EPS output natively. Most vector drawing programs like \Xfig{} and Corel Draw also offer export functionality for producing EPS output (though sometimes buggy).

2. If you are stuck with bitmaps, use PNG for images with sharp color transitions, such as black and white boundaries.

3. For photographs, you can use JPEG in most cases, since the quality loss by compression is normally imperceptible when printed. On most devices, a resolution of 100 to 200 dpi will be sufficient (remember that screen resolution is normally about 75 to 100 dpi, and color printers claim to have high resolutions but dither color prints, so you will hardly notice the difference compared to JPEGs with higher resolution).

The \TeX{} graphics package

Since the introduction of \TeX{}2\varepsilon, the graphics bundle is part of the standard package set accompanying the \TeX{} base distribution [1]. It consists of two style files, graphics.sty and graphicx.sty. While graphics.sty requires the use of \scalebox{} and \rotatebox{} for scaling or rotating graphics, the extended style graphicx.sty supports scaling and rotating using the \keyval{} package, which provides a convenient interface for specifying parameters. In general, there is no reason not to always use graphicx.sty.

So the first step is to load the graphicx style file after the \documentclass statement:

\begin{verbatim}
\usepackage[graphicx]
\end{verbatim}

In fact, the \TeX{} compiler doesn’t know anything about graphics, and including them is done by the DVI driver. So the graphicx package has to do two things:

1. find the bounding box of the image (this can be troublesome when you have e.g. an EPS file created by an application that wrote a wrong BoundingBox comment—in this case, it can be helpful to put the \includegraphics{} command into an \fbox{} to find out what graphicx{} thinks about the bounding box);

2. produce the appropriate \special{} for the output driver; thus, the usage of the graphics bundle is driver dependent.

Nowadays, there are two main workflows for producing documents: using latex to produce a DVI file and then dvips for converting it to PostScript, and using pdflatex to produce a PDF file. Most modern \TeX{} systems are configured to automatically check whether you are using latex or pdflatex{} and producing dvips \specials{} in the first case and the appropriate pdfimage commands in the second case. So if you are using one of the above workflows, you shouldn’t need to specify your output backend explicitly. If you are using another backend you have to specify it as an option, e.g.

\begin{verbatim}
\usepackage[dvipsone]{graphicx}
\end{verbatim}

(for the Y&Y dvipsone{} driver), but be aware that other backends often don’t support scaling or rotating. For example, DVI previewers like xdvi or windvi{} try to interpret the dvips \specials, but rotations may not be displayed properly in DVI preview.

After the package is loaded, to include an image simply use:

\begin{verbatim}
\includegraphics{sample}
\end{verbatim}

Please notice that no extension for the file was given. The explanation why will follow later. In the case of using \includegraphics{} without options the image is included at its natural size, as shown above. When using the graphicx style, you can scale your image by a factor:
Strategies for including graphics in \LaTeX
documents

\begin{itemize}
  \item \texttt{\includegraphics*[scale=0.5]{sample}}
  \item \texttt{\includegraphics*[scale=1.2]{sample}}
\end{itemize}

Another option supports rotating an image:
\begin{itemize}
  \item \texttt{\includegraphics*[angle=30]{sample}}
  \item \texttt{\includegraphics*[angle=-10]{sample}}
\end{itemize}

Positive numbers lead to counterclockwise rotation, negative numbers to clockwise rotation. The origin for the rotation is the lower left corner of the image, so in the clockwise rotation above the result has not only a height but also a depth below the baseline (as shown by the rules).

Images can not only be scaled by a given factor, you can specify a height and/or width for the resulting image instead:
\begin{itemize}
  \item \texttt{\includegraphics*[width=2cm]{sample}}
  \item \texttt{\includegraphics*[height=1.5cm]{sample}}
\end{itemize}

\texttt{height} gives the height above the baseline. If your image has a depth, you can use \texttt{totalheight} instead, i.e. the sum of height and depth will be scaled to the given length.
\begin{itemize}
  \item \texttt{\includegraphics*[angle=-30,height=1cm]{sample}}
  \item \texttt{\includegraphics*[angle=-30, totalheight=1cm]{sample}}
\end{itemize}

You can specify both \texttt{width} and \texttt{height}. In this case your image may be scaled differently in horizontal and vertical direction, unless you use the \texttt{keepaspectratio} option:
\begin{itemize}
  \item \texttt{\includegraphics*[width=1.5cm,height=1.5cm]{sample}}
  \item \texttt{\includegraphics*[width=1.5cm,height=1.5cm, keepaspectratio]{sample}}
\end{itemize}

Please notice that usage of \texttt{angle} and \texttt{width} or \texttt{height} is sensitive to the order in which the options are given. Specifying the angle first means that your image is rotated first and then the rotated image is scaled to the desired width or height, while specifying a width or height first will first scale the natural image and rotate it afterwards.

\begin{table}
\begin{tabular}{|c|c|c|}
\hline
Source & Target & Tool \\
\hline
\texttt{latex+dvips} & \texttt{EPS} & \texttt{directly supported} \\
\texttt{latex+dvips} & \texttt{PNG} & \texttt{EPS} \\
\texttt{latex+dvips} & \texttt{JPEG} & \texttt{EPS} \\
\texttt{latex+dvips} & \texttt{TIFF} & \texttt{EPS} \\
\texttt{pdflatex} & \texttt{PDF} & \texttt{directly supported} \\
\texttt{pdflatex} & \texttt{EPS} & \texttt{PDF} \texttt{epstopdf} \\
\texttt{pdflatex} & \texttt{PNG} & \texttt{directly supported} \\
\texttt{pdflatex} & \texttt{JPEG} & \texttt{directly supported} \\
\texttt{pdflatex} & \texttt{TIFF} & \texttt{PNG} \texttt{ImageMagick/netpbm} \\
\texttt{pdflatex} & \texttt{TIFF} & \texttt{PDF} \texttt{tif2eps+epstopdf} \\
\hline
\end{tabular}
\end{table}

Table 1: Conversion of graphics formats supported by \texttt{latex+dvips} and \texttt{pdflatex}.

Supported graphics formats

To make things a bit more complicated, \texttt{latex} with \texttt{dvips} and \texttt{pdflatex} support different graphics formats:
\begin{itemize}
  \item \texttt{latex+dvips}: EPS
  \item \texttt{pdflatex}: PDF, PNG, JPEG, MPS
\end{itemize}

Table 1 shows ways to convert the standard graphics formats to supported formats. In particular, converting EPS graphics used with \texttt{latex+dvips} to PDF for \texttt{pdflatex} workflow is quite easy; just run the \texttt{epstopdf} Perl script, which uses Ghostscript to convert EPS to PDF.

This also explains why it is generally best to give the file names in \texttt{\includegraphics} commands without extensions. In this case the \texttt{graphics} package looks for a supported graphics format automatically. So if you have an image both as EPS and (e.g.) PDF, you can use both the \texttt{latex+dvips} and \texttt{pdflatex} workflows without changing your source.

One other useful special case: including the output of \texttt{METAPOST} is also easy; although it is technically an EPS file, it uses only a small set of com-
mands. So \texttt{pdflatex} can support the inclusion of \texttt{METAFONT} output directly. The only thing you have to do is to change the file extension of the output file to \texttt{.mps}.

**Tools for image conversion**

There exist several tools for conversion of graphics formats, both free and commercial. Besides free GUI-based tools like Gimp on Unix systems there are two command line tools available for Unix and Windows: ImageMagick [2] and netpbm [3]. ImageMagick can convert images directly, e.g. by typing

\begin{verbatim}
convert sample.gif sample.png
\end{verbatim}

while netpbm uses the pnm format as intermediate format:

\begin{verbatim}
giftopnm sample.gif | pnmtopng -> sample.png
\end{verbatim}

Another nice tool is \texttt{tif2eps} by Boguslaw Jackowski et al. [4] which uses Ghostscript to convert a TIFF file to EPS, e.g.

\begin{verbatim}
gs -- tif2eps.ps sample.tif sample.esp -rh
\end{verbatim}

which produces a RLE compressed and hex encoded EPS file. In my experience EPS files produced with \texttt{tif2eps} are smaller than those produced by ImageMagick. Additionally it supports CMYK TIFF files smoothly.

**Additional tools**

There are many other helpful tools. I will mention two I use quite often.

\texttt{overpic} is a \LaTeX{} package written by Rolf Neprachk [5]. It includes an image into a \LaTeX{} picture environment, giving you the opportunity to add new elements into the image with normal \LaTeX{} picture commands. Fig. 3 shows a map overlaid with symbols and text at some points. The source code for this picture looks like

\begin{verbatim}
\usepackage[abs]{overpic}
...
\begin{document}
\begin{overpic}[grid,tics=5]{map}
  \put(32,74){\includegraphics[scale=.3]{busstop.mps}}
  \put(32,77){\llap{\scriptsize\colorbox{back}{Windm"uhle}}}
  \put(28,63){\small\textcolor{red}{\ding{55}}}
  ...
  \put(17.5,11){\scriptsize\colorbox{back}{\Pisymbol{ftsy}{65} Fr}}
  \put(6.3,13){\colorbox{back}{\Pisymbol{ftsy}{68}}}
  \put(29.8,61.4){\color{blue}\vector(-1,-3){2}}
  \put(38.6,63){\color{blue}\vector(1,3){2}}
\end{overpic}
\end{document}
\end{verbatim}

\texttt{potrace} is a tool to convert a pure black and white bitmap to vector graphics [6]. Fig. 4 shows a sample bitmap converted to a vector image.

**References**

[1] CTAN:macros/latex/required/graphics
Making a booklet

Joe Hogg
Los Angeles, CA
Joseph.Hogg@bigfoot.com

Background
In May 2004, I became a docent at the Los Angeles Zoo and Botanical Gardens and joined the Docent Botany Committee. The Zoo has a valuable collection of plants distributed throughout the Zoo, and two Botany Committee members had written a self-guided tour of this collection. One of the authors created botanical illustrations that she wanted to include in the tour booklet. The draft of the booklet was in a Microsoft Word file and the drawings were in the artist’s notebook.

Feasibility
I decided to use \TeX to typeset the booklet since I knew \TeX could produce a high-quality product. I purchased a copy of Kopka and Daley’s Guide to \LaTeX [1], and used the fp\TeX software from the book’s enclosed CD to typeset the botanical tour booklet and several other projects.

We outlined the project’s timetable and budget, getting preliminary estimates of printing costs from four printers for initial runs of 500, 1,000 and 2,000 copies. We estimated that the booklet would be no longer than 40 pages and, based on unit costs for each run level, decided that 1,000 copies gave us the right balance between unit cost and the number of copies we thought might sell. Burbank Printing Center, with whom we worked, recommended printing the body of the booklet in black and white and reserving color for the cover in order to keep costs down. To further manage costs, I agreed to typeset and deliver the booklet as a pdf file. The printer agreed to perform the imposition step since he wanted the flexibility to choose the paper size for his offset press. The plate would be created directly from the imposed pdf file. The binding would be saddle stitched using metal staples.

The budget guidelines translated directly into design guidelines which, if followed, would allow us to have an economically feasible project. It was important to have these guidelines at the beginning because many times during the editing and typesetting steps, we considered changes to the booklet that, if adopted, could easily have made the project uneconomic. What we learned at this step helped keep us on course.

First steps
With a preliminary estimate of costs and an notion of how the final product might look, I started the typesetting process. I saved the Word file as plain text and opened it in the WinEdit editor I licensed for this project. I scanned the botanical illustrations and selected twenty-two for the booklet, with the goal of placing one or two illustrations per page next to the text discussing those plants.

While this article presents the typesetting steps as a sequential process, the actual process included many discussions of style and a great deal of editing and experimenting with \TeX and various packages. The level of team effort was high and all of us were committed to the project’s success.

Page layout
Our conversation with the printer indicated that a page size of 8.5 inches high and 6.5 inches wide might be economical for the printer and work for the text and graphics. The draft text had several short chapters, some no more than one or two pages of this size. A text block five inches wide gave comfortable left and right margins of 0.75 inches. I wanted the graphics to be less than half the width of the text block and placed on either the right or left side of the page with text wrapping around the graphics. I decided to make all drawings 1.75 inches wide with light gray backgrounds to create a mat effect. This set off the pen and ink drawings from the text while balancing the color weight between text and graphics.

I used the \texttt{report} document class for the booklet since I needed a separate title page and chapter headings. I thought I might use section headings, but did not in the final draft. The text was straightforward and the chapters short. There was no need for running headers or footers. A simple page number in the middle of the footer was enough. After a few tests with various fonts, we decided on 10 point Computer Modern for the body text.
The graphics were placed on the page using the \texttt{picins} package. This package has many options for fine-tuning the placement of the graphic and generally worked well. Nevertheless, \texttt{picins} seems to force more than a normal amount of space between paragraphs. I'll discuss a work-around for this in the section on final tuning.

All chapters but the last described plants in a particular geographic area of the Zoo. These geographical areas are identified by the names of continents since animals and plants from those areas of the world are displayed in these areas of the Zoo. The last chapter contained a bloom calendar and was organized in a list environment.

I tried to have all chapters start on the right page of a spread. Occasionally, I would need to insert a blank page, for which I used the \texttt{nextpage} package.

At this stage, the body of the booklet was largely done and contained twenty-four pages.

\section*{Navigation}

We used both botanical names and common names for plants throughout the text, and there were various categories and concepts we wanted readers to find easily. The index was developed using the \texttt{makeidx} package. In addition to indexing the names of plants, we added categories such as: food for (e.g., koalas); food, source of (e.g., chocolate); ancient plants (e.g., cycads); commercial plants (e.g., carob tree); and California native plants (e.g., California lilac). The final index was five pages long.

There are twenty-two botanical drawings in the body plus a map of the Zoo. \TeX provides the capability for a List of Figures, but we thought that was overkill for such a short work. Nevertheless, we wanted the reader to be able to see the list of illustrations and find one of particular interest. The solution was to integrate the Table of Contents and the List of Figures using the \texttt{tocbibind} package. Figures are numbered consecutively, but fall under the appropriate chapter headings in the Table of Contents.

Every chapter and figure has its own text or caption. To avoid the repetition of the words “Chapter” and “Figure” in the Table of Contents and text, I suppressed the chapter numbering and only included the number of the figure and the name of the plant illustrated in the figure caption. Chapter numbers and numbering of the top level of the list environment of the bloom calendar were suppressed using the \texttt{setcounter\{secnumdepth\}\{-2\}} setting.

The default figure captions were altered using features of the \texttt{caption} package:

\begin{verbatim}
\usepackage[font={small,it},labelsep=period]{caption}
\DeclareCaptionLabelFormat{numOnly}{#2}
\captionsetup{labelformat=empty}
\end{verbatim}

These commands set the illustrations’ titles in a small, italic font. An illustration’s number is separated from the illustration’s title by a period.

\section*{Front and back matter}

The first few pages of the booklet included Welcome, Remembrance, and Acknowledgments pages plus the Table of Contents. The Index and a map of the Zoo followed the text. The map was created in GIMP and printed inside the back cover. I edited the Table of Contents file to remove the page number for the inside of the back cover and substituted the phrase, “Inside Back Cover.” Also, column breaks in the Index did not always occur where I wanted them to. I inserted the \texttt{newpage} command to adjust this. After these changes, I ran \texttt{pdflatex} one time to get a correct Table of Contents and Index.

\section*{Cover}

I created the cover in Macromedia Fireworks using botanical drawings for the front and back outside cover pages. All graphics in the booklet were saved in portable network graphics (png) file format. Including the cover pages, we managed to keep the booklet length to forty pages.

\section*{Final tuning}

To add visual interest to the first page of each chapter, I added a relevant quotation at the top of the page using the \texttt{quotchap} package and a dropped capital letter, using the \texttt{lettrine} package, at the start of the chapter text. Readers have enjoyed the quotations.

In a few instances, \TeX failed to correctly hyphenate some of the botanical names. This was easy to fix by manually adding the correct hyphenation to a list processed with the \texttt{hyphenation} command in the preamble. A few other instances required minor editing to let \TeX correctly break a line.

I mentioned above that the \texttt{picins} package seems to add more space than normal between the paragraph in which it is used and the previous paragraph. I tried to correct this by fiddling with the parameters in the package, but was unsuccessful. I ended up with a brute-force solution: between the paragraphs affected, I simply added a command \texttt{\vspace{-0.375\baselineskip}}. That reduced the excess spacing to approximate a normal break.
Comments

Using the \texttt{report} document class made typesetting the booklet easier than custom designing all features of the page layout. Nevertheless, I often felt I would have benefited from the advice of a professional book designer.

\textit{A Botanical Tour of the Los Angeles Zoo and Botanical Gardens} was published November 2004 and Zoo docents are using it as a manual to add botanical comments to their public tours. The booklet is on sale at the Zoo’s gift shops.

\begin{quote}
Let it be borne in mind how infinitely complex and close-fitting are the mutual relations of all organic beings to each other and to their physical conditions of life.

Charles Darwin
\end{quote}

Welcome

In March 2002, the Zoo received accreditation from the American Association of Museums for both its animal and plant collections. In recognition of this honor, the City of Los Angeles changed its name to the \textit{Los Angeles Zoo and Botanical Gardens}. We are proud of our gardens and eager to share them with all the Zoo’s visitors. We hope you will see the Zoo with fresh eyes and take away a more memorable experience for having seen this diversity.

The Zoo is divided into areas named for continents which display animals and plants native to those continents. Animal enclosures are placed on circular walkways or loops with plants gracing these paths and enclosures. Australia, North America, Africa, Asia, and South America make up the geographical loops. There are also areas for Aquatics with water-dwelling animals; the Aviary; Treertops Terrace, a prairie and meeting area; and the Reptile House. Major walkways link all of these areas together.

In addition to trees and shrubs identified in this tour, we have included in the second half of this booklet a calendar listing those plants likely to be in bloom at various times of the year. We hope the tour will delight and inform you today as well as encourage you to return often to see the Zoo in bloom.

References

\LaTeX\ on the Web

Peter Flynn  
Electronic Publishing Unit, University College, Cork, Ireland  
pflynn@ucc.ie  
http://imbolc.ucc.ie/~pflynn

Abstract

This paper presents some techniques for use when authoring in \LaTeX\ which can be used to minimize the conversion problems where a document is to be converted to HTML for serving to the Web, while continuing to produce the quality of typesetting for PostScript/PDF that users have become accustomed to.

The Web as the new mode of publishing

Between \TeX\ users, distributing source documents via the Internet (whether Web, email, or other facility) is easy because the types of files are known (e.g. \texttt{.tex}, \texttt{.bib}) and they are easily reprocessed. It is also relatively easy to make the final-quality typeset output available to others as single files using PostScript or PDF.

However, PostScript files can be huge, especially if they contain bitmap graphics, and they require a reader which few outside the graphic arts field have installed (although it’s simple to do and freely available). PDF browsers are readily available, and many of them implement simple HTML-style hyperlinking for cross-references and URIs, but they have had other limitations, including font instability, the lack of inward addressability, and some typographic alignment problems.

Large or complex documents benefit from being split into chunks for serving, and from being served faster than PostScript or PDF by using HTML or XML and CSS. But HTML editors are notorious for their lack of structural, typographic, and document management facilities—which \LaTeX\ users are accustomed to having at their disposal. For all the hype, XML editors have failed miserably to live up to or exceed these expectations.

Conversion from \LaTeX\ to HTML is widely available, but unavoidably suffers from the inherent mismatch between feature-sets, and from the inherent reprogrammability of \LaTeX\. Authoring in XML, with conversion both to HTML/CSS and to PDF via \LaTeX\ is one option, but has its own drawbacks in the learning curve and the early quality of some software.

Between consenting \TeX\ users . . .

For \TeX\ files, the file format is known and expected, and the software to handle it already exists and is installed. \TeX\ files are small by comparison with other systems, and can be transferred by HTTP, FTP, email, etc., without licensing or copyright issues, and there are no known security problems (viruses, worms, etc.).

In fact, it’s not necessarily so easy. The user’s browser, client, or operating system may not know what to do with the file types, even if a \TeX\ system is installed. The sender’s server or operating system may not know what to do with the file types either, and may serve them with inappropriate or misleading labels, i.e., MIME Content-Types.\footnote{There was a proposal some years ago to register \texttt{.tex} etc as known media types, but we are still using \texttt{application/x-latex}. TUG needs to do something about this or someone else will register them as something else.}

To be sure the files can be used properly, we need to know what ancillary files are needed, and whether non-default fonts need to be sent as well. Even where the user’s system recognises the file types, what do you do actually want to do with \texttt{.tex} files when you click on them in a browser or FTP client window? Display them? Edit them? Process them? Save them? Rename them to something else?

Serving non-source files

For many applications it is sufficient to serve the output, rather than the source. While it is possible to serve DVI files, it’s unusual because there can be problems if there is any use of fonts outside the default set supplied with a standard installation of \TeX\ or \LaTeX, and there can also be problems with the inclusion of images, which need to be provided separately. It will almost certainly be self-defeating if the objective is co-operative editing.

PDF

For final-format documents, it’s important to remember that most users nowadays have never seen or heard of a PostScript file: as explained earlier,
PDF is ubiquitous, there is a choice of viewers, it has become the *de facto* standard, and it provides for hyperlinks.

Against it count the poor handling of bitmap fonts in some Adobe readers, and the problems experienced with the feature for shrinking or expanding the page-image to fit the paper. This is probably the cause of most grief: if you have carefully set \LaTeX to produce an exact text width and height, with margins to fit your size of paper, it can be surprising to get email back from a user complaining either that it doesn’t print properly, or that the dimensions are not what you claim they are. In these cases they have almost certainly allowed Acrobat Reader to scale the page up or down for their local paper size.

**HTML** To take full advantage of the Web today means serving HTML (or, increasingly, XHTML, the more rigorous XML version of HTML). From a \LaTeX source this means using a conversion program, of which there are several available: the two most common are \LaTeX2HTML and \TeX4ht.

However, many browsers still lack rendering ability and font control. The use of home-brew macros will often defeat convertability by making it virtually impossible for a converter to figure out how the output is to appear. Generally, the converters do an excellent job, but they cannot be 100% error-free. Differences between browsers can cause user community problems if not everyone uses the exact same version and build (usually only achievable in tightly-controlled corporate environments). But the fundamental reason for differences in rendering is simply the feature-set mismatch between TeX and HTML: they are intended to perform different tasks, so it is perhaps unreasonable to expect them to be completely congruent.

**XML** The long-term solution to many of these problems may be to author in XML and convert to HTML for interactive browsing and to \LaTeX or ConTeXt for generating PDF.

XML has the advantages that it is very controllable, conversion is robust, it is a *de facto* standard, and that open source solutions are available throughout the process.

Against it stand a steep learning curve for authors unused to structured document markup (\LaTeX users have a head start here), the poor quality of much XML editing software, and the need for the additional steps to get to HTML or PDF.

Overall, the fact that an XML document can be reused much more successfully than a \LaTeX document tends to indicate that this is the direction for authoring, provided the problems with editors can be overcome.

**Making more use of your \LaTeX**

For successful conversion to HTML there are some key steps you can take:

- Keep your source code neat;
- Make it predictable and recognizable;
- Use white-space above and below all standalone control sequences;
- De-abbreviate any home-brew shortcut macros before conversion;
- Re-use equivalent \LaTeX environment and command names rather than inventing your own;
- Above all *be consistent*.

An example of this put into practice might be something like this:

```latex
\section{Making more use of your \LaTeX}

For successful conversion ...

\begin{itemize}
\item Keep your source code neat;
\item Make it predictable and recognizable;
\item Above all, \textit{be consistent}.
\end{itemize}
```

**Seen and heard ...**

During the course of the Practical \TeX conference, a number of people had been discussing these topics, and I noted down a few of the opinions:

- Author in HTML and make images for formulae;
- Author in XML and use XML\LaTeX;
- Author in Word using named styles, convert to XML with \textit{DynaTag}, then to \LaTeX;
- Author in Word and use the inbuilt ‘Save As... XML’, then convert to \LaTeX;
- Author in \textit{AbiWord} and ‘Save As...’ both \LaTeX and XML;
- Write a version of \LaTeX that outputs HTML (after all, it works for PDF...).

As always, there is an infinity of solutions to choose from, and getting your own document onto the Web may involve pieces from more than one of the pathways I have described.
Beamer by example

Andrew Mertz, William Slough
Department of Mathematics and Computer Science
Eastern Illinois University
Charleston, IL 61920
cfaem@eiu.edu, cfwas@eiu.edu

Abstract

There are a variety of \LaTeX \ classes which can be used to produce “overhead slides” for presentations. One of these, beamer, provides flexible and powerful environments which can be used to create slides and PDF-based documents suitable for presentations. Although the class is extensively documented, first-time users may prefer learning about this class using a collection of graduated examples. The examples presented here cover a wide spectrum of use, from the simplest static slides to those with dynamic effects.

Introduction

\LaTeX \ users in search of a way to produce “overhead slides” for use with an LCD projector have many choices today — perhaps too many! For example, Michael Wiedmann has a web site [6] that lists 12 different tools, all \LaTeX-based, capable of producing PDF output.

For first-time users, it can be difficult to decide which of these many approaches best matches their needs. In our experience, we have found the beamer class [5] to be easy to use, flexible, and well documented.

The user’s guide for beamer is, of course, the ultimate authority for its use. However, at 203 pages, a potential user might be frightened off before having a chance to experience its capabilities. Our intention here is to provide a sampling of beamer’s capabilities by displaying a variety of examples.

A first example

A beamer document consists of a sequence of frames. In the simplest case, all of a frame’s content is displayed at once. A frame of this type is the electronic equivalent of an overhead transparency.

Figure 1 gives a complete example of a beamer presentation, stored in a file named talk.tex. In the preamble, familiar \LaTeX \ commands appear. The body of the document specifies a title page is to appear, followed by two frames. Each frame environment specifies the desired frame title and the contents to appear on that frame.

Processing the talk.tex source with pdflatex yields talk.pdf, a PDF file suitable for presentations. Figure 2 shows the resulting output.

\documentclass{beamer}
\title{A Tiny Example}
\author{Andrew Mertz and William Slough}
\date{June 15, 2005}
\begin{document}
\maketitle
\begin{frame}
\frametitle{First Slide}
Contents of the first slide
\end{frame}
\begin{frame}
\frametitle{Second Slide}
Contents of the second slide
\end{frame}
\end{document}

Figure 1: Contents of our initial talk.tex.

Using this simple example as a template, a newcomer to beamer can produce a wide variety of presentations — in effect by learning about just one new environment, the frame.

Frame content

A frame can be subdivided into the following basic components, many of which are optional:

- Head line and foot line
- Left and right sidebars
- Navigation bars
- Logo
Beamer by example

Figure 2: The three output frames of talk.pdf.

\begin{frame}
\frametitle{Practical \TeX\ 2005 Events}
\begin{center}
\begin{tabular}{|r|l|l|}
\hline
8-9 am & Registration & \\
9 am & Karl Berry & Opening \\
9:15 am & Nelson Beebe & Keynote address \\
10:15 am & Break & \\
10:30 am & Peter Flom & A True Beginner Looks at \LaTeX\ \\
... etc ...
\end{tabular}
\end{center}
\end{frame}

Figure 3: Source and output for a frame with a centered table.

- Frame title
- Background
- Content

In our examples, we use relatively few of these components, choosing to emphasize the content over the more decorative elements which are possible. Our first example does include a navigation bar located at the lower right hand corner of the frame, which is present by default.

As far as frame content is concerned, most of the \LaTeX\ environments and commands work within \texttt{beamer} documents in the expected ways. So, for example, enumerated and itemized lists, mathematics, and tables can all be expressed in ways familiar to \LaTeX\ users. The only thing different is that these commands must appear within \texttt{beamer}'s \texttt{frame} environment. To illustrate, Figure 3 shows how a table can be centered on a frame, along with its corresponding output.

One special case is worth noting, however. If verbatim text is to appear within a frame, the frame must be marked as \texttt{fragile}. This is accomplished as follows:

\begin{frame}[fragile]
% ... frame contents ...
\end{frame}

To include graphics within a frame, facilities of the \texttt{graphicx} package may be used. Several points are worth noting related to graphics. To begin with, \texttt{beamer} automatically loads the \texttt{graphicx} package, so no explicit \texttt{\usepackage} statement is needed. For properly sizing graphics within a frame, it helps to know that \texttt{beamer} formats its output to a size of 128 millimeters by 96 millimeters, or 5.04 inches by 3.78 inches. The native graphics formats supported by pdf\LaTeX\ are JPEG, PNG, PDF, and MetaPost output.

Figure 4 illustrates how the graphics file named p2005.png can be placed within a frame.

Frames with color

Emphasis in presentations may be obtained by changes in color, in addition to the more traditional font changes. Since \texttt{beamer} automatically loads the \texttt{xcolor} package [2], colors can be specified using the syntax of \texttt{xcolor}. In particular, the “named” color model can be combined with a percentage using the \texttt{xcolor !} specifier. For example,

\begin{verbatim}
{\color{BlueViolet!30} A B C}
\end{verbatim}

will typeset the text “A B C” using the BlueViolet color, at 30% intensity.
A particularly effective use of color can be applied to a tabular environment, via the \rowcolors command from \texttt{xcolor}. In this command, a starting row number and two colors are specified. These two colors are used to alternately shade the rows of a table, beginning with the given row. For example,

\begin{verbatim}
\rowcolors{1}{RoyalBlue!20}{RoyalBlue!5}
\begin{tabular}{|r|l|l|}
0.8-9 am & Registration & \\
8-9 am & etc ...
\end{tabular}
\end{verbatim}

specifies three different options to be used with the \texttt{xcolor} package. The first option, \texttt{pdftex}, provides information about the correct color driver to use. The option \texttt{dvipsnames} allows a set of predefined color names, such as \texttt{RoyalBlue}, to be used. (These named colors are sometimes referred to as the “Crayola” colors.) Finally, the \texttt{table} option informs \texttt{xcolor} that the \texttt{colortbl} package needs to be loaded. It is this last option that defines the \rowcolors command used in Figure 5.

Frames with two columns

Since frames have a landscape orientation, it can be helpful to be able to subdivide a frame into two columns. The \texttt{columns} environment of \texttt{beamer} is designed to meet this need.

Figure 6 provides an example of how a frame can be subdivided into two columns. The \texttt{columns} environment allows an alignment option which specifies whether columns are to be aligned along their top line, bottom line, or centered. In this example, the \texttt{c} option causes the two columns to be aligned along their vertical centers. Within the \texttt{columns} environment two columns appear, as specified with the two \texttt{column} commands.
Frames with overlays

Up to this point, the frames we have considered consisted of a single overlay. When the frame is displayed, everything on that frame appears at once.

Alternatively, a frame can consist of a sequence of overlays, which can be used to support incremental display. Overlays can be used to “hold back” information during a presentation or to produce certain kinds of animated effects.

As a matter of taste, some people feel that a single overlay is preferable, since information is not hidden from the audience. However, we feel that there are situations where multiple overlays are appropriate, especially when used judiciously.

The beamer class provides numerous ways to specify frames with multiple overlays. We illustrate three techniques:

- Using the \pause command
- Using overlay specifications
- Including multiple graphics files

Using the \pause command is a simple way to produce overlays. All text from the beginning of the frame to the place where a \pause command appears is formatted and placed into an overlay. In this way, the example in Figure 7 creates a frame with three overlays. When the resulting PDF file is viewed, the three lines of output are incrementally revealed. The command

\setbeamercovered{dynamic}

is in effect, so overlays not yet revealed will faintly appear. This allows the speaker to focus on the current overlay, yet not entirely hide information from the audience.

Our second example of incremental display involves overlay specifications. In beamer, every overlay within a frame is assigned a number, starting with one, reflecting the order in which they are displayed.

\begin{frame}
\frametitle{Overlays with \tt pause}
\setbeamercovered{dynamic}
Practical \TeX\ 2005 \pause
Practical \TeX\ 2005 \pause
Practical \TeX\ 2005
\end{frame}

Figure 7: Specifying multiple overlays with pause.

Figure 8 illustrates how a game of tic-tac-toe can be displayed. In this example, there are ten overlays—one for the grid, and one for each successive play in the game. The syntax

\onslide<->\{text\}

indicates that the specified text is to appear on every overlay from m onwards.

Initially, only the grid should appear, so no X or O appears on the first overlay. In this example, the first play is an X in the upper right-hand corner, so this X appears on overlay 2 and every successive overlay. O counters by playing in the center, so this O appears on overlay 3 and every successive overlay. The rest of the example follows in a similar way. In this situation, we do want to hide from view overlays not yet revealed, so the command

\setbeamercovered{invisible}

is appropriate.

For our third example of incremental effects, we use the facilities of the package xmpmulti and a drawing tool which supports multiple layers. We use xfig, but many other choices are possible.
Revisiting the game of tic-tac-toe, a grid can be drawn on layer 0, followed by the first move on layer 1, the second move on layer 2, and so forth. After all layers of the drawing are complete, each layer is exported to a PDF file with a suffix which matches the layer number. For example, layer 0 is exported to \textit{game-0.pdf}, layer 1 is exported to \textit{game-1.pdf} and so forth. The \texttt{\multiinclude} command of the \texttt{xmpmulti} package, illustrated in Figure 9, causes the graphics files to appear as overlays.

It is worth noting that \texttt{beamer} overlays are numbered beginning with 1, but \texttt{xmpmulti} considers the first overlay to begin with 0.

\textbf{Ornamental aspects}

It is possible to “dress up” \texttt{beamer} presentations in a variety of ways. For example, one could use the Microsoft Comic Sans font, which can be made available with the \texttt{comicsans} package \cite{4}. Once this font has been established within the \LaTeX{} system, adding the following two lines to the preamble of the \texttt{beamer} document will make it the default font:

\begin{verbatim}
\usepackage{comicsans}
\renewcommand{\sfdefault}{comic}
\end{verbatim}

Figure 10 shows a resulting frame.

Another possibility is to choose from among the many different \texttt{beamer} themes. As an example, adding the following lines to the preamble of our \texttt{beamer} presentation gives the title frame shown in Figure 11.

\begin{verbatim}
\usepackage{beamerthemesplit}
\usetheme{Berkeley}
\usecolortheme{dolphin}
\end{verbatim}

In addition to the visual appeal of themes, additional navigation tools are incorporated in the frames based on the \LaTeX{} sections and subsections present in the \texttt{beamer} presentation.

\textbf{Producing N-up output}

It is often desirable to produce a printed document which mirrors the content of the \texttt{beamer} presentation. There are two steps needed to accomplish this; first, create an overlay-free version of the presentation, and second, produce an N-up version of the presentation.
Removing overlays from a \texttt{beamer} presentation is easily done within the preamble, by adding the \texttt{handout} option:

\begin{verbatim}
\documentclass[handout, xcolor=pdftr, dvipsnames, table]{beamer}
\end{verbatim}

Processing a \texttt{beamer} document with this option causes all overlays for a given frame to be collapsed into a single frame.

Once overlays have been removed, putting multiple frames onto a single sheet of paper is a separate problem related to PDF files. The \texttt{pdfpages} package [3], for example, solves this problem. For an automated approach based on the same package, \texttt{pdfjam} [1], a Unix shell script, can be used.

\section*{Conclusion}

The contribution of many individuals in the \LaTeX{} community have made it possible to produce overhead slides using typesetting standards of the highest quality. We are especially indebted to the work of Till Tantau and the other package designers cited earlier.

\section*{References}

[1] David Firth. PDFjam. \url{http://www2.warwick.ac.uk/fac/sci/statistics/staff/academic/firth/software/pdfjam}.


\textbf{Figure 11}: Title frame composed with the Berkeley theme.
1 Introduction

A constant criticism of \TeX{} is that it is not user-friendly. In today’s computer environment, users expect to be able to click buttons and choose menus, and to see a result immediately. Of course we know that \TeX{} is a mark-up language, and there is no way that all of \TeX{}’s power can be accessed via menus and buttons. But with some limitations, it turns out that a graphical user interface (GUI) can be useful. In this article I will describe my attempt at such a GUI, which I have called \texttt{Batch Commander}.

2 Why \texttt{Batch Commander}?

When I first started on this project, I chose a name with ‘\TeX{}’ in it. Then I realised that the GUI need not be restricted to \TeX{}, but can be used with other programs that take text as input and produce graphical output. (The \TeX{} preview is indeed a graphical output, even though it contains mainly text.) I have already used the GUI successfully with Povray (http://www.povray.org), and I believe it can be used with MetaPost.

Using the \texttt{Batch Commander} with different programs is simple, given the appropriate setup. When the main file is chosen from the pop-up menu, the file extension tells \texttt{Batch Commander} which program should be used. If the extension is .\texttt{tex}, then it knows that the config file (we’ll come to this later) must have a .\texttt{sty} extension, and that the file should run through \TeX{}.

If it has a .\texttt{pov} extension, then it will use .\texttt{inc} for the extension of the config file, and run Povray on the main file.

3 Limitations

Let’s look at the limitations within which we will be working.

3.1 Use only global controls

\texttt{Batch Commander} will only be used to apply \textit{global}, as opposed to \textit{local} controls. Thus, we will be able to change parameters such as \texttt{\baselineskip} and \texttt{\textwidth}, which apply to the whole document, but we cannot use a command like \texttt{\emph{O}}, which applies to part of the document. So in principle, we can design a whole class file, but we cannot apply a style to a particular section of the text.

3.2 Use \texttt{\LaTeX{}}

Although the program I will describe can be applied to any \TeX{} program, e.g. plain \TeX{} or \ConTeXt{}, I will only consider a standard \texttt{\LaTeX{}} file. We will use the predictable structure of a \texttt{\LaTeX{}} file to our advantage. We will be able to load packages and modify the options, and we’ll also be able to load any \TeX{} command or primitive just after the \texttt{\begin{document}}.

4 File structure

Let us consider the minimal \texttt{\LaTeX{}} file shown in figure 1(a). There are two areas of ‘global controls’ which affect the output in this file:

1. The packages loaded, and their options;
2. (\texttt{\LaTeX{}}) commands such as \texttt{\baselineskip...}, which come after the \texttt{\begin{document}}.

Let us make this file simpler by putting all global controls into another style file. Figure 1(b) shows this simplification by introducing another file called \texttt{river_valley.sty}. (The \texttt{\AtBeginDocument{...}} command ensures that anything enclosed within the braces is read only after \texttt{\begin{document}}.)

Separating the document file from the ‘controlling’ file means that once we have decided on a set of controls, they can be easily applied to other documents.

5 The overall concept

Our goal is to have a GUI that allows us to control the contents of \texttt{river_valley.sty} interactively, have the file saved to disk, run the main file through \TeX{} immediately, and finally show the preview.

The operating system I have used is Macintosh OS X (Tiger). As far as possible I have tried to keep the components of the system platform independent, so that it can be ported easily to other systems.

5.1 Controls

Figure 2 shows the process of typesetting a file using the controls of a GUI. By ‘controls’ we mean graphical objects such as buttons and menus which take the place of editing a text file. This is how the process works:

- The user makes a change to a control, e.g. types in a number, or clicks a checkbox;
\documentclass{article}
\usepackage[pdftex, linkcolor=red, ]{hyperref}
\AtBeginDocument{
  \baselineskip = 12pt
  \hyphenpenalty = 50
  \...
}
\documentclass{article}
\usepackage[pdftex, linkcolor=red, ]{hyperref}
\begin{document}
\section{Introduction}
This is the main text....
\end{document}

(a) (b)

\begin{figure}[h]
\centering
\begin{minipage}[b]{0.45\textwidth}
\begin{verbatim}
\documentclass{article}
\usepackage[pdftex, linkcolor=red, ]{hyperref}
\begin{document}
\baselineskip = 12pt
\hyphenpenalty = 50
\...
\section{Introduction}
This is the main text....
\end{document}
\end{verbatim}
\end{minipage}\hspace{0.5cm}
\begin{minipage}[b]{0.45\textwidth}
\begin{verbatim}
\documentclass{article}
\usepackage[pdftex, linkcolor=red, ]{hyperref}
\begin{document}
\baselineskip = 12pt
\hyphenpenalty = 50
\...
\section{Introduction}
This is the main text....
\end{document}
\end{verbatim}
\end{minipage}
\caption{Simplifying the main file by putting all ‘controls’ in an external style file. (a) The original file; (b) the new main file which reads in an external ‘controlling’ file at run time.}
\end{figure}

- the system immediately writes out a ‘config’ file, in our case ‘river-valley.sty’;
- the main file is run through \TeX,
- the screen preview is shown.

The idea is that the user sees only the controls, and the final screen preview, not the intermediate text files, unless he/she expressly wishes to.

6 Programming tools

6.1 The GUI

The GUI needs to be completely flexible, and have the following facilities:

- Buttons, pop-up menus, and a rich mix of other interactive features to allow efficient control of values for parameters, choice of options, etc.
- ability to read and write text files;
- ability to communicate with other programs;
- be easy to program
- be easily portable to different platforms.

I chose the Runtime Revolution product (http://www.runrev.com) for the development environment. Revolution is a successor of Apple’s HyperCard, which was a highly innovative scripting program written by Bill Atkinson, but which was neglected by Apple over the years, and effectively died a slow death. (In my opinion the ‘killing off’ of HyperCard was one of Apple’s worst decisions.) My familiarity with HyperCard allowed me a quick start in Revolution, and it has worked extremely well so far, with no major drawbacks.

An advantage of Revolution is that it is cross-platform, so the majority of the work writing the GUI need not be duplicated for other platforms.

6.2 Communication of Batch Commander with \TeX

After the interactive changes have been made, \TeX needs to be told to run the file and to show the preview. For this stage I used the scripting language of Revolution, i.e. Transcript, together with a mixture of shell scripts and AppleScript. This is the main area where each platform would need its own support.

6.3 \TeX implementation

I used pdf\TeX exclusively for typesetting the \TeX files. So when a document is typeset, a PDF file is produced in a single step. One reason for this is that we want to embed PDF-specific items, and we want to test that they are recorded correctly. pdf\TeX allows the fastest route to the generation of PDF.
The point to note is that each control can have different output forms, but the same type on the GUI. Equally, two controls might have different forms for selection, but the same output form. We will see later that this classification is useful in generating the controls automatically.

Having looked at the type of the control and the output form, let us look at other attributes of a control, so that we can define clearly what a control should look like. Figure 4 shows some attributes which a control might have. Here is a comprehensive list of the possible attributes of a control:

7.1 Group
When there are a lot of controls, it is convenient to group them together logically, so that only a selected number of controls are visible at any time. So we should make each control be associated with a particular group.

7.2 Name
Each control has a name. Depending on the output form, the name may or may not be written to the config file.

7.3 Description
It is useful to have a short description of each control, in order to remind the user what the control does. This might be shown permanently on the GUI, or might be a tooltip or a pop-up field.

7.4 Selection type
This is what we discussed above. It might, for example, be choice, number, or toggle.

7.5 Output form
As discussed above, the output form determines how the data corresponding to the control is written to the config file, e.g. option.

7.6 Position
The data from each control is normally written either as an option to a style file, within square brackets, or in an \AtBeginDocument{...} command. So the value given to position is either package or begindoc.

7.7 Unit
A control may or may not have a unit associated with it.

7.8 Minimum and maximum values
These apply in the case of number controls, where a value needs to be chosen from a range of numbers.

7.9 Choices
If the selection type is choice, then the list of possible choices must be specified.

7.10 Default value
This is the default, either of a numerical value, or from a list of choices. If the user does not interact with the controls, this is the value written.

7.11 Increment
For numerical values, this is the increment between successive values offered to the user as choices.

7.12 Decimals
This determines how many decimal units are displayed for numerical values of a particular control.

8 General anatomy of the GUI
Figure 5 shows the main overall control area of Batch Commander. The controls for a specific style file appear below this area. By looking at these overall controls, we can get a feel for the functionality of the GUI.

The top left pop-up button selects the main \LaTeX file which is to be typeset (figure 6). This file will be opened, and all current settings will be applied to it. If a file is not in the list of available .tex files, then by choosing New..., the file and its path will be added to the list.
The top right button determines whether any changes made should be applied immediately to the config file (river_valley.sty), or only after the user clicks the run button (figure 7). (On slower systems, the Delayed option is best.) With Immediate selected, as soon as any control is clicked, a new config file is written, and the \LaTeX file is run to show the preview. The style management button is used to make style files available to the user (figure 8).

When a style file is available, it is included in the list of checkboxes just below it. Whether it is ‘loaded’, i.e. included in the config file, depends on whether the checkbox is ticked (figure 9). Furthermore, the style file will be written out in the order that they appear in the GUI, i.e. controls.sty first, and geometry.sty last. There is an intuitive mechanism to reorder the styles, by simply dragging them on the screen.

If the Friendly mode button is not checked, then each control for a style will have the actual name of the control shown, i.e. the name that will be written to the config file, and holding the mouse over this name shows the ‘description’, i.e. a short explanation of what the control does. Checking the Friendly mode button reverses this mode. See figures 10 and 11. This is simply a user preference which does not affect the functionality of the GUI.
Batch Commander: A graphical user interface for TeX

<table>
<thead>
<tr>
<th>Geometry</th>
<th>the_name</th>
<th>description</th>
<th>selection_type</th>
<th>output_form</th>
<th>position</th>
<th>the_unit</th>
<th>min_val</th>
<th>max_val</th>
<th>choice</th>
<th>default</th>
<th>increment</th>
<th>decimals</th>
</tr>
</thead>
<tbody>
<tr>
<td>orientation</td>
<td>Orientation choice</td>
<td>option</td>
<td>package</td>
<td>-</td>
<td>-</td>
<td>&quot;Land portrait&quot;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paper</td>
<td>Paper choice</td>
<td>option</td>
<td>package</td>
<td>-</td>
<td>-</td>
<td>&quot;A4 paper&quot;</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textheight</td>
<td>&quot;Height of number&quot;</td>
<td>option_with_value</td>
<td>package</td>
<td>cm</td>
<td>1</td>
<td>30</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>textwidth</td>
<td>&quot;Width of text&quot;</td>
<td>option_with_value</td>
<td>package</td>
<td>cm</td>
<td>1</td>
<td>20</td>
<td>-</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Figure 12:** A data file for `geometry.sty`. The set of controls to be included can be easily modified.

**Figure 13:** The controls generated using `geometry.data` shown in figure 12. Notice that only the 'geometry' group is checked and visible.

**Figure 14:** As figure 13, but with both 'geometry' and 'Page' groups shown.

9 Generating the style controls

The underlying idea is that each style file has an associated set of controls. So we want it to be easy to create a set of controls when a style file is ‘loaded’.

The method I have used is to create a `.data` file which has all the information about the control. This file is read by Batch Commander when a style file is loaded, and the controls are created immediately. The data file is simply a tab-delimited text file which contains a list of the controls, with all the attributes discussed above.

Figure 12 shows an example data file; in this case, the file is `geometry.data`, containing the control data for `geometry.sty`. It is important to note that the `.data` file is not definitive for each style file, and is ‘designed’ by the person who writes it. In this case I decided that the six entries under the `the_name` column were the controls I needed for my purposes. Another user might require a different set of controls.

9.1 The data file

The way the data file is read in and interpreted can be best understood by examining figure 12, and the resulting GUI pages shown in figures 13 and 14. (To retain a reasonable text size, some of the items have been truncated, but it should be obvious what they are.) Here are some features of the data file:

- The first column is reserved for the group name. All rows between a group name and the first occurrence of ‘======’ are considered to be in that group. By grouping controls together, we can show only those groups we are interested in at one time.

- Apart from the first column, which denotes the grouping, all other columns can be written in any order. The system reads the title of the column in the first row, and thereafter assigns the correct attribute to each control.

- A single dash in a cell denotes ‘not applicable’. For example a toggle control cannot have minimum and maximum values. There must be no empty columns.

- If a description is more than one word, then it must be enclosed in double quotation marks.
Choosing a different style file (controls.sty), and revealing a new set of style controls.

The data file needs to be written manually, but only once for each style file.

10 Using the controls

The controls have been designed to be used intuitively and quickly. Pop-up menus in choice-type controls, for instance, are set using the mouse, while number-type controls can be set in a variety of ways: (1) direct input from the keyboard; (2) up and down arrows for increasing and decreasing the value; and (3) choosing a value from a small pop-up box. In cases (2) and (3), the change is determined by the increment value set in the data file.

The controls for each style file are shown on a separate page, or ‘card’, as each record is called in Revolution. By selecting the required style file from a pop-up menu on the left, the card corresponding to that style is shown. The main controls at the top remain, but the relevant style controls now appear below them, as can be seen in figure 15.

11 Writing the config file

When Batch Commander writes out the config file, it goes through each card (i.e. style file) in turn, gathers the data from the controls on that card, and then appends the data to the config file, to obtain a file as shown in figure 1(b). Figure 16 shows a typical config file. If Immediate is switched on, then as soon as any control is modified, the whole process of writing out the config file is redone, and the main file is run through TeX, in our case using TeXShop.

This works quite well, and does not seem to slow down the interactive facility. If the user manipulates the controls before the end of the cycle of writing the file and running TeX, then the cycle is silently abandoned and restarted.

12 Status of Batch Commander

At the time of writing, the program is at an ‘alpha’ stage. When set up, it works well and with very fast feedback, but it needs work in several areas, in particular:

• Improving its general stability and reliability
• Some support for undoing actions
• Ability to read data from a config file and set controls according to it
• Support for flexible file management, directory structures, etc.
• Porting to different platforms, for which I will need the help of others.
• Improving the structure of the data file, in particular obviating the need for the ‘======’, and allowing empty columns without a dash.

13 Availability

The program will be made available free of charge, although the license has not been finalized yet. If you would like to use the program, please mail the author.
Word to \LaTeX for a large, multi-author scientific paper

D. W. Ignat  
P.O. Box 1380  
Middlebury, VT 05753 USA  
ignat at mailaps dot org

Abstract

Multiple authors from diverse locations submitted to a scientific journal a manuscript of a large review article in many sections, each formatted in MS Word. Journal policy for reviews, which attract no page charges, required a translation to \LaTeX, including the transformation of section-based references to a non-repetitive article-based list. Saving Word files in RTF format and using rtf2latex2e accomplished the basic translation, and then a perl program was used to get the references into acceptable condition. This approach to conversion succeeded and may be useful to others.

Introduction

Twelve authors from five countries and ten research institutions proposed to the Nuclear Fusion journal (NF) of the International Atomic Energy Agency (IAEA) in Vienna, Austria, a review paper with six sections plus a glossary. This unusually large manuscript had some hundred thousand words and a thousand references. The sections had different lead authors, so that the references of each section were independent of those in other sections, while often repetitive among sections.

The IAEA gave review papers the privilege of waived publication charges ($150/page), but required authors to ease the publisher’s costs by submitting manuscripts of reviews in \LaTeX, the journal’s typesetting system. Therefore, a considerable financial incentive appeared for finding a somewhat automated transformation of all the Word sources into a unified \LaTeX source.

I was the editor of IAEA’s NF from mid-1996 to mid-2002 with primary responsibility for the refereeing system and the development of the journal. Previous experience in Unix and and \LaTeX for my own research brought an unofficial role as adviser to the IAEA production office on shell scripts, \LaTeX, regular expressions, perl, and web mounting.

Since the paper appeared valuable from the point of view of journal development, and at the same time a challenge in computer processing, I became particularly interested, and encouraged the authors to find ways to satisfy the IAEA requirement: a \LaTeX manuscript to better support refereeing and eventual publication.

In the end, the paper in question [1] was published in NF and was very well received by the research community, at great credit to the co-authors and also good for NF.

When the recent call for papers at Prac\TeX came in, it occurred to me that the story might be interesting for this audience.

Translation from Word to \LaTeX

At the time of submission (mid-1999) the IAEA and NF had investigated with a consultant conversions from Word to \LaTeX, but had not found a satisfactory solution. One of the twelve authors suggested rtf2latex2e by Ujwal Sathyam (now Ujwal Setlur) and Scott Prahl, following the Word-native RTF (Rich Text Format) writer.

Here is Microsoft’s description of RTF from msdn.microsoft.com:

“The Rich Text Format (RTF) Specification provides a format for text and graphics interchange that can be used with different output devices, operating environments, and operating systems. RTF uses the ANSI, PC-8, Macintosh, or IBM PC character set to control the representation and formatting of a document, both on the screen and in print. With the RTF Specification, documents created under different operating systems and with different software applications can be transferred between those operating systems and applications.”

rtf2latex2e uses the RTF reader by Paul DuBois and converts RTF files to \LaTeX. Some features are: detects text style (bold, italic, etc.); reads embedded figures; reads tables; converts embedded MathType; converts most Greek and math symbols; reads footnotes; translates hyperlinks. It
should compile on any platform that supports a C compiler. Versions for Macintosh, Unix-type systems, and Windows are available. The distribution, issued under the terms of the GNU General Public License as published by the Free Software Foundation, comes with example .rtf files.

The current, and final, version of rtf2latex2e can be found on the Comprehensive \TeX\ Archive Network, \url{http://www.ctan.org/tex-archive/support/rtf2latex2e} and at sourceforge.net.

The result of translation gave the expected

\documentclass{article}
\begin{document}
\section*{1. INTRODUCTION}

and looked good regarding mathematics and tables, but left all citations as footnotes with the expected chaos with repeated references. A typical reference (of the thousands) appeared many times with different but “look” similar. The footnotes were rendered, for example, as \cite{1.footnote{[1.] Author, A., Some Journal \textbf{36} (1997) 123.}] in section 1, but in generally the same way in section 2, except that the “1.” became “2.”.

One task is to transform the \footnote style that survives after the Word-RTF-\LaTeX transformation into the normal \cite{[...]}-\bibitem{[...]} representation of references. More complicated is to detect as identical those references to the same work presented with slight differences; and to detect as distinct those references that are actually different but “look” similar.

The goal was a unique citation in the body, such as \cite{AuthorA36p123}, and a corresponding entry in the bibliography, such as \bibitem{AuthorA36p123} Author, A., Some Journal, \textbf{36} (1997) 123.

The power of perl (Practical Extraction and Reporting Language) and its version of “regular expressions” made order from chaos, and produced material suitable for refereeing, and, eventually, publication.

Basic regular expressions

A “regular expression” (regex for short) is a generalized string for matching patterns, and possibly replacing whatever is found found with something else. The programs grep (Global Regular Expression Print), sed (Stream EDitor), and the text editor emacs, all of which are part of Unix-like systems, incorporate regex-es. (The tools mentioned above had versions workable under Windows 95, but comments on the capability of later Windows and Macintosh systems are outside the scope of this document.)

For a flavor of the regex world:

\begin{verbatim}
s/Old/New/g : Old → New globally (g) s/"Old/New/ : Old → New at line start ("") s/(...)Old/NEW\1/ : xxyOld → NEWxyz
\end{verbatim}

In the last example, the string Old is sought, but only if it preceded on its line by 3 characters, which are to be remembered by the parentheses ( ) with the label \1. Then, Old is to be replaced by NEW but followed by the 3 characters just found (here called xyz).

These examples only suggest the full power of searching and replacing available, in particular with perl.


Manipulating the references

The lead author of Ref. 1, Gianfranco Federici, contacted a colleague, Andreas Schott, about the challenge of rationalizing the references. Schott produced a perl script foot2cite.pl which accomplished the task.

A few years previously the NF Office had developed a collection of bash [9] shell and perl scripts to produce print masters and files for mounting PDF and HTML [10] articles on the IAEA web server. The \LaTeX source of individual articles led to tables of contents and indexes of authors and subjects from individual article source files with the help of native \LaTeX markup plus additional markup commands of the local style file. From that experience it appeared interesting to develop an IAEA-local program which could be the base of solutions that might be needed in the future. Some features of the resulting ref\_manip.pl are described in the following.

The idea is to use the “hash” facility in perl. Here, a hash is a 1-dimensional array in which the index and the value of the index are both character strings. The 1-to-1 hash num2cite connected the Word-original reference number, such as “1.101,”

\footnote{The utility of combining \LaTeX with scripting languages has been explored recently in TUGboat; see for example William M. Richter, “\TeX and Scripting Languages”, TUGboat, Vol. 25, No. 1, p. 71 (2004).}
to a string designed to be unique (except in pathological circumstances) such as "AuthorA36p123." For diagnostic purposes the 1-to-1-or-more hash cite2nums connected the (uniquely created) \cite and \bibitem string such as "AuthorA36p123" to the (multiple, in general) original reference numbers such as "1.101"; "2.45".

The multiple \LaTeX section files produced by rtf2lateg2e are scanned in sequence for a footnote. If footnote-style text of the nature author-journal-volume-page is found, then an identifier string is made of the first author’s last name, first initial, volume number, page number (AuthorA36p123). The text of the reference is entered in a holder for the bibliography under \bibitem{AuthorA36p123}, while the footnote is replaced by \cite{AuthorA36p123}. Next, the two hashes receive the appropriate entries — such as "1.101" and "AuthorA36p12" — with the help of a counter in the perl script. That counter should not become out of synchronization with the footnote numbers given in the paper unless there is a mistake in the original text.

The references not citing journal articles are detected by the absence of a bolding of an alphabetic volume number in a footnote. In that case, the \cite/bibitem identifier is formed from the first twenty alpha-numeric characters in the citation, excluding all white space. Again, the text of the reference goes to the bibliography as a \bibitem.

If the footnote is to a previously used number, such as [1.101] or [2.202], then the num2cite hash is used to enter the citation with the \cite format, without adding anything to the bibliography.

In the script as developed, pre-processing of the raw section files does, for example, the following:

- takes out explicit section numbering
- makes all citations (recall, they are of the \footnote type) begin in column one as \cite{...} and occupy one entire (sometimes very long) line
- makes the bolded volume numbers into a particular form that will not confuse later searches for a right brace closing the footnote.

That pre-processing is no doubt a sign of ignorance of the full power of perl, and no doubt extends the execution time. However, execution time is not a practical issue, but being able to construct the script in small pieces that do small, easily testable, things was very much an issue in the environment of the NF editorial and production offices.

The final pass changes \cite{[...]} into \cite{[...]}, writes the \bibitem{[...]} entries, and, optionally, saves the hashes num2cite and cite2nums for diagnostics.

There are vulnerabilities. A simple one, which could be programmed around, is that the original footnotes cannot contain inside them the characters { } or, other than for volume bolding, \}. A more difficult vulnerability, practically speaking inevitable, is that truly identical references have to be presented in pretty much the same way. There is probably no automated way to defend against typographical errors in the names, volume or page numbers. (The potential vulnerability to different amounts of white space had a simple defense.) However, an off-line sort of all the \cite and \bibitem texts would have a good chance of revealing a problem.

The result

The processing into \LaTeX of the first draft manuscript created one format completely common to all contributing institutions and authors. With that common form, adjustments in response to the concerns of the NF editorial office and referees became easier, as did changes originating with the paper’s authors as the review developed. Even so, the refereeing process was extensive, which is not uncommon for articles appearing in NF, and particularly articles of such a length.

Independent of what the authors of Ref. 1 feel about their article and the process of publishing it, the publishing journal and its home organization have interests.

The Institute for Scientific Information (ISI) keeps track of an “Impact Factor” (IF) for thousands of journals [11]. The IF is (at least approximately) the number of citations to a journal divided by the number of articles in the period studied. Nature and Science have IFs in the 20–30 range. The very prestigious Physical Review Letters has an IF around 6, and the Physical Review, (series A, B, C, D, and E) is typically between 2 and 3. Journals covering plasma physics and nuclear fusion range from 0.5 to 3 or so, and NF is consistently the highest in the group. In the six years ending in 2003 NF was between 2.2 and 3.4.

According to Google’s newly introduced “Scholar” service, articles from all journals covered referenced Ref. 1 23 times, which is unusually high for the sub-field of science and engineering covered by NF. (The time frame was not apparent from the information at Google.) Records available at the IAEA show that for Ref. 1 there were 162 downloads in 2003, placing it number 7 in the top 10 downloads for that year;
and that the citation rate is roughly double the next most cited article, and far above the average rate.

The numbers quoted above suggest that the research community received Ref. 1 unusually well, making it a fine credit to each of the authors and to their institutions. The numbers also say that the article had significant positive influence on the IF of NF, and therefore a positive influence on the continued success of NF. In other words, the appearance of this article was very good for its authors as well as the IAEA and NF.

Remembering that publication in NF, and at low cost to submitters, required a \LaTeX\ manuscript, one can wonder if all the good news would have happened without rtf2latex2e and perl. My speculation:
1. the research paper would have come out, if at all, later than it did,
2. it would not have appeared in NF,
3. the authors would not have gotten quite the recognition they did,
4. the IAEA and its journal would have a lower IF for the relevant period.

Acknowledgments
Co-author of rtf2latex2e, Ujwal S. Setlur, assisted the co-authors of Ref. 1 during the preparation of a manuscript that the IAEA would accept. As mentioned previously, Andreas Schott, a computer professional experienced in perl, produced the script that the co-authors actually used.

\LaTeX\ production and web-posting at the IAEA owed particular thanks to M. Bergamini-Rödler, N. Douchev, H. Giller, P. Gillingwater, F. Hannak, I. Kurtev, A. Primes, N. Robertson, M. Sherwin, J. Weil, and, for the \LaTeX\-to-HTML translations, I. Hutchinson, author of \ttt\ [10]. The management support of R. Kelleher and D. McLaughlin was indispensable, particularly as NF production processes grew to depend heavily on the tools of Unix shell scripts, perl, native and locally developed \LaTeX\ markup.

In January 2002, the Institute of Physics Publishing (IoPP) of Bristol, UK, assumed responsibility for production (again, based on \LaTeX) while the IAEA editorial office, located in Vienna, Austria, continued to manage content. The Federici paper [1] is now mounted on the web by the IoPP. The present editor of NF is F.C. Schüller of The Netherlands.

David Walden contributed helpful comments on a preliminary draft of this paper.

References
Biblet: A portable Bib\TeX{} bibliography style for generating highly customizable XHTML

Tristan Miller
German Research Center for Artificial Intelligence (DFKI GmbH)
Postfach 20 80
67608 Kaiserslautern, Germany
Tristan.Miller@dfki.de
http://www.dfki.uni-kl.de/~miller/

Abstract

We present Biblet, a set of \texttt{Bib\TeX}{} bibliography styles (\texttt{bst}) which generate XHTML from \texttt{Bib\TeX}{} databases. Unlike other \texttt{Bib\TeX}{} to XML/HTML converters, Biblet is written entirely in the native \texttt{Bib\TeX}{} style language and therefore works “out of the box” on any system that runs \texttt{Bib\TeX}{}. Features include automatic conversion of \texttt{Bib\TeX}{} symbols to HTML or Unicode entities; customizable graphical hyperlinks to PostScript, PDF, DVI, \texttt{Bib\TeX}{} and HTML resources; support for nonstandard but common fields such as \texttt{day}, \texttt{isbn}, and \texttt{abstract}; hideable text blocks; and output of the original \texttt{Bib\TeX}{} entry for sharing citations. Biblet’s highly structured XHTML output means that bibliography appearance can be drastically altered simply by specifying a Cascading Style Sheet (CSS), or easily postprocessed with third-party XML, HTML, or text processing tools.

We compare and contrast Biblet to other common converters, describe basic usage of Biblet, give examples of how to produce custom-formatted bibliographies, and provide a basic overview of the implementation details for those wishing to modify the style files.

Introduction

In today’s world of ubiquitous Internet access, it is becoming increasingly expected that every researcher, graduate student, professor, and other academic have a personal web page listing one’s contact information, qualifications, teaching schedule, ongoing and completed research projects, and publications. Normally such pages are maintained by the academic himself, and thanks to the extensive formatting capabilities of HTML [65], XHTML [81], and CSS [8, 47], authors can easily give their home pages a unique personal style.

Despite these tools, creating and maintaining an online list of publications has traditionally been a troublesome process. Authors must manually enter their bibliography data using the appropriate HTML\textsuperscript{1} and CSS markup to ensure that the list’s formatting matches the rest of the website. Since many authors already maintain a database of their publications in a format like \texttt{Bib\TeX}{} [57], this approach entails maintaining two separate bibliographies which can easily get out of sync.

Furthermore, if the author at some point decides to change the style in which the bibliography is displayed, CSS can help only so much. By altering the list’s style sheet, one can change the style of book titles from italicized to bold, or suppress the display of abstracts and annotations. However, CSS cannot make changes such as abbreviating author or journal names, switching the order of volume and issue numbers, or changing the sort order of publications from author to year. To make such changes, the author must tediously edit the individual list entries in the HTML file.

One solution to these problems is to use some tool to automatically convert the author’s existing \texttt{Bib\TeX}{} database to HTML, possibly employing some intermediate format such as \texttt{Bib\TeX}{} [44] or XML [9]. Then only one publication database need be maintained; the author can rerun the conversion whenever the \texttt{Bib\TeX}{} database is updated or whenever he wishes to effect a fundamental change in formatting, such as sort order.

In this paper, we present Biblet, one such tool for converting \texttt{Bib\TeX}{} databases to HTML web pages. We compare and contrast its features and

\textsuperscript{1} Hereinafter, unless otherwise noted, we use the term ‘HTML’ to refer to HTML and XHTML collectively.
capabilities to those of similar software, and discuss its limitations and the limitations of the underlying Bib\TeX{} format.

**Background**

**Using Bib\TeX**  Bib\TeX{} [56–58] is a bibliography program originally designed to work with Leslie Lamport's \LaTeX{} [44] document preparation system. To use it, the author creates a database of publications he wants to reference in a Bib\TeX{} database file with the filename extension \texttt{.bib}. The contents of this file are a series of records in a format similar to the following:

\begin{verbatim}
@ARTICLE{m05toc,
    author  = {Tristan Miller},
    title   = {The Tyranny of Copyright},
    journal = {Imagine},
    year    = {2005},
    month   = may,
    volume  = {4},
    number  = {1},
    pages   = {1,8-11},
    issn    = {1710-5994},
}
\end{verbatim}

The \texttt{@ARTICLE} token specifies the type of publication, which in turn determines the available fields. Other common publication types include \texttt{@MANUAL}, \texttt{@PROCEEDINGS}, and \texttt{@BOOK}. The text \texttt{m05toc} is the database key used to refer to the publication in \LaTeX{} citation commands and elsewhere in the \texttt{bib} database. The remainder of the record consists of a comma-delimited list of field-value pairs, where the values can be either predefined macros (such as \texttt{may} in the above example) or strings delimited with curly braces or quotation marks.

To use a Bib\TeX{} database called \texttt{imagine.bib} with a \LaTeX{} document \texttt{foo.tex}, the author must write the following commands at the place in the document where the bibliography\footnote{In this article we use the terms “bibliography” and “reference list” interchangeably.} is to appear:

\begin{verbatim}
\bibliographystyle{modern}
\bibliography{imagine}
\end{verbatim}

The first command tells the system how to format the bibliography and the second command tells it which database to use. To cite documents from the database, the \LaTeX{} \texttt{\cite} command is used:

\begin{verbatim}
... in my article \cite{m05toc} ...
\end{verbatim}

Note that the argument to \texttt{\cite} is the database key of the publication to be referenced. \LaTeX{} might typeset the above example as follows:

\begin{verbatim}
... in my article [1] ...
\end{verbatim}

At the end of the document in a separate section, the full bibliographic details of every publication cited appear, possibly as follows:

\begin{verbatim}
\end{verbatim}

The exact formatting of this reference list depends on the argument to \texttt{\bibliographystyle}.

In order to properly typeset the citations and references, the author must invoke the \LaTeX{} and Bib\TeX{} programs a number of times. Exactly what goes on behind the scenes is illustrated in Figure 1. (In this diagram, files furnished by the user are indicated by light rounded boxes, while computer-generated files are indicated by dark rounded boxes.)

First, \LaTeX{} is run on the \LaTeX{} document \texttt{foo.tex}, which produces an incomplete typeset version of the document, \texttt{foo.dvi}, and an auxiliary data file, \texttt{foo.aux}. This auxiliary file contains information for use by Bib\TeX{} — namely, the bibliography style, the bibliography database filename, and which publications from said database to include in the reference list. The contents of the \texttt{aux} file in our example might look as follows:

\begin{verbatim}
\bibliographystyle{modern}
\bibliography{imagine}
\end{verbatim}

Next, Bib\TeX{} is invoked on \texttt{foo.aux}. Seeing the \texttt{\bibliographystyle} and \texttt{\bibliography} commands, Bib\TeX{} searches for and opens the files \texttt{imagine.bib} and \texttt{modern.bst}. The bst file is actually a program which specifies how to convert a Bib\TeX{} bibliography — in this case, \texttt{imagine.bib} — into \LaTeX{} code. Bib\TeX{} scans \texttt{imagine.bib} until it encounters the \texttt{m05toc} entry, applies to it the transformation rules specified in \texttt{modern.bst}, and writes the output in a new file named \texttt{foo.bbl}. This bbl file contains \LaTeX{} code which, depending on the bibliography style, may contain something like the following:

\begin{verbatim}
\begin{thebibliography}{1}
\bibitem{m05toc} Tristan Miller.
\newblock The tyranny of copyright.
\end{thebibliography}
\end{verbatim}

When \LaTeX{} is next run on \texttt{foo.tex}, it inserts the contents of \texttt{foo.bbl} at the exact position where the \texttt{\bibliography} command occurs, producing a new version of \texttt{foo.dvi} which includes the list of references. Typically, \LaTeX{} must be run once more to resolve references, the \texttt{\bibitem} and \texttt{\cite} commands
being analogous to the \texttt{\textbackslash label} and \texttt{\textbackslash ref} commands used to create other types of cross-references \cite{44, \S 4.2}. The resulting DVI file is the final typeset version of the document; it can then be converted to a \texttt{ps} file for printing on a PostScript printer, or to a PDF for distribution on the Internet.

\textbf{Previous work} Because \LaTeX\ outputs \LaTeX\ command sequences, authors wishing to create an online list of publications typically have three options. The simplest but least convenient to the casual web page visitor is to simply post the \texttt{bib} database itself on one’s website, either by linking to the file directly or by embedding it as preformatted text in an HTML page. The former case can cause problems for some web browsers which correctly recognize the database as a file with the MIME media type \texttt{text/x-bibtex} \cite{20, 21} but do not know how to display it for the user. The second case is guaranteed to allow users to view the database from within their web browsers, albeit only in the crude original format.

The second option is to create a skeleton \LaTeX\ document citing the desired publications, run a \LaTeX\-to-HTML converter on it, and then extract the resulting bibliography for use on another web page. Programs implementing this approach include Nikos Drakos and Ross Moore’s \LaTeX\Xhtml \cite{16, 24, 26}, Eitan Gurari’s \TeX\4ht \cite{25, 31, 59}, Luc Maranget’s \Hvea \cite{50, 51, 77}, and over a dozen lesser-known applications \cite{2, 7, 10–12, 22, 28, 36, 53–55, 62, 69–73, 79, 80, 86}. These programs have the advantage that any existing \Bib\TeX\ bibliography style can be used.

The third option is to use a program which directly converts the \Bib\TeX\ database to HTML (or to XML, which is easy to convert to HTML). While there are several such utilities available \cite{4, 17, 23, 27, 29, 32, 33, 35, 42, 43, 48, 60, 61, 66, 68, 74, 75, 82, 85}, none of them seems to be as well-known or widely used as their \LaTeX\-to-HTML counterparts. Indeed, most of them have no presence on CTAN and some of them even share the same name.

There are a number of criteria to consider when evaluating software implementing these latter two approaches:

\textbf{License} Many people expect their software to be free in the sense that they may freely modify and redistribute the program for any purpose \cite{76}. The ability to modify the program necessarily implies that human-readable source code is provided. Most of the software cited above is available under a permissive license such as the GNU General Public License \cite{19} or the \LaTeX\ Project Public License (LPPL) \cite{45}, though some packages impose restrictions on commercial use or redistribution, and others have restrictive proprietary licenses and do not include source code.
Portability  Part of Bib\TeX{}'s popularity springs from its availability on a wide variety of computing platforms. To be useful for Bib\TeX{} users as a whole, any conversion tool should be available for a large subset of these platforms and should be installable with minimal effort. Most existing converters are implemented in widely available scripting languages such as Awk [67], Perl [83], or Python [49], or in portable compiled languages such as C [39]. Some, like bibtex2html [17], are implemented in relatively esoteric languages for which compilers are not widely available.

Standards compliance  In order to be displayed and indexed properly and consistently by all web browsers and other Internet applications, the HTML output by a converter should conform to the official W3C hypertext standards [5, 63–65, 81]. The HTML produced by some converters is not syntactically valid, or conforms to an obsolete standard.

Symbols  Bib\TeX{} bibliographies often use some \LaTeX{} markup for special characters such as accented letters and typographical symbols. A good converter should transform these into their Unicode/ISO 10646 [41, 78] or HTML [65, §24] equivalents. Unfortunately, even the most common and actively maintained converters often fail spectacularly in this regard, particularly with regard to basic punctuation such as quotation marks and dashes.

Math  Handling embedded math mode is a problem, since HTML and Unicode alone are not sufficient to display most mathematical constructs. Some converters transform \LaTeX{} math markup into MathML [3], though the latter is not yet widely supported by web browsers. Most (perhaps forgivably) ignore this problem and simply output the original \LaTeX{} code, though others resort to questionable solutions such as producing bitmap images (which do not scale with the surrounding text) or using deprecated, non-portable hacks [18].

Custom \LaTeX{} macros  Some bibliographies use custom \LaTeX{} macros, defined in the \LaTeX{} \texttt{\@PREAMBLE} or in a separate \LaTeX{} document. The wisdom of employing such macros is questionable, though in some cases they are the only way to get around \LaTeX{}'s inherent limitations.\footnote{An example of this is the oft-used \texttt{\noopsort} kludge to force proper sorting of non-English names [57, pp. 4–5].} Unfortunately, \LaTeX{} is notoriously difficult to parse by anything except \LaTeX{} itself, so most converters offer no or partial solutions to this problem.

Hyperlinks  The principal advantage of HTML is that it allows the inclusion of links to other documents. In the case of online bibliographies, it would be useful if each entry included a link to the document itself (for example, as a PostScript or PDF file) where available. Some converters cooperate with packages such as url and hyperref, or with bibliography styles such as natbib which support url, ps, and pdf fields. Others simply have no support for hyperlinks.

Anchors  HTML also allows links to certain points, called anchors, within documents. Since users may wish to link to individual entries in their publication list from other web pages, it would be useful if the converter would associate a unique anchor with each bibliography entry. Few existing converters implement this feature.

Styling  Perhaps most importantly, the converter's output should be adjustable by the user. The converter should allow at least as much variation in reference formatting as \LaTeX{} itself, but should ideally output its HTML in such a way that the appearance can be further customized via CSS. In this manner the online bibliography can be made an integral part of the author’s web page rather than a generic-looking computer-generated list.

Introducing Biblet

Though the existing converters we examined typically excelled in some of the above-mentioned criteria, none of them provided good support across the board. This lack of a combination of good features in existing software was the primary impetus for the development of a new tool, Biblet.\footnote{According to the Oxford English Dictionary, “biblet” is an archaic word of uncertain origin and meaning, though it is thought to refer to a small library.} Biblet sports the following features:

- It is freely available and redistributable under the terms of the LPPL.
- It outputs valid XHTML 1.0, making its pages viewable by any conforming web browser and facilitating postprocessing by XML applications.
- It makes use of an extensive \LaTeX{}-to-Unicode translation table, ensuring proper display of most typographical symbols.
- It has extensive support for internal anchors and external hyperlinks.
- Virtually every element of a bibliography entry is encapsulated in its own named HTML tag, allowing for extensive styling with CSS.
Biblet: A portable BibT\TeX{} bibliography style for generating highly customizable XHTML

Figure 2: Biblet workflow

```xml
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'
'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'>
<head><title>My publications</title></head>
<body>
  <h1>My publications</h1>
  <h2 class='bib-year' id='year-2005'>2005</h2>
  <ul>
    <li class='bib-bibitem' id='cite-m05toc'>
      <div class='bib-article'>
        <p>
          <span class='bib-author'>Tristan Miller.</span>
          <span class='bib-title'>The tyranny of copyright.</span>
        </p>
        ISSN 1710-5994.
      </div>
    </li>
  </ul>
</body>
</html>
```

Listing 1: Sample output of Biblet (abridged)

Significantly, Biblet is written entirely in the \texttt{BibT\TeX}{} stack language \cite{56}, making it portable to any system that can run \texttt{BibT\TeX}{} itself. It is, in effect, simply another bibliography style (\texttt{bst}) file, just like the standard \texttt{plain}, \texttt{abbrv}, and \texttt{alpha} styles. The difference is that while the latter output \texttt{L\TeX}{} code, the Biblet produces HTML.

The basic Biblet workflow is illustrated in Figure 2. Note that, unlike in the regular \texttt{BibT\TeX}{} workflow of Figure 1, there is no \texttt{tex} file as input; rather, the user creates the \texttt{aux} file directly. In this file, the user issues a \texttt{\textbackslash citation\{\textbackslash{}key\}} command for each bibliography item, or \texttt{\textbackslash citation\{\ast\}} to include all items. This is followed by a \texttt{\textbackslash bibstyle} command indicating which Biblet \texttt{bst} style to use and a \texttt{\textbackslash bibdata} command indicating which \texttt{bib} database to process. A sample \texttt{aux} file for use with Biblet might look as follows:

```latex
\citation{m05toc}
\bibstyle{blplain}
\bibdata{imagine}
```

The user runs \texttt{BibT\TeX}{} on this \texttt{aux} file as usual. The resulting \texttt{bbl} file, however, is actually an HTML document. It can be renamed and opened in any web browser or HTML editor.

Listing 1 shows the abridged contents of the \texttt{bbl} file produced in the above example, and Figure 3 shows how this file appears when opened in a web browser.

**Sprucing things up** The reader will note from Listing 1 that Biblet has enclosed most of the important parts of the bibliography in their own HTML.
Tristan Miller

Figure 3: Biblet output as viewed by the Mozilla web browser

elements. This makes it easy to alter the appearance of the list by applying CSS styles. For example, say we wish all entries of type @ARTICLE to be displayed with a pink background, author names to be printed in bold type, journal names to be underlined instead of italicized, and article titles to be enclosed in quotation marks. Rather than manually editing the HTML file produced by Biblet, or even the bst file which generates the HTML, we simply write a file mystyle.css as follows:

```
.bib-article {
    background-color: pink;
}
.bib-author {
    font-weight: bold;
}
.bib-article em {
    font-style: normal;
    text-decoration: underline;
}
.bib-title:before {
    content: "\201C";
}
.bib-title:after {
    content: "\201D";
}
```

(In this example, 201C and 201D are the hexadecimal values for left and right double quotation marks in Unicode.) To apply this style to our HTML file, we insert the following line into the <head> element:

```
<link href='mystyle.css' type='text/css' rel='stylesheet' />
```

When the browser view is refreshed, the new formatting takes effect, as shown in Figure 4.

Because Biblet is aimed at producing online document catalogues, it recognizes a number of special fields in bib databases. Among these are ps, pdf, dvi, html, tex, and txt, whose values contain URLs [6] referencing respectively PostScript, PDF, DVI, HTML, (La)TeX, and plain-text versions of the publication. Biblet converts these fields to textual or graphical hyperlinks within the bibliography entry. Other common fields such as abstract, isbn, and issn are also supported.

It is also possible for a Biblet bst style to output arbitrary HTML at the beginning and end of a bibliography, as well as within and in between entries. In fact, the default bst files distributed with Biblet produce far more extensive HTML than is actually shown in Listing 1. For example, styles which provide icon hyperlinks to online versions of the document will typically include a legend explaining the purpose of each icon. Other styles allow the option of outputting the original bib entry for researchers to copy and paste into their own BibTeX bibliographies, plus JavaScript [37] code allowing the user to toggle the display of the BibTeX entry and/or abstracts.

Biblet comes with several predefined CSS styles and icon sets for use with the publication lists it generates. Figure 5 gives a sampling of the available styles; note also the formatting of legends, abstracts, hyperlinks, original BibTeX data, and toggles.

Implementation

As mentioned previously, Biblet is implemented as a BibTeX style (bst) in the nameless BibTeX stack language. The idea for this approach came through the observation that the bst styles were solely responsible for the producing the content of the bbl files; that is, the BibTeX program itself did not write anything to these files unless directly instructed to do so by the bst style. The initial steps in the development of Biblet therefore involved going through Patashnik’s original plain.bst style and replacing all the BtBTeX markup it outputted with analogous HTML markup. Thus, {em ...} was changed to <em>... </em>, the thebibliography environment
Biblet: A portable BibTeX bibliography style for generating highly customizable XHTML

Figure 5: A gallery of four Biblet bibliographies: (a) uses the “Traditional” style with the “Nuvola” icon set; (b) uses the “Boxy” style with the “Noia Warm” icon set; (c) uses the “Fruity Typewriter” style with the “Slick” icon set; and (d) uses the “Amethyst” style with the “Nuvola” icon set. All examples were typeset with the \texttt{plain.bst} BibTeX style; only the CSS styles and icon sets were altered.

These simple substitutions resulted in a \texttt{bbl} file containing the bibliography as an HTML list (<ul>) which, while not a complete HTML document itself, could be cut and pasted into an existing web page. To make the \texttt{bbl} file stand on its own as a web page, it was necessary to modify the Biblet \texttt{bst} so it outputted some additional HTML markup at the beginning and end of the file. We also added some more HTML code to the bibliography entries themselves, wrapping parts of them in various named containers so that the user could later customize their appearance with CSS.

The next step was to add some custom sorting routines typical of those seen on hand-crafted author publication lists. Most academics sort their publications by year or by publication type (book chapter, article in journal, article in conference proceedings, etc.). Besides the actual sorting code, it was necessary to output HTML headers marking a change in the value of a sort key.

Finally, we had to write the code to convert any \LaTeX\ symbols contained in the Bib\TeX\ bibliography itself to Unicode or HTML entities. This was the most difficult and time-consuming of all the development tasks. The \texttt{bst} language is extremely crude, having been designed principally for ease of implementation on the computers of 1988; features that programmers take for granted in modern general-purpose programming languages, such
Figure 6: XML to bst

<char pos="127">
<entity name="para" set="iso-8879-num">
<desc>=pilcrow (paragraph sign)</desc>
</entity>
<entity name="para" set="html4-lat1">
<desc>pilcrow sign = paragraph sign</desc>
</entity>
<unicode value="00B6">
<desc>PILCROW SIGN</desc>
</unicode>
<latex>
<seq>
\P
</seq>
<latex>
<seq req="textcomp">
\textparagraph
</seq>
<seq req="textcomp">
\textpilcrow
</seq>
</latex>
<plain value="B6" set="iso-8859-1" glyph="¶"/>
</char>

Figure 7: Sample entry from ent.xml

as function arguments, arrays, local variables, string manipulation, and dynamic memory allocation are poorly supported or even completely absent. The features of Biblet previously discussed were accomplished with relative ease as they made use of the bst language’s built-in output and sorting routines. For the symbol conversion task, however, what appeared to be a simple search-and-replace routine ended up being a programmer’s nightmare.

For one thing, the language’s string operations are limited to concatenating two strings, returning the first n characters of a string, and returning the string length, though the latter treats strings containing special characters idiosyncratically and cannot be used directly for our purposes. It was therefore necessary to code our own string-length (string.length) and find-replace (find.replace) routines.

Because the bst language also does not support arrays, it was not possible to simply enter a static translation table pairing \LaTeX symbols with HTML entities, and then have some loop iterate over the table cells, calling find.replace. Instead, each symbol mapping had to be entered as a separate function call. This gave rise to another problem: function definitions are statically limited to 100 tokens. Since it takes three tokens to do a find-and-replace (one for the search string, one for the replacement text, and one for the call to find.replace), a maximum of 33 substitutions can be performed in one function. Since we needed to make over 500 symbol substitutions, we had to split the code over sixteen separate functions, and add a seventeenth function whose purpose was simply to call the others in sequence.

Rather than tediously coding all this by hand, we wrote a number of support programs to generate the code. Their operation is illustrated in Figure 6. For the \LaTeX-to-HTML mappings we used Vidar Bronken Gundersen and Rune Mathisen’s comprehensive database [30, 52] which they have kindly made available for any purpose. The database, which is distributed as an XML file named ent.xml, has entries for every SGML character [38, 40], giving data such as its name, Unicode value, and, where known, \LaTeX macro(s). A sample entry from ent.xml is shown in Figure 7. We used an XSL transformation (XSLT) [13] to convert the information in this database to a sequence of find.replace calls in a new file, trans.bst. The XSLT will opt to convert symbols to named HTML entities when possible; otherwise it will output numbered hexadecimal entities. Here are a few lines from trans.bst:

"\textparagraph" "&para;" find.replace
"\textpilcrow" "&para;" find.replace
"\checkmark" "&x2713;" find.replace

For a number of reasons, the trans.bst file output by the XSLT is not directly usable. First, as mentioned before, it needs to be split up into functions of no more than 33 lines each. Second, ent.xml is sometimes a little too pedantic for our
purposes, including some glyphs (e. g., the ‘fi’ and ‘fl’ ligatures) which we would rather not convert to HTML entities. It is also missing some other glyphs and common \TeX\ macros we would indeed like to convert — examples include the \TeX\ logo (\TeX) and the breakable slash (\slash). To remedy these last two problems we must edit \texttt{trans.bst} by hand; the first can then be rectified by a short shell script, \texttt{trans.bst}, which wraps consecutive sets of 33 lines in their own \texttt{bst} function definitions. The output of this script can then be inserted into a Biblet \texttt{bst} style.

**Unresolved issues** Biblet’s approach to building online publication lists is certainly entirely portable, though it does have its drawbacks, most of which stem from the limitations of Bib\TeX\ itself.

The first and most apparent problem is Biblet’s execution speed — using the interpreted \texttt{bst} language to perform extensive string manipulation. With \LaTeX\-to-HTML symbol conversion enabled, running Biblet on a \texttt{bib} file with only a few dozen publications can take several minutes even on a reasonably fast (1.4 GHz) machine. By sacrificing portability, this problem could be solved by writing the symbol conversion routine in an interpreted text-processing language such as Sed [15] or Perl [83], or as a compiled lexical analyzer using Lex and C [39, 46].

A second problem — actually a class of related problems — is Biblet’s extensibility. The root of this problem is that there is no way to pass to Bib\TeX\ any information besides the \texttt{bst} and \texttt{bib} files to use. Thus there is no way to specify the title of the HTML document produced by Biblet; likewise the user cannot tell Biblet to include a link back to his home page at the end of the bibliography. To effect such changes, the user must either edit the HTML output by Biblet, or edit the \texttt{bst} styles himself — either is a potentially daunting task for someone not familiar with HTML or the \texttt{bst} language.

The rigid syntax of \texttt{bib} files also poses a problem for Biblet’s extensibility. While users and style developers are free to create new publication types and fields, some applications require an extra level of specification that Bib\TeX\ simply does not support. An example of such an application is found in Biblet’s hyperlink fields. Our \texttt{pdf} field, for example, specifies the URL of a PDF file; Biblet might convert the value of this field into an HTML hyperlink as follows:

```html
<a href="{url}"
   type="application/pdf"
   title="{(title)}">...</a>
```

The value of the \texttt{type} attribute, a MIME content type [20, 21], tells the browser what kind of file to expect when the user follows the link, in case such information is not provided by the server hosting the PDF file. Thus, when processing \texttt{html} and \texttt{ps} fields, Biblet substitutes the appropriate MIME type — in this case, \texttt{text/html} or \texttt{application/postscript}.

However, this mapping of fields to MIME types must be hard-coded in the Biblet \texttt{bst}. Should a user wish to provide a link to some other kind of file type — say, a Rich Text File (RTF) — he will have to edit the \texttt{bst} source. A better solution would be for Bib\TeX\ databases to support parameterized fields so that users could specify unusual MIME types for document links in the \texttt{bib} file itself. For example, if a user wanted to provide links to an HTML, an RTF, and a sound recording version of an article, he could specify the URLs as follows:

```json
@ARTICLE{m05toc,
   title = {The Tyranny of Copyright},
   ...
   url[type="text/html"] = {http://...},
   url[type="text/rtf"] = {http://...},
   url[type="audio/mp4"] = {http://...},
}
```

There are some extensions to and replacements for Bib\TeX\ which go some way towards solving these extensibility issues [14, 34, 84], though none of them are yet popular or stable enough to wholly supplant Bib\TeX\ with LaTeX. Patashnik himself has been planning to extend the “official” version of Bib\TeX\ to allow for better communication between Bib\TeX\ and its environment [58], so it is possible that Biblet’s extensibility problems may one day be solved without having to compromise its portability.

**Development status and availability**

At the time of this writing, Biblet is under active development, and while the interface is not yet stable, the program is nevertheless very usable. A preliminary version of Biblet is available for download at the project’s web page, http://www.nothingisreal.com/biblet/. Apart from this article, no formal documentation is yet available. By publication time a beta version of a complete Biblet package, including several \texttt{bst} and CSS styles and a user’s guide, may be available on CTAN.

**Bibliography**


[31] Eitan M. Gurari. \( \text{	extit{\LaTeX}} \)X4ht: \( \text{	extit{\LaTeX}} \)X and \( \text{	extit{\LaTeX}} \) for Hypertext, February 2005. URL http://www.cse.ohio-state.edu/~gurari/\textit{TeX4ht/}.


[34] Jean-Michel Hufflen. \( \text{	extit{\LaTeX}} \)\textit{\Bib}: a “\textit{\LaTeX} to HTML” translator, 3.40 edition. URL http://hutchinson.belmont.ma.us/\textit{th/}/manual/.


[45] \( \text{	extit{\LaTeX}} \)X3 Project. The \( \text{	extit{\LaTeX}} \)X Project Public License, version 1.3a, October 2004. URL http://www.latex-project.org/lppl/lppl-1-3a.html.


[54] Bruce R. Miller. *\textit{\LaTeX}X: An \textit{\LaTeX} to XML Converter; Preview Version 0.3.0*. URL http://dlmf.nist.gov/LaTeXML/LaTeXML.html.


[57] Oren Patashnik. *\textit{\Bib\textit{\TeX}}\textit{\Xing}* February 1988. Documentation for \textit{\Bib\textit{\TeX}}\textit{\X} 0.99b.


[60] Luca Previtali, Brenno Lucari, and Erik Wilde. \textit{\Bib\textit{\LaTeX}}\textit{\X}: An XML representation of \( \text{	extit{\LaTeX}} \).


Tristan Miller


Abstracts

Dynamic presentations using \TeX power and PSTricks
David M. Allen, University of Kentucky

The nature of the full version of this presentation requires that it be viewed on a screen rather than paper, as dynamic features cannot be illustrated on paper. An electronic version of the full paper is available from the author's web site: http://www.ms.uky.edu/~allen/.

A typical presentation consists of displaying a sequence of slides (a metaphor for screens) in a predetermined order. This presentation is to demonstrate methods for preparing dynamic presentations in the following contexts:

1. Rather than showing a set of slides in predetermined order, one may select the slides and their order after the presentation starts. This would likely be in response to questions from the audience.
2. If the discussion gets deep, it may be useful to visit a web site.
3. A math professor might want to show a multiline derivation one line at a time to focus attention to the current point of discussion.
4. An engineer might want to show a graphic depicting the assembling of a device one part at a time.

The \TeX packages used in this endeavor and their URL's follow.

Items 1 and 2 are implemented using the hyperref package, http://www.ctan.org/tex-archive/macros/latex/contrib/hyperref/. Extensive facilities for navigation within a document, between documents, and on the web are provided by hyperref.

Items 3 and 4 are implemented using \TeX power, http://texpower.sourceforge.net/. \TeX power is a \LaTeX package providing incremental display and special effects similar to those found in Microsoft PowerPoint.

Item 4 assumes there are graphics to be displayed, and my examples use graphics produced by the PSTricks package, http://www.pstricks.de/. PSTricks provides a user friendly front end to the PostScript language. It is a generic \TeX package providing extensive computational graphics capabilities.

Typesetting critical editions of poetry with poemscol
John Burt, Brandeis University

poemscol provides macros for \LaTeX for setting collections of poetry. It provides the structures required to produce a critical edition of the kind specified by the Modern Language Association’s Committee on Scholarly Editions, providing line numbering, endnote sections for textual variants (both substantives and accidentals), emendations, explanatory notes, and an index of titles and first lines. It provides running headers of the form “Emendations to pp. xx–yy” for the endnotes sections. It provides structures for different kinds of poetic text. It automatically marks every occasion where a stanza break falls on a page break. Aids for preparing parallel-text (as for instance editions with facing-page translations) editions are under development.

(Full papers on poemscol were published in TUGboat 22(4) and The Prac\TeX Journal 2005-3. Ed.)

Indexing, MakeIndex, and SAS
Ronald Fehd, CDC

\LaTeX provides the fancyverb package which can be very useful in preparing a document providing an overview of a collection of computer programs. This paper examines the theory of indexing and the \LaTeX MakeIndex package. The author provides two SAS programs which read all programs in a project directory and then write an index of intra- and inter-program references.

MathML via \TeX4ht and other tools
Eitan Gurari

The support provided by graphical browsers for the HTML standard was a major ingredient in developing the Internet into a popular medium for archiving and distributing general content. Two recent advancements suggest a similar bright future for mathematical content expressed with the MathML standard. The Mozilla Firefox browser, released last November, now offers native support for MathML. Also, the MathPlayer version 2 plug-in for MS Internet Explorer, which is easily installed and was released a year ago, is now capable of serving general MathML files.

This presentation will provide insight into how \TeX4ht produces MathML from \LaTeX sources, and will consider issues involved in creating MathML with \TeX4ht and other tools.
Abstracts

\textbf{\LaTeX{} and PitStop: An unusual but powerful alliance}  
Mirko Janc, INFORMS

I will share some experiences in preparing art files for inclusion in \LaTeX{} in the production cycle in our Institute. We publish 11 scholarly journals in Operations Research using \LaTeX{} with a special font setup (presented at the TUG 2003 conference in Hawaii).

Powerful \LaTeX{} math typesetting capabilities coupled with PitStop, a commercial Acrobat plug-in, enable easy relabeling of figures with most complex math. Unlike other methods, exact positioning and scaling is a breeze. We also use this same method for updating colored covers where color issues are at stake, so the underlying PDF template can be properly preserved.

Some other related “tricks” to get clean art ready for proper inclusion in \LaTeX{} will also be discussed.

\textbf{An introduction to Xe\TeX{}}  
Jonathan Kew

Professor Donald Knuth’s \TeX{} is a typesetting system with a wide user community, and a range of supporting packages and enhancements is available for many types of publishing work. However, it dates back to the 1980s and is tightly wedded to 8-bit character data and custom-encoded fonts, making it difficult to configure \TeX{} for many complex-script languages.

This paper will introduce Xe\TeX{}, a system that extends \TeX{} with direct support for modern OpenType and AAT fonts and the Unicode character set. This makes it possible to typeset almost any script and language with the same power and flexibility as \TeX{} has traditionally offered in the 8-bit, simple-script world of European languages. Xe\TeX{} (currently available on Mac OS X, but possibly on other platforms in the future) integrates the \TeX{} formatting engine with technologies from both the host operating system (Apple Type Services, CoreGraphics, QuickTime) and auxiliary libraries (ICU, TECkit), to provide a simple yet powerful system for multilingual and multiscr ipt typesetting.

The most significant extensions which Xe\TeX{} provides are its native support for the Unicode character set, replacing the myriad of 8-bit encodings traditionally used in \TeX{} with a single standard for both input text encoding and font access; and an extended \verb|\font| command that provides direct access by name to all the fonts installed in the user’s computer. It also provides a mechanism to access many of the advanced layout features of modern fonts.

Additional features that will also be discussed include built-in support for a wide variety of graphic file formats, and an extended line-breaking mechanism that supports Asian languages such as Chinese or Thai that are written without word spaces.

Finally, we look briefly at some user-contributed packages that help integrate the features of Xe\TeX{} with the established \TeX{} system. Will Robertson’s \verb|fontspec.sty| provides a simple, consistent user interface in \XeLaTeX{} for loading both AAT and OpenType fonts, and accessing virtually all of the advanced features these fonts offer; Ross Moore’s \verb|xunicode.sty| is a package that allows legacy \LaTeX{} documents to be typeset using native OS X fonts without converting the input text entirely to Unicode, by supporting traditional \TeX{} input conventions for accents and other ‘special’ (i.e., non-ASCII) characters.

(We expect to publish the full paper in the next issue of \textit{TUGboat}. Ed.)

\textbf{Producing beautiful slides with \LaTeX{}: An introduction to the HA-prosper package}  
Tristan Miller

In this paper, we present HA-prosper, a \LaTeX{} package for creating overhead slides. We describe the features of the package and give examples of their use. We also discuss what advantages there are to producing slides with \LaTeX{} versus the presentation software typically bundled with today’s office suites.

(The full paper on HA-prosper was published in \textit{The Prac\TeX{} Journal} 2005-2. Ed.)

\textbf{\TeX{} font installation and usage}  
Steve Peter

This talk is designed to be a near-comprehensive roadmap of installing and using fonts with \TeX{} (except for bitmapped fonts). We will start with the basics of \TeX{} font handling (TFMs, etc.), along with a discussion of the major font technologies (PostScript, TrueType, and OpenType) and \TeX{}’s virtual fonts. Then we move to NFSS and fontinst, followed by \TeX{}font and Con\TeX{}xt typescripts. Time permitting, we will configure an expert font, complete with fi, fl, ff, ffl, and ffl ligatures, suitable for professional typesetting.

(We expect to publish the full paper in a future issue of \textit{TUGboat}. Ed.)
Calendar

2005

Jun 6 – Jul 29  Rare Book School, University of Virginia, Charlottesville, Virginia. Many one-week courses on topics concerning typography, bookbinding, calligraphy, printing, electronic texts, and more. For information, visit http://www.virginia.edu/oldbooks.


Jun 14 – 17 Workshops and presentations on LaTeX, TeX, ConTeXt, and more. For information, visit http://www.tug.org/practicaltex2005/.


Jun 24 – 26 NTG 35th meeting, Terschelling, Netherlands. For information, visit http://www.ntg.nl/bijeen/bijeen35.html.

Jul 14 – 17 SHARP Conference (Society for the History of Authorship, Reading and Publishing). “Navigating Texts and Contexts”, Dalhousie University, Halifax, Canada. For information, visit http://sharpweb.org/ or http://www.dal.ca/~sharp05/.


Aug 1 – 5 Extreme Markup Languages 2005, Montréal, Québec. For information, visit http://www.extrememarxup.com/extreme/.

Aug 23 – 25 The 26th annual meeting of the TeX Users Group. For information, visit http://www.tug.org/tug2005/.


Status as of 15 August 2005

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 503 223-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Location</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep 29–30</td>
<td>DANTE, 33rd meeting, Christian-Albrechts-Universität zu Kiel, Germany</td>
<td>Kiel, Germany</td>
<td><a href="http://www.dante.de/events/">http://www.dante.de/events/</a></td>
</tr>
<tr>
<td>2006</td>
<td>Mar</td>
<td>DANTE, 34th meeting, Technische Universität Berlin, Germany</td>
<td><a href="http://www.dante.de/events/">http://www.dante.de/events/</a></td>
</tr>
<tr>
<td>Mar 6–10</td>
<td>Rare Book School, University of Virginia, Charlottesville, Virginia</td>
<td>Virginia</td>
<td><a href="http://www.virginia.edu/oldbooks">http://www.virginia.edu/oldbooks</a></td>
</tr>
<tr>
<td>Apr 6–8</td>
<td>“Jobbing printing — the stuff of life”, joint conference of the Printing Historical Society and The Ephemera Society, University of Reading, UK</td>
<td>Reading, UK</td>
<td><a href="http://www.printinghistoricalsociety.org.uk/events/">http://www.printinghistoricalsociety.org.uk/events/</a></td>
</tr>
</tbody>
</table>
NO MATTER IF YOUR DOCUMENTS ARE BIZARRE
smokin’ friendly
FORMAL
OR JUST A LITTLE
\[
\frac{\text{odd}}{2} \neq \left\lfloor \frac{\text{odd}}{2} \right\rfloor
\]
THEY’RE SPECIAL TO YOU
SO BRING THEM TO US

silmaril consultants
▷ \LaTeX\ typesetting
▷ XML document systems
http://www.silmaril.ie/

Entertainment  Tax Benefits
Health and Happiness  Creativity

Helping Buyers and Sellers Succeed

Joe Hogg, Broker-Associate
Seniors Real Estate Specialist; Realtor
\begin{center}
\textbf{RE/MAX TRI-CITY}, Los Angeles, CA 90027
\end{center}
voice: (323) 842-9764
e-mail: joe@joehogg.com
web: http://www.joehogg.com

Building Equity

Carleton Production Centre

\textbf{HUMANITIES TYPESETTING}
Specialising in Linguistics
Since 1991

613-823-3630  •  15 Wiltshire Circle
Nepean, Ont., Canada  •  K2J 4K9
Nature trusts us…
You can too!

Nature Physics is a new journal. Like others from Nature Publishing Group, it is set to become the leader in its field. And it’ll be typeset using \TeX{}. And of course River Valley were chosen for the task. Needless to say, the final PDF will come from XML with zero intervention. See why NPG, IOP and Elsevier trust us with their toughest journals. Perhaps we can breathe new life into your journals.

www.river-valley.com

Math and Science Equations 40% Faster!

MathType for Windows and Macintosh

MathType is the full-featured, professional version of the Equation Editor in Microsoft Office. It has hundreds of extras that you do not get with Equation Editor, and installs special features into Microsoft Word/PowerPoint, that effectively creates an integrated math word processor, slide creator, and web page editor.

Get the job done faster!

Tests have proven that using MathType gets the same job done 40% faster than using Equation Editor.

Download a fully-functional 30-day trial of MathType today!

www.dessci.com

\textbf{MathType}  The best thing for writing equations since chalk™
\textcopyright{}  MathType™  WebEQ™  MathPlayer™  MathFlow™

Design Science, Inc.  4028 Broadway, Long Beach, CA 90803, USA
www.dessci.com  Telephone: 562-433-0685, Fax: 562-433-6969, Email:sales@dessci.com

MathType and its logo are trademarks of Design Science. All other company and product names are trademarks and/or registered trademarks of their respective owners.

\textbf{The Lament Companion} has long been the essential resource for anyone using \LaTeX{} to create high-quality printed documents. This completely updated edition brings you all the latest information about \LaTeX{} and the vast range of add-on packages now available—over 200 are covered. Like its predecessor, \textit{The Lament Companion, Second Edition} is an indispensable reference for anyone wishing to use \LaTeX{} productively.

\textbf{Available at fine bookstores everywhere.}

For more information, visit: www.awprofessional.com/titles/0201362996

Frank Mittelbach and Michel Goossens
with Johannes Braams, David Carlisle, and Chris Rowley

\textbf{ISBN: 0-201-36299-6}

Frank Mittelbach and Michel Goossens
with Johannes Braams, David Carlisle, and Chris Rowley

The Lament Companion, Second Edition

The Lament Companion, Second Edition
Specializing in foreign language, linguistic, and technical typesetting using TeX, \LaTeX{}, and Con\TeX t, I have typeset books for Oxford University Press, Routledge, and Kluwer, and have helped numerous authors turn rough manuscripts, some with dozens of languages, into beautiful camera-ready copy.

With a background in linguistics and information technology, I have extensive experience in planning, editing, proofreading, and writing documentation, to offer you a complete solution.

I am an award-winning teacher and offer training in all matters TeX for individuals, groups, and corporations. Training is targeted to the specific needs of the client.

Phone: +1 732 287-5392  
Email: speter@dandy.net

Do you need on-site training for \LaTeX{}?

Contact Cheryl Ponchin at  
cponchin@comcast.net

Training will be customized for your company needs.

Any level, from Beginning to Advanced.
Institutional Members

American Mathematical Society, Providence, Rhode Island
Banca d’Italia, Roma, Italy
Center for Computing Science, Bowie, Maryland
Certicom Corp., Mississauga, Ontario Canada
CNRS - IDRIS, Orsay, France
CSTUG, Praha, Czech Republic
Florida State University, School of Computational Science and Information Technology, Tallahassee, Florida
IBM Corporation, T J Watson Research Center, Yorktown, New York
Institute for Defense Analyses, Center for Communications Research, Princeton, New Jersey
MacKichan Software, Washington/New Mexico, USA
Masaryk University, Faculty of Informatics, Brno, Czechoslovakia
New York University, Academic Computing Facility, New York, New York
Princeton University, Department of Mathematics, Princeton, New Jersey
Springer-Verlag Heidelberg, Heidelberg, Germany
Stanford Linear Accelerator Center (SLAC), Stanford, California

Stanford University, Computer Science Department, Stanford, California
Stockholm University, Department of Mathematics, Stockholm, Sweden
University College, Cork, Computer Centre, Cork, Ireland
University of Delaware, Computing and Network Services, Newark, Delaware
Université Laval, Ste-Foy, Québec, Canada
University of Oslo, Institute of Informatics, Blindern, Oslo, Norway
Uppsala University, Uppsala, Sweden
Vanderbilt University, Nashville, Tennessee

Tex Consultants

Ogawa, Arthur
40453 Cherokee Oaks Drive
Three Rivers, CA 93271-9743
(209) 561-4585
Email: arthur.ogawa@teleport.com
Bookbuilding services, including design, copyedit, art, and composition; color is my speciality. Custom \TeX macros and \LaTeX{} packages document classes and packages. Instruction, support, and consulting for workgroups and authors. Application development in \LaTeX, \TeX, SGML, PostScript, Java, and C++. Database and corporate publishing. Extensive references.

Veytsman, Boris
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: boris@lk.net
I provide training, consulting, software design and implementation for Unix, Perl, SQL, \TeX{}, and \LaTeX{}. I have authored several popular packages for \LaTeX{} and \texttt{latex2html}. I have contributed to several web-based projects for generating and typesetting reports. For more information please visit my web page: http://users.lk.net/~borisv.

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

The TUG office mentions the consultants listed here to people seeking \TeX{} workers. If you’d like to be included, or place a larger ad in TUGboat, please contact the office or see our web pages:

\TeX{} Users Group
1466 NW Naito Parkway, Suite 3141
Portland, OR 97208-2311, U.S.A.

Phone: +1 503 223-9994
Fax: +1 503 223-3960
Email: office@tug.org
Web: http://tug.org/consultants.html
http://tug.org/TUGboat/advertising.html
Introductory

3 Barbara Beeton / Editorial comments
  • typography and TUGboat news
5 Barbara Beeton / Hyphenation exception log
  • update to missed and incorrect U.S. English hyphenations
3 Karl Berry / From the president
  • TUG activities and information for 2005
52 Peter Flom / A \LaTeX{} fledgling struggles to take flight
  • incentives and barriers to learning \LaTeX{}
31 Peter Flom and Tristan Miller / Impressions of Prac\TeX{}'05
  • comments on the conference from attendees
66 Peter Flynn / \LaTeX{} on the Web
  • review of \LaTeX{} interactions with the Web
59 Klaus Hoppner / Strategies for including graphics in \LaTeX{} documents
  • introduction to common graphics formats and tools
63 Joseph Hogg / Making a booklet
  • notes on useful packages and processes for booklet production
68 Andrew Mertz and William Slough / Beamer by example
  • producing slide presentations via a series of examples
56 Anita Schwartz / The art of \LaTeX{} problem solving
  • summary of strategies and resources for debugging \LaTeX{}

Intermediate

74 Kaveh Bazargan / Batch Commander: A graphical user interface for \TeX{}
  • interactive parameter setting and instant feedback
81 David Ignat / Word to \LaTeX{} for a large, multi-author scientific paper
  • conversion from Word and Perl manipulation of references
97 Abstracts (Allen, Burt, Fehd, Gurari, Janc, Kew, Peter)

Intermediate Plus

85 Tristan Miller / Biblet: A portable Bib\TeX{} bibliography style for generating highly customizable XHTML
  • making Web pages from bibliographies in pure Bib\TeX{}
17 Tristan Miller / Using the RPM package manager for (\LaTeX{}) packages
  • using and building RPM packages for \TeX{}

Advanced

33 Nelson Beebe / The design of \TeX{} and METAPOST: A retrospective
  • historical review of the \TeX{} system software design and development
7 Pedro Quaresma / Stacks in \TeX{}
  • stacks, for package writers, and commutative diagrams
10 Denis Roegel / Kissing circles: A French romance in MetaPost
  • MetaPost construction of the “Apollonian gasket” of tangent circles

Reports and notices

99 Calendar
104 Institutional members
29 Practical \TeX{} 2005 conference information

Advertisements

101 Carleton Production Centre
102 Design Science, Inc.
101 Joe Hogg
102 The \LaTeX{} Companion, 2nd edition, by Frank Mittelbach et al.
103 MacKichan Software, Inc.
102 Personal \TeX{}, Inc.
103 Steve Peter
102 Cheryl Ponchin Training
102 River Valley Technologies
101 Silmaril Consultants
104 \TeX{} consulting and production services
General Delivery 3 Karl Berry / From the president
3 Barbara Beeton / Editorial comments
   Old TUGboat issues go electronic; CTAN announcement archives;
   Another \LaTeX manual— for word processor users;
   Create your own alphabet; Type design exhibition “Letras Latinas”;
   The cost of a bad proofreader;
   Looking at the same text in different ways: CSS on the web;
   Some comments on mathematical typesetting
5 Barbara Beeton / Hyphenation exception log

\LaTeX 7 Pedro Quaresma / Stacks in \TeX
Graphics 10 Denis Roegel / Kissing circles: A French romance in MetaPost
Software & Tools 17 Tristan Miller / Using the RPM package manager for (\La)\TeX packages

Practical \TeX 2005 29 Conference program, delegates, and sponsors
31 Peter Flom and Tristan Miller / Impressions from Prac\TeX’05
Keynote 33 Nelson Beebe / The design of \TeX and METAFont: A retrospective
Talks 52 Peter Flom / A \LaTeX fledgling struggles to take flight
56 Anita Schwartz / The art of \LaTeX problem solving
59 Klaus Höppner / Strategies for including graphics in \LaTeX documents
63 Joseph Hogg / Making a booklet
66 Peter Flynn / \LaTeX on the Web
68 Andrew Mertz and William Slough / Beamer by example
74 Kaveh Bazargan / Batch Commander: A graphical user interface for \TeX
81 David Ignat / Word to \LaTeX for a large, multi-author scientific paper
85 Tristan Miller / Biblet: A portable Bib\TeX bibliography style for generating
   highly customizable XHTML
97 Abstracts (Allen, Burt, Fehd, Gurari, Janc, Kew, Peter)

News 99 Calendar
TUG Business 104 Institutional members
Advertisements 104 \TeX consulting and production services
101 Silmaril Consultants
101 Joe Hogg
101 Carleton Production Centre
102 Personal \TeX, Inc.
102 River Valley Technologies
102 Design Science, Inc.
102 The \LaTeX Companion, 2nd edition, by Frank Mittelbach et al.
103 Steve Peter
103 Cheryl Ponchin Training
103 MacKichan Software, Inc.
Introductory

3  Barbara Beeton / Editorial comments
   • typography and TUGboat news
5  Barbara Beeton / Hyphenation exception log
   • update to missed and incorrect U.S. English hyphenations
3  Karl Berry / From the president
   • TUG activities and information for 2005
52 Peter Flom / A \LaTeX{} fledgling struggles to take flight
   • incentives and barriers to learning \LaTeX{}
31 Peter Flom and Tristan Miller / Impressions of PracT\TeX{}’05
   • comments on the conference from attendees
66 Peter Flynn / \LaTeX{} on the Web
   • review of \LaTeX{} interactions with the Web
59 Klaus Höppner / Strategies for including graphics in \LaTeX{} documents
   • introduction to common graphics formats and tools
63 Joseph Hogg / Making a booklet
   • notes on useful packages and processes for booklet production
68 Andrew Mertz and William Slough / Beamer by example
   • producing slide presentations via a series of examples
56 Anita Schwartz / The art of \LaTeX{} problem solving
   • summary of strategies and resources for debugging \LaTeX{}

Intermediate

74  Kaveh Bazargan / Batch Commander: A graphical user interface for \TeX{}
    • interactive parameter setting and instant feedback
81  David Ignat / Word to \LaTeX{} for a large, multi-author scientific paper
    • conversion from Word and Perl manipulation of references
97  Abstracts (Allen, Burt, Fehd, Gurari, Janc, Kew, Peter)

Intermediate Plus

85  Tristan Miller / Biblet: A portable \BibTeX{} bibliography style for generating
    highly customizable XHTML
    • making Web pages from bibliographies in pure \BibTeX{}
17  Tristan Miller / Using the RPM package manager for (\LaTeX{})\TeX{} packages
    • using and building RPM packages for \TeX{}

Advanced

33  Nelson Beebe / The design of \TeX\ and METAFONT: A retrospective
    • historical review of the \TeX\ system software design and development
7  Pedro Quaresma / Stacks in \TeX
    • stacks, for package writers, and commutative diagrams
10  Denis Roegel / Kissing circles: A French romance in MetaPost
    • MetaPost construction of the “Apollonian gasket” of tangent circles

Reports and notices

99  Calendar
104 Institutional members
29  Practical \TeX{} 2005 conference information

Advertisements

101 Carleton Production Centre
102 Design Science, Inc.
101 Joe Hogg
102 The \LaTeX{} Companion, 2nd edition, by Frank Mittelbach et al.
103 MacKichan Software, Inc.
102 Personal \TeX{}, Inc.
103 Steve Peter
102 Cheryl Ponchin Training
102 River Valley Technologies
101 Silmaril Consultants
104 \TeX{} consulting and production services