# TUGboat

Volume 27, Number 2 / 2006
TUG 2006 Conference Proceedings

**TeX Users Group**

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

**Memberships and Subscriptions**

2006 dues for individual members are as follows:

- Ordinary members: $85.
- 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.

Also, (non-voting) *TUGboat* subscriptions are available to organizations and others wishing to receive *TUGboat* in a name other than that of an individual. The subscription rate is $95 per year, including air mail delivery.

**Institutional Membership**

Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

TeX is a trademark of the American Mathematical Society.

Printed in U.S.A.

**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**
+1 206 203-3960

**Electronic Mail**
(Internet)

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

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

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

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

**World Wide Web**
`http://www.tug.org/`
`http://www.tug.org/TUGboat/`

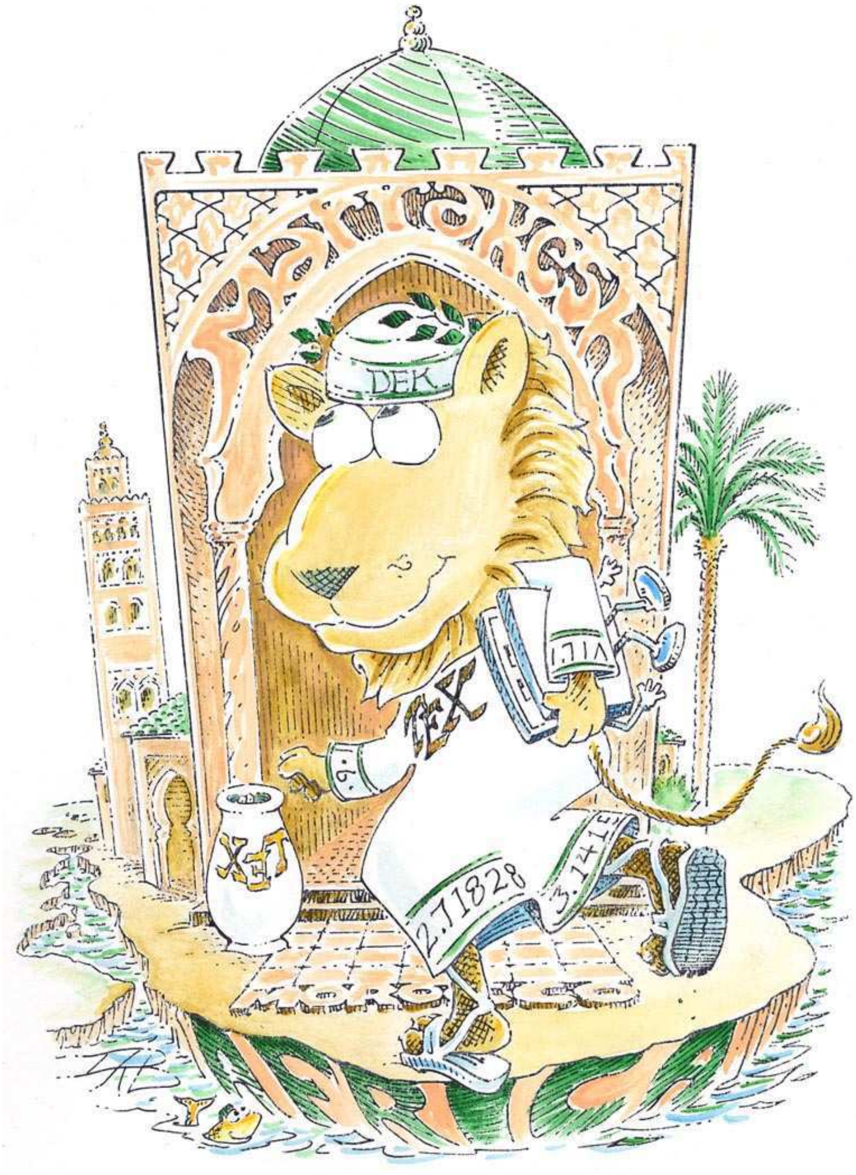
**Have a suggestion? Problems not resolved?**

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: December 2006]

# TUGboat

The Communications of the TEX Users Group



Volume 27, Number 2, 2006
TUG 2006 Conference Proceedings

## TeX Users Group

_TUGboat_ (ISSN 0896-3207) is published by the TeX Users Group.

### Memberships and Subscriptions

2006 dues for individual members are as follows:

- Ordinary members: $85.
- 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.

Also, (non-voting) _TUGboat_ subscriptions are available to organizations and others wishing to receive _TUGboat_ in a name other than that of an individual. The subscription rate is $95 per year, including air mail delivery.

### Institutional Membership

Institutional membership is a means of showing continuing interest in and support for both TeX and the TeX Users Group, as well as providing a discounted group rate and other benefits. For further information, contact the TUG office or see our web site.

Printed in U.S.A.

### Board of Directors

See `http://tug.org/board.html` for past board members.

### 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

+1 206 203-3960

### Electronic Mail

(Internet)

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

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

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

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

### World Wide Web

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

### Have a suggestion? Problems not resolved?

The TUG Board wants to hear from you:
Please email `board@tug.org`.

[printing date: December 2006]

## Table of Contents (ordered by difficulty)

### Introductory

### Intermediate

### Intermediate Plus

### Advanced

### Contents of other TeX journals

### Reports and notices

# TUGboat

Volume 27, Number 2 / 2006
TUG 2006 Conference Proceedings

**TUG 2006 Proceedings**

Marrakesh, Morocco
November 9–11, 2006

# TUGboat

COMMUNICATIONS OF THE TEX USERS GROUP

TUGboat Editor Barbara Beeton
Proceedings Editors Azzeddine Lazrek
Karl Berry

Volume 27, Number 2 • 2006
Portland • Oregon • U.S.A.

# General Delivery

## From the President

Karl Berry

As I write this at the end of November 2006, perhaps the largest outstanding project is the next edition of TEX Live; work is ongoing, and we hope it will be completed by the end of the year. If you can help build, document, test, or assist in any other area, please see `http://tug.org/texlive`. Meanwhile, other recent news follows.

### The Utopia fonts freely available (again)

As old-timers may recall, Adobe made the Utopia font family freely available many years ago. Unfortunately, the precise wording of the original license was ambiguous as to whether modified versions could be redistributed. I am happy to report that Adobe has now clarified the wording in a slightly revised agreement with TUG; the fonts were always intended to be free (as in freedom). So Utopia will reappear in the next edition of TEX Live, and are already included in current MiKTEX and gwTEX updates. The Utopia family has been extended by several other packages, such as vnTEX and fourier-GUT, so this is especially welcome news.

Major thanks go to Terry O'Donnell at Adobe for having the patience to shepherd this through the management there. The specific wording and other details, as well as the fonts themselves (which are entirely unchanged from the original release), are available at `http://tug.org/fonts/utopia`, and from CTAN in `fonts/utopia`.

### Colorado State University grant

Another bit of good news: Colorado State University has awarded a grant of over $40,000 for a new incarnation of pdfTEX, which will include support for Arabic typesetting and OpenType as well as the Lua embedded language.

The implementation effort is being led by Taco Hoekwater, in conjunction with the MetaPost and pdfTEX teams. Professor Idris Samawi Hamid of CSU (Philosophy Department) was the instigator of the grant, and we thank Dr. Hamid and his institution. Please find more information in a separate editorial in this issue, and in the previous issue of *TUGboat*. Taco's recent installment in the Interview Corner feature of the TUG web site, `http://tug.org/interviews` includes an overview of these projects.

### Further TUG joint memberships

TUG now offers joint memberships with DANTE e.V. and DK-TUG; thanks to their respective memberships for supporting this, and to Klaus Höppner and Kaja Christiansen for leading the respective efforts.

These new joint memberships are essentially the same as the agreements with NTG and UK-TUG which go back many years. Links to the joint membership pages are at `http://tug.org/join.html`.

We are very happy in general to be part of a new level of co-operation among all the TEX user groups. We have been working closely together both administratively, on conference planning and publications as well as these joint memberships, and technically, with the large TEX Collection releases, the TEX Gyre font (see the article by Hans Hagen et al. elsewhere in this issue), and other projects.

### Board addition and election

Finally, I am happy to announce that Jon Breitenbucher has joined the TUG board. Jon joins us from the College of Wooster in Ohio, where he has been working to spread TEX usage among students and staff. His report will soon be published in the Practical TEX 2006 proceedings issue of *TUGboat*.

If you are interested in running for the TUG board or president, 2007 will be a TUG election year. Nominations for these openings are now invited; the deadline to receive nominations is 1 February 2007. Please see `http://tug.org/election` for information and forms. (Since a few people have asked — barring unforeseen events, I expect to run for another term as president.)

Thank you all for supporting TEX and TUG.

◇ Karl Berry
`president@tug.org`

## Editorial Comments

Barbara Beeton

### TUG 2006

TUG 2006 was a great success, as you can read from Taco's report later in this issue. Marrakesh was a whole new experience for me, my first visit to the African continent; the sounds and smells were, in varying degrees, familiar (I'm always willing to try a new cuisine, and am a long-time devotee of international folk music), but the sights were new, and certainly so were the customs. I wasn't really prepared for such a garden spot in the middle of what I expected to be semi-desert.

I brought home many memories — and a determination to learn how to make mint tea. I learned that a cape I've owned and loved for years is indeed a genuine Tuareg garment; we saw one in the museum that was woven to the same pattern. And I discovered, no surprise, that bargaining with merchants is something better learned when young; alas, I'm not ready to be taught that new trick.

And it was good to see so many old friends, and make some new ones as well. Although the number of participants was not so large, the enthusiasm evoked by a spirited discussion of how to extend TEX to generate documents of superb calligraphic quality was as high as I've ever encountered at a TUG meeting.

I'd like to extend my personal thanks publicly to Azzeddine Lazrek for his hospitality, making the experience so enjoyable for my husband and myself. Thanks, Azzeddine!

### Chuck Bigelow goes to RIT

Charles Bigelow has been named as the next Melbert B. Cary Professor at the Rochester Institute of Technology. This prestigious chair was previously held by Alexander Lawson and Hermann Zapf.

Chuck had been contemplating teaching again, so this opportunity came at a very auspicious time. He plans first to offer a course on the history, theory, and technology of typefaces and fonts, followed by an advanced course on newspaper typography (not only types and fonts, but also usage and the typographic image and identity of a newspaper) and later, advanced typography seminars on other topics, such as how typography contributes to the experience of super markets, chain stores, and the rest of our marketed civilization.

Together with Don Knuth, Chuck established a master's program in digital typography at Stanford in 1982, and has taught and lectured extensively. Together with Kris Holmes, he created the Lucida family, one of the first fonts that optimized typography for output on personal computers and lower resolution printers. Bigelow & Holmes are also responsible for some of the first TrueType fonts, including many of Apple's "city" fonts.

Best wishes to Chuck in his new position. And thanks to Jill Bell, who provided much of this information to the TYPO-L discussion list.

### DEK in the news again

A delightful illustrated profile appeared in *The Stanford Magazine* last spring, recounting what Don Knuth is up to these days.

Read it at `http://www.stanfordalumni.org/news/magazine/2006/mayjun/features/knuth.html`

### Corrigendum: *TUGboat* 21:2

In an article on Thai fonts by Werner Lemberg (**21**:2 (2000), pages 113–120), a reader fluent in Thai encountered an example that contained a serious typo. The error has been corrected in the version posted on the TUG web site. Thanks to Werner for assistance in patching this up.

### Out-of-copyright books on the Web

As of the end of August, the Google Book Search makes it possible for readers to download out-of-copyright books to be read at your own pace. The collection is diverse, including well-known classics as well as more obscure gems.

Go to `books.google.com` and select the "Full view" radio button to find out what books can be downloaded. The collection continues to grow as more and more of the world's books are digitized.

This site also contains many books which are still protected by copyright. These books can be searched as well, but only a few sentences surrounding your search term will be displayed, just enough to enable you to determine whether you've found what you're looking for.

You may also be interested in Project Gutenberg, `http://www.gutenberg.org`, a longstanding effort which has made over 19,000 books with expired copyrights freely available.

Happy reading.

◇ Barbara Beeton
American Mathematical Society
201 Charles Street
Providence, RI 02904 USA
`bnb (at) ams dot org`

# Fonts

## The installation and use of OpenType fonts in LaTeX

John D. Owens

### Abstract

The emerging file standard in digital typography is the OpenType font standard, jointly developed by Microsoft and Adobe. OpenType fonts are natively supported by several popular operating systems and have many features and advantages that make them desirable for high-quality typography. However, OpenType fonts are not natively supported by the standard TeX engine. This article is a practical guide to installing OpenType fonts for use as text fonts in LaTeX.

The steps to install an OpenType font for use in LaTeX are:

1. For each OpenType font file, and for each combination of attributes for that font file, generate and install font metric and encoding files.
2. Next, for each font family, generate and install a font description (.fd) file that maps LaTeX font selection commands to the installed font files.
3. Finally, write and install a style (.sty) file that allows the user to select the font and its options for use within TeX.

We begin by discussing font background, the TeX font handling scheme, and existing font tools, then describe each of the three steps above in detail.

### 1 Font basics and font families

The advanced typographic features of the OpenType font format have motivated its widespread use in a variety of demanding applications. Before we dive into supporting OpenType in TeX, however, let's take a step back and look at our eventual goal. As a TeX user, we are less interested in using just a single font with a single set of options and more interested in using a *font family*: a collection of compatible font variants, usually from the same typeface, that can be used together. For instance, we might want to group together a plain and an italic form of a particular typeface into a family. We might want to make small caps available in our family as well, or perhaps incorporate several different weights or optical sizes. Once we have defined our font family, we would then like to ask TeX to enable the entire family with a single command. As an example, this document is typeset in an Adobe OpenType Minion Pro font family with old-style figures, with code segments typeset in Adobe's OpenType Myriad Pro, using the following LaTeX commands:

```
\usepackage{minion}
\usepackage[tt,sf,lining,scale=0.92]{myriad}
```

| In different files | Within one OpenType file |
|---|---|
| weight (light, **black**) | kerning (VAVAV vs. VAVAV) |
| width (abc vs. abc) | ligatures (fi vs. fi) |
| optical size | figure style (1234 vs. 1234) |
| variant (e.g. *italics*) | SMALL CAPS |

**Table 1**: Font features provided by different OpenType files (left) and within a single OpenType file (right).

What are the different typeface alternatives that can be part of our font family? (Gelderman provides an introduction to typeface characteristics [3].) We can group possible alternatives into several broad categories, and then indicate how OpenType handles each category.

Our first four categories are weight, width, optical size, and variant. The *weight* of a typeface refers to the thickness of the strokes that constitute its glyphs. A font designer may also vary a typeface's *width* relative to its height. The major advantage of vector font formats (such as OpenType, PostScript Type 1, and TrueType) is their ability to be scaled to any size. However, type designers have found that the most visually appealing fonts are ones that are designed for a particular size range, called an *optical size*. Finally, the standard upright "roman" style for fonts is not the only possible style. Font users may also desire italic or oblique or outline forms of a particular typeface, which together are termed *variants*. In LaTeX's New Font Selection Scheme (NFSS), the combination of weight and width is called *series* and the variant is called *shape*; we use this terminology in Section 4.2.

In OpenType, each unique combination of weight, width, optical size, and variant is associated with a separate font file. As an example, the Adobe Kepler typeface has several alternatives in each of these categories. Kepler features six weights (light, regular, medium, semibold, bold, and black), each with four widths (condensed, semicondensed, regular, and extended). Most of Kepler's combinations of weight and width feature four optical sizes (from smallest to largest, caption, regular, subhead, and display), and each weight-width-optical-size combination has both an upright and italic variant. Thus it is little wonder that Kepler's many combinations require 168 different OpenType files. And after we catch our breath to consider all the typographical options already available to us, we dive into a single OpenType file to find still more options.

In addition to the categories that require different files, the OpenType font format also allows a single font

Editor's note: Due to the nature of this article, it is typeset in the Adobe Minion and Adobe Myriad typefaces. We thank Adobe for permission to use these fonts in both the print and web publications.

file to specify a variety of other features. Not all of these features are currently supported in TEX, but many of them are. For instance, the *kerning* feature adjusts the spaces between pairs of glyphs. Enabling *ligatures* replaces pairs of glyphs like 'f' and 'i' with a single 'fi' glyph. Besides kerning and ligatures, the two classes of OpenType features that we cover in this article are the choice of figure styles (for example, old-style [01234] vs. lining [01234]) and Small Caps.

Table 1 summarizes the features provided by different OpenType files and within an OpenType file. With this overview of the many typeface alternatives that we would like to assemble into families, we can turn to how TEX interacts with fonts.

## 2 TEX font handling

In modern operating systems such as Microsoft Windows and Apple Mac OS X, applications that use OpenType fonts can read font information directly from the OpenType file. TEX, on the other hand, stores font information in a variety of files, and the complexity of creating and installing these files is the major reason that font handling has traditionally been a tricky task in TEX.

The OpenType font format can contain font data in either of two formats, Adobe PostScript Type 1 or TrueType. In this section we describe the font files that are associated with Type 1 and Type-1-flavored OpenType fonts. To support a Type 1 font, TEX requires the following files, each with its own function, with file locations specified by the TEX Directory Structure standard. More detailed descriptions of these formats can be found in Alan Hoenig's *TEX Unbound* [5] and Chapter 7 of the second edition of *The LATEX Companion* [11].

**TFM** "TEX Font Metrics" (tfm) files describe the dimensions of each character (glyph), along with a few font-wide parameter, which together are used by TEX to perform layout.

**PFB** For Type 1 fonts, "Printer Font Binary" (pfb) files contain Adobe PostScript Type 1 procedures that describe the shape of each glyph. These procedures are included by the output driver (for example, the dvips or pdftex program) in the output file (for example, PostScript or PDF).

**VF** "Virtual Font" files provide a mapping between the glyphs in the tfm files and the glyph order used by TEX (which is, in turn, specified with the encoding file, below). They are not needed for all fonts.

**ENC** Encoding files specify an ordering of glyphs called the "font-encoding vector". While typeset documents in English might require only the glyphs in TEX's default "OT1" font-encoding vector (used by Computer Modern Roman, for example), other languages or scripts need more or different glyphs.

In this article, we use the "LY1" encoding, an alternative to OT1 developed by Y&Y that is well-suited for Type-1-flavored fonts. (Among other advantages, the LY1 encoding maps directly to Adobe's font encoding and thus requires no virtual fonts.)

**MAP** Finally, the map files tie the above files together. map files (and map file formats) are specific to output drivers and associate tfm and Type 1 font names with pfb files, which contain the shapes of glyphs in those fonts.

Only when these files have been properly installed for a particular font, and system databases updated, can TEX then typeset glyphs from that font in a document.

Writing all these files by hand would be both tedious and error-prone, so two excellent pieces of software have automated the font installation process.

- fontinst [6], by Alan Jeffrey, Rowland McDonnell, and Lars Hellström, automates the installation of PostScript Type 1 fonts into TEX. Philipp Lehman's font installation guide [9] is an outstanding tutorial for fontinst.

  However, fontinst does not offer access to OpenType features. Also, fontinst scripts are written in TEX, and are challenging for non-experts to write and use.
- Eddie Kohler's otftotfm [8], part of his LCDF Typetools suite, creates and installs the required TEX files (tfm, pfb, vf, enc, and map) from OpenType font files. Note that otftotfm generates PostScript Type 1 fonts from OpenType; please ensure that the legal license for your fonts allows such a format conversion. otftotfm is a command-line tool that accepts a set of options and applies them to a single OpenType font file.

  Section 6 describes two tools built around otftotfm that both automate calls to otftotfm across multiple font options and also create the necessary TEX fd and sty support files.

In this article, we focus on otftotfm as the underlying tool that translates OpenType fonts into a TEX-readable form. We also focus on the procedure for setting up text fonts. The setup for math fonts requires additional commands described in Chapter 7.10.7 of *The LATEX Companion* [11]. The next section outlines how otftotfm installs OpenType fonts, and the remainder of the article describes how to extend otftotfm to handle multiple font files and font families and how to make the installed fonts available to TEX users.

## 3 OpenType to TEX

The first step in making OpenType fonts available to TEX users is to deposit the various font files into the TEX installation for each variant in the font family. We begin by showing how otftotfm installs a single font, using Adobe Minion Pro's Semibold Italic font as an example.

```
otftotfm -a -e texnansx -fonum -fkern -fliga \
  MinionPro-SemiboldIt.otf \
  LY1-MinionPro-SemiboldIt-onum
```

Let's analyze this example. -a is the magic "automatic" flag, automatically installing the relevant TEX font files from Section 2 (tfm, pfb, vf, enc, and map) into their proper locations within the TEX directories. -e texnansx specifies the encoding file for the LY1 encoding. Three OpenType features (old-style numerals, kerning, and ligatures) are requested with the -f flags, and the final two arguments are the names of the OpenType input font file and the output font name. The otftotfm manual explains these options in detail, and also enumerates available OpenType features [8].

Extending otftotfm to more input fonts and more variants is straightforward: simply call otftotfm for each and every combination of desired features. For complex variant combinations and fully featured font families, the number of calls to otftotfm can exceed many hundreds. The tools described in Section 6 automate this process.

LCDF's otfinfo tool [8] can identify the supported OpenType features for any OpenType font file, but which features are interesting for TEX users?

- The kerning (kern) and ligature (liga) features should always be turned on if available.
- OpenType fonts may support several kinds of numerals; onum (old-style numerals) and lnum (lining numerals) can both be supported in TEX and are commonly requested typographic features.
- Small Caps are enabled by the smcp feature.
- Superior (sups) and inferior (sinf) figures are useful for footnotes, inline fractions, and scientific typesetting; swashes (swsh) are more *DECORATIVE* alternatives to standard characters.

The otftotfm web page [8], in "otftotfm examples", contains examples of more advanced OpenType features, but as we note in Section 7, more advanced features rarely have high-level support in TEX or LATEX. In this article we concentrate on the overall installation procedure for OpenType fonts and support of more basic features; readers in need of more advanced features may consider the ConTEXt environment [10] or XƎTEX [7].

After otftotfm finishes installing all font files into TEX, texhash and updmap must be called to make TEX aware of the new installation. Now, the new fonts are available for typesetting in TEX — but how? The next section describes how to instruct TEX to *use* the correct font. TEX uses the "font description" file for this purpose.

## 4 Font description (fd) files

A font description file is a TEX source file that maps installed font file names to font attributes as used in (LA)TEX. Typically, each font family is described by a single fd file. As we previously mentioned, these techniques are applicable to text fonts; math fonts require additional commands [11].

Only two TEX commands are necessary in an fd file. The first declares a font family, and the second declares a specific font within that font family. We'll look at them one at a time.

### 4.1 \DeclareFontFamily

The \DeclareFontFamily command indicates that a certain font family is available in a certain encoding scheme. The names of encoding schemes are fixed (as mentioned before, we use LY1 in this paper), but the name of the font family is arbitrary. The most well-known naming scheme is Karl Berry's fontname scheme [1], which concatenates a unique three-letter code for each typeface with a one-letter suffix that indicates the font family (Section 4.4). (This naming scheme is required when using nfssext.sty, described in the next section.)

Let us continue with the Minion-Pro-with-old-style-numerals example; Minion is abbreviated pmn, and font families associated with old-style numerals are designated by j (more details about what constitutes a font family are in Section 4.4). The resulting command is:

```
\DeclareFontFamily{LY1}{pmnj}{}
```

The third argument to \DeclareFontFamily is less often used; it can contain special options for font loading and is explained in *The LATEX Companion* [11].

### 4.2 \DeclareFontShape

The \DeclareFontShape command associates a particular font with a particular combination of encoding, font family, series, and shape, a classification which we previously discussed in Section 1. To classify the particular Adobe Minion font we installed in Section 3, we invoke the following 6-argument command:

```
\DeclareFontShape{LY1}{pmnj}{sb}{it}{
  <-> LY1-MinionPro-SemiboldIt-onum}{}
```

The first four arguments are the classification; the fifth argument is the font file(s) associated with that classification; and the last argument is used in a similar way to the third argument of \DeclareFontFamily. This particular command associates the combination of LY1 encoding, Minion-with-old-style-numerals font family, semibold series (sb), and italic shape (it) with the installed font named LY1-MinionPro-SemiboldIt-onum. Note this font name is (necessarily) identical to the output name in the command we invoked in Section 3.

With only these two commands, you can specify a completely functional fd file. Three additional commands are useful, however, for a more fully featured family.

**Optical size variants** Now, what's the <-> symbol before the font name (in the above \DeclareFontShape example)? It's the size range and indicates the font sizes associated with that font name. <-> is actually a special case of the more general notation <n-m>, meaning "use this font only for point sizes greater than or equal to n and up to m". Removal of n or m removes the bound, so <-> indicates a match for all font sizes. With this notation, the extension to multiple font files for a particular combination at different sizes (necessary for optical size variants) is straightforward:

```
\DeclareFontShape{LY1}{pmnj}{sb}{it}{
  <6-8.4>  LY1-MinionPro-SemiboldItCapt-onum
  <8.4-13>  LY1-MinionPro-SemiboldIt-onum
  <13-19.9> LY1-MinionPro-SemiboldItSubh-onum
  <19.9-72> LY1-MinionPro-SemiboldItDisp-onum}{}
```

**Font substitution** What happens if you're missing a particular variant for a font family? The sub command allows the substitution of one variant for another. For instance, few font families feature a slanted (oblique) variant, so fd files often substitute italic for slanted if slanted is requested. The following command asks for any reference, at any size, to semibold-slanted in our font family to be satisfied instead by semibold-italic.

```
\DeclareFontShape{LY1}{pmnj}{sb}{sl}{
  <-> sub * pmnj/sb/it}{}
```

Besides substituting italic for slanted, substituting bold for bold-extended is also common, as in the example below for the normal (n) shape.

```
\DeclareFontShape{LY1}{pmnj}{bx}{n}{
  <-> sub * pmnj/b/n}{}
```

**Scaling** Finally, \DeclareFontShape permits a font to be automatically scaled through the size command,[1] which is invoked by placing the scaling factor in brackets between the size range and the filename. The example below instructs TeX to typeset Minion's semibold italic variant at 95% of its natural size.

```
\DeclareFontShape{LY1}{pmnj}{sb}{it}{
  <-> [0.95] LY1-MinionPro-SemiboldIt-onum}{}
```

### 4.3 Naming shape and series

The de facto standard for the abbreviation strings associated with shape and series in fd files is described by LaTeX's "New Font Selection Scheme" (NFSS) [15]. Any choice for shape and series abbreviation strings, including NFSS, must work together with the font selection commands in Section 5. Philipp Lehman's tutorial contains a fairly complete mapping between weight, width, and variant names and NFSS encodings [9]; we reproduce common encodings in Table 2.

| Weight | NFSS series |
|---|---|
| light | l |
| book | m |
| regular | m |
| medium | mb |
| demibold | db |
| semibold | sb |
| bold | b |
| black | eb |
| Width | NFSS series |
| condensed | c |
| narrow | n |
| semicondensed | sc |
| regular | — |
| semiextended | sx |
| extended | x |
| Variant | NFSS shape |
| normal (upright) | n |
| italic | it |
| slanted | sl |
| oblique | sl |
| outline | ol |
| small caps | sc |
| small caps + italic | si |

**Table 2**: A selection of NFSS codes for font weights, widths, and variants. From Lehman [9].

Lehman presents the following algorithms for generating the series and shape abbreviations used in \DeclareFontShape. First, the weight and width are combined to create "series". If both weight and width are "regular", the series is set to m; otherwise the series is set to the concatenation of the weight and width codes, ignoring "regular" if present. Creating the shape is also straightforward: if the variant is "regular", the shape is n, otherwise the shape is the concatenation of all variant codes, with the exception of small-caps and italics. This shape is instead designated si, and font selection using si is described in Section 5.3.2.

### 4.4 Font families

Some font features do not fit into the series-shape scheme. The most common of these features is numerical figure types, which may vary as lining (1234), old-style (1234), superior ($^{1234}$), inferior ($_{1234}$), and so on. To handle font selection with different styles of figures in TeX, typically,

[1] This is most common when two different typefaces that do not match in size are used together in a document; in the next section we expose this capability to the document author.

each type of figures generates its own font family. To generate the name of the font family, the three-letter font designation has a one-letter suffix appended to its 3-letter font name. Lining figures are associated with no suffix or with x, the "expert" suffix; old-style figures are j; superiors receive 1 and inferiors 0; and so on. Thus the Minion (pmn) font family with lining figures is pmnx; Minion with old-style figures is pmnj; and so on. Section 5.3.3 shows how to perform font selection with different font families.

## 5 Style files

At this point we have installed our fonts into TEX (Section 3) and categorized them by family, shape, and series (Section 4). The final step is to make those fonts available to the TEX document author as text fonts. The tools described in Section 6 automate the creation of the sty files that contain the commands in this section.

### 5.1 Selecting a font family

The default "roman" (text) font family is defined by the TEX variable \rmdefault. Redefining \rmdefault to another font family (as named by \DeclareFontFamily) resets the roman font family. For instance, the command below sets the current font family to our example font family, Adobe Minion with old-style figures.

```
\renewcommand*{\rmdefault}{pmnj}
```

In fact this is all we need to do to use our new font family. (Similarly, we set the default sans serif font family by setting the variable \sfdefault, and the typewriter family with \ttdefault.) However, rather than using one of these commands directly in TEX files, it's typical to instead wrap it in a style file and invoke \usepackage on that style file to perform this declaration. A minimal (but complete) style file called minion.sty for LATEX 2$\varepsilon$ that uses the LY1 encoding follows.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{minion}[Minion Pro OSF v0.99a 9/06]
\RequirePackage[LY1]{fontenc} % uses LY1 encoding
\renewcommand*{\rmdefault}{pmnj}
\endinput
```

### 5.2 Selecting between multiple font families

What if we'd like to use the same style file to support Minion font families with both old-style (pmnj) and lining (pmnx) figures? We use a package option to choose between the two font families:

```
\usepackage[oldstyle]{minion}
```

or

```
\usepackage[lining]{minion}
```

The extended style file that supports these options is:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{minion}[Minion Pro OSF v0.99b 9/06]
\RequirePackage[LY1]{fontenc} % uses LY1 encoding
\DeclareOption{lining}{\renewcommand*{%
 \rmdefault}{pmnx}}
\DeclareOption{oldstyle}{\renewcommand*{%
 \rmdefault}{pmnj}}
\ExecuteOptions{oldstyle}
\ProcessOptions*
\endinput
```

### 5.3 Selecting font variants

Now we know how to select a given font family, which may feature a large number of font weights, widths, and variants within it. Once we have selected a font family, how can we direct TEX to select from our many alternatives within that font family, such as boldface, italic, small-caps, and so on? The answer is to change the current series and shape.

While the low-level TEX commands (\fontseries and \fontshape) directly change the current series and shape, LATEX's higher-level commands are more commonly used. Most LATEX users know that \textbf selects boldface; LATEX implements this internally by setting the font series to \bfdefault, which is in turn defined as bx. Similarly, \textit (italics) utilizes italics by setting the font shape to \itdefault, defined as it. And \textsc (small caps) sets the shape to \scdefault, defined as sc.

We can use similar techniques to add more selection commands for more features that are not part of the LATEX core set of commands. Philipp Lehman's font installation guide is an excellent tutorial for this task [9]; it carefully constructs and explains a style file of NFSS extensions, nfssext.sty. We now take a closer look at how to support alternate weights and how nfssext.sty supports small caps with italics and switching between old-style and lining figures.

#### 5.3.1 Supporting alternate weights

By default, LATEX supports a bold (\textbf) weight command. Let's say we feel the default bold is a little too dark, and we'd like to use semibold-condensed instead. We can accomplish this with a single line in our sty file:

```
\renewcommand*{\bfdefault}{sbc}
```

And now we like semibold-condensed so much, we'd like to add it as a new command, \textsb.

```
\newcommand\sbdefault{sbc}
\DeclareRobustCommand\sbseries
    {\not@math@alphabet\sbseries\mathsb
     \fontseries\sbdefault\selectfont}
\DeclareTextFontCommand{\textsb}{\sbseries}
```

For simple features, declaring a default value, a RobustCommand, and a TextFontCommand suffice to make the feature available within TEX.

### 5.3.2 Supporting small caps with italics

Lehman points out that italics and small caps are both in the same "variant" category, so the built-in \textit and \textsc commands do not work harmoniously together. Barring changes to the core LaTeX font selection primitives, text set to both italic and small-caps would only preserve the innermost setting.

nfssext.sty remedies this problem by declaring an si shape, analogous to it and sc, and its associated selection commands:

```
\newcommand*{\sidefault}{si}
\DeclareRobustCommand{\sishape}{%
 \not@math@alphabet\sishape\relax
 \fontshape\sidefault\selectfont}
```

It then changes the italic and small-caps commands to check the current shape before setting the new shape. Only if the current shape is italic and the new shape is small-caps, or vice versa, does it set the new shape to si. (Recall that we assigned the si code to small-caps + italic variants in Section 4.3.)

The *result* is *properly nested* small-cap, italic text.
{\textsc{The \textit{result}} is \textit{properly
\textsc{nested}} small-cap, italic text.

### 5.3.3 Supporting old-style and lining figures

Section 5.2 showed how to choose old-style or lining figures by default. However, it may be useful to have one as the default and use the other via an explicit command. In nfssext.sty, the new commands \textos selects old-style figures and \textln lining figures. In this article, for instance, old-style is the default, so 1234\textln{1234} results in 12341234.

Because each style of figures is associated with a different font family, using an alternate figure style requires changing the family. nfssext.sty accomplishes this task as follows. Switching to lining figures for the font named abc first tries font family abcx then font family abc, using the TeX primitive \selectfont; switching to old-style figures switches to font family abcj. Fortunately this complexity is all hidden inside nfssext.sty.

## 6 Tools

For any OpenType font installation into TeX, the vital tool is otftotfm [8]. However, otftotfm only installs from a single OpenType font file with a single set of options, while users typically would like to install an entire family of OpenType font files with all available options. In addition, otftotfm does not address the problem of creating fd or sty files.

To address these issues, Marc Penninga wrote autoinst [13] and John Owens wrote otfinst [12], both of which wrap around otftotfm to install entire TeX font families. The two tools have far more similarities than differences and should suffice for most users; the authors' use of Perl (autoinst) or Python (otfinst) may make the difference for programmers familiar with one or the other.

Among the features supported by both tools are:

- A command-line interface that takes one or more OpenType font files as arguments;
- Installation of font families with the following features if present: roman and italic text; small-caps; lining, old-style, superior, and inferior figures; and swashes;
- nfssext.sty font selection commands;
- Support of optical sizes; and
- Auto-generation and installation of sty and fd files.

Some of the differences are that autoinst also supports numerators, denominators, upright swash, and titling text, and generates ornaments, while otfinst supports a scaling option at runtime. otfinst uses fontname naming, while autoinst is more verbose in its naming scheme. Finally, otfinst uses the metadata associated with each OpenType font to determine the font's characteristics, while autoinst extracts the characteristics from the font's filename.

### 6.1 Other tools

Geoffrey Washburn's otftofd [16] automates the process of creating fd files from OpenType fonts. Washburn indicates that it, like autoinst, is designed primarily for Adobe font conventions. otftofd is a shell script written in the Caml Shell and uses otftotfm.

The MinionPro TeX package [2] provides extensive TeX support files for Adobe Minion, including comprehensive options for figure types, encodings, ornaments, letterspaced small caps, and calligraphic, math, blackboard, and Greek fonts. The MinionPro distribution was built using otftotfm and thus contains all TeX support files without the need for the steps described in this article. MinionPro includes a package called fontaxes that extends (and partially replaces) NFSS's rigid classification of font attributes.

## 7 Conclusions

This article has described the steps necessary to use OpenType fonts in TeX: use otftotfm to install TeX font metric and encoding files; build a font description file for each font family; and build a style file for convenient font selection in a document.

OpenType handling in TeX is still far from ideal, however. Systems like XeTeX [7] use OpenType fonts natively with truly impressive results, and native OpenType support is slated for future versions of pdftex [4]. However, equally important are the other components of TeX that relate to font selection and invocation.

- NFSS is insufficient to elegantly describe the wide variety of available font attributes. The combination of weight and width into series is awkward, multiple variants may combine into a single shape, and features such as figure styles are not covered at all. The ideal font selection scheme not only includes all variants of a typeface but also allows simple, orthogonal selection of any set of typeface features, and transparent substitution when features are not present. New, flexible approaches such as fontspec in XeTeX [14] and MinionPro's fontaxes are encouraging steps toward such a scheme.

- Even within NFSS the codes are not standardized. Lehman's scheme appears to be widely used, however, which is encouraging.

- Even for simple features, font selection is wholly nonstandardized and non-orthogonal. Selection of alternate widths is not possible without low-level commands, the default italic and small caps commands do not work together because LaTeX's default handling of the "variant" category does not address multiple variants, and using non-standard but desirable selection commands such as \textln in shared files is discouraged because a default TeX installation does not support them.

- Finally, while many advanced OpenType features can be supported in TeX's font files, *invoking* those features with high-level commands is a much more troublesome task. For most features beyond the basic ones, TeX and LaTeX have no standardized support at the author level (or, in many cases, no support at all). Features like ornaments, contextual alternates, and discretionary ligatures would benefit from a discussion about how they can be invoked by the programmer in a standard way.

**Acknowledgements** Many thanks to Karl Berry for encouraging me to write this article and for his helpful suggestions along the way. Eddie Kohler's tools greatly ease the task of OpenType support in TeX, and I also thank Eddie for his prompt and thorough answers to my many questions about his tools. Karl Berry and Philipp Lehman were both quite helpful in understanding fontname and how it's used within the TeX world. The use of Philipp's nfssext.sty was vital in the development of otfinst. Thomas Phinney, Geraldine Wade, and Michael Duggan provided fonts for testing, and Thomas secured permission to use Adobe fonts for the article itself. Finally, thanks also to Nelson Beebe, Stephen Hartke, Oleg Katsitadze, Eddie Kohler, Marc Penninga, Will Robertson, and Michael Zedler for their thoughtful comments on the article during the review process.

## References

[1] Karl Berry. Filenames for fonts. *TUGboat*, 11(4):517–520, November 1990. http://www.tug.org/fontname.

[2] Achim Blumensath, Andreas Bühmann, and Michael Zedler. MinionPro, September 2005. http://www.ctan.org/tex-archive/fonts/minionpro/.

[3] Maarten Gelderman. A short introduction to font characteristics. *TUGboat*, 20(2):96–104, June 1999.

[4] Taco Hoekwater. Opening up the type. *TUGboat*, 27(1):16–17, 2006.

[5] Alan Hoenig. *TeX Unbound*. Oxford University Press, New York, NY, 1998.

[6] Alan Jeffrey, Rowland McDonnell, and Lars Hellström. fontinst: Font installation software for TeX, December 2004. http://www.tug.org/applications/fontinst/.

[7] Jonathan Kew. The XeTeX typesetting system, 2006. http://scripts.sil.org/xetex.

[8] Eddie Kohler. LCDF type software, 2006. http://www.lcdf.org/type/.

[9] Philipp Lehman. The font installation guide, December 2004. http://www.ctan.org/tex-archive/info/Type1fonts/fontinstallationguide/.

[10] Adam T. Lindsay. OpenType installation basics for ConTeXt. *The PracTeX Journal*, 2005(2), April 2005.

[11] Frank Mittelbach and Michel Goossens. *The LaTeX Companion*. Addison-Wesley, Boston, MA, second edition, 2004.

[12] John Owens. otfinst, 2006. http://www.ctan.org/tex-archive/fonts/utilities/otfinst/.

[13] Marc Penninga. fontools, 2006. http://www.ctan.org/tex-archive/fonts/utilities/fontools/.

[14] Will Robertson. Advanced font features with XeTeX—the fontspec package. *TUGboat*, 26(3):215–223, 2005.

[15] LaTeX3 Project Team. LaTeX2$\varepsilon$ font selection, June 1994. http://www.ctan.org/tex-archive/macros/latex/doc/fntguide.pdf.

[16] Geoffrey Washburn. otftofd, 2005. http://www.ctan.org/tex-archive/fonts/utilities/otftofd/.

◇ John D. Owens
Department of Electrical and
Computer Engineering
University of California
One Shields Avenue
Davis, CA 95616 USA
jowens (at) ece dot ucdavis dot edu
http://www.ece.ucdavis.edu/~jowens/

# Hints & Tricks

## Glisterings

Peter Wilson

> Remember still that the loftier minde
> That in this world doth seek to glister so,
> Blowne on this rock by fonde vainglorious winde,
> Falls headlong down to everlasting wo.
>
> *The Ship of safegarde*, (1569)
> BARNABE GOOGE

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two while hopefully not making things worse through any errors of mine.

Corrections, suggestions, and contributions will always be welcome.

> Addresses are given to us to conceal our whereabouts.
>
> *Reginald in Russia*,
> SAKI

### 1 Address lists

For many a year I have been promising my wife that I would print labels for the envelopes for our Christmas cards. I even went as far as buying some software to run on an OS that went out of date in the last century. This year (2005) I have at last salved this part of my conscience with the aid of LaTeX and Boris Veytsman's EnvLab package [1].

After some experiments I created a package file, myenvlab, that gave me the setup that I wanted. I found that I had to use the EnvLab's `\SetLabel` command to adjust the address spacing to match the sheets of labels that I was planning to use. I also organised things so that each label could be framed by an `\fbox` and I could then print a page of addresses onto an ordinary sheet of paper to check if the spacing was correct for the real label sheets. For further information consult the package documentation.

```
% file myenvlab.sty
\usepackage[avery5160label,
            noprintbarcodes,
            nocapaddress]{envlab}
%% Subtract 0.1 inch from vertical dimensions!!
\SetLabel{4.19in}{1.23in}{0.73in}{0.16in}%
         {0.19in}{2}{7}
\newif\ifboxlabel
\let\oldPrintLabel\PrintLabel
\renewcommand{\PrintLabel}[1]{%
  \ifboxlabel
    \fbox{\oldPrintLabel{#1}}%
  \else
    \oldPrintLabel{#1}%
  \fi}
\boxlabeltrue
\endinput
```

The myenvlab package could then be used in a file like the one below to print out a set of address labels, where the `\add` macro defined the name and address for a label.

```
% file xmas.tex
\documentclass[12pt]{letter}
\usepackage{myenvlab}
%%\boxlabelfalse
\newcommand{\add}[1]{%
  \mbox{}\mlabel{\mbox{}}{#1}}
\newcommand{\UK}{UNITED KINGDOM}
\startlabels
\begin{document}
\add{John Doe \\
     98765 931st St \\ Someplace YN 12345}
\add{A N Other \\
    The House \\ The Road \\ Town \\ \UK}
% etc., etc.
\end{document}
```

Having printed out sheets of labels it occurred to me that probably other labels would be required at other times. My wife was also talking about starting a new address book because the current one was becoming illegible due to many deletions, changes, and additions. I was toying with the idea of subverting BibTeX into an address database but fortunately hesitated before having the joy of programming with the BibTeX language. I now keep all the addresses in a file that looks like this, where I specify a macro for each name and address, and any other personal details that might be of interest:

```
% file addresslist.tex
%%% \add{name}{address}{telephone}{email}{notes}
\newcommand*{\UK}{UNITED KINGDOM}
\newcommand*{\DoeJ}{\add{John Doe}%
  {98765 931st St \\ Someplace YN 12345}%
  {(981) 123--4567}%
  {\url{jd576@email.moc}}%
  {birthday 01/01/01}}
\newcommand*{\OtherAN}{\add{A N Other}%
  {The House \\ The Road \\ Town \\ \UK}%
  {+44 1273 5798 8975}%
  {\url{ano@ano.org}}%
  {dog: Fido}}
% etc., etc.
\endinput
```

Using a suitable definition for `\add`, which in this case puts the various arguments into a minipage, I can print an address book by:

```
% file addressbook.tex
\documentclass[12pt,twocolumn]{article}
```

```
\usepackage{url}
\newcommand{\add}[5]{%
  \begin{minipage}{\linewidth}\raggedright
  #1 \\ #2 \\ #3 \\ #4 \\ #5 \end{minipage}%
  \\[\baselineskip]}
\begin{document}
\input{addresslist}
\DoeJ
\OtherAN
% etc., etc.
\end{document}
```

With a different definition for `\add`, which here just uses the first two arguments, namely the name and address, I can print labels by:

```
% file xmas.tex
\documentclass[12pt]{letter}
\usepackage{url}
\usepackage{myenvlab}
\newcommand{\add}[5]{%
  \mbox{}\mlabel{\mbox{}}{#1\\#2}}
\startlabels
\begin{document}
\input{addresslist}
\DoeJ
\OtherAN
% etc., etc.
\end{document}
```

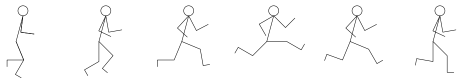
> Now, *here*, you see, it takes all the running *you* can do, to keep in the same place. If you want to go somewhere else, you must run at least twice as fast as that.
>
> *Through the Looking Glass*,
> LEWIS CARROLL

## 2 Animated books

*While sorting through some old files I found a piece written by Jeremy Gibbons for his* Hey — it works! *column but which was not published before he handed his baton over to me. Jeremy kindly gave me permission to include it here.*

When I was a child, my father used to make little booklets, each page with a slightly different picture; flicking through the booklet quickly makes a ‘movie’. Recently James Willans from York asked on `comp.text.tex` how to achieve this effect in LaTeX, and a Michael Liebling answered. Here I show a simpler version of Liebling’s approach.

First you need a collection of little pictures; the following assumes that they are all the same size. I used METAPOST to generate a running man in different positions. Here is a representative sample:



The METAPOST file `running.mp` to create these figures is linked from the *TUGboat* web page `http://tug.org/TUGboat/Contents/contents27-1.html`.

Next we compute the page number modulo the number of different pictures:

```
\def\compute@modulus#1#2{%
  \@tempcnta=#2\relax
  \divide\@tempcnta by #1\relax
  \multiply\@tempcnta by #1\relax
  \multiply\@tempcnta by -1\relax
  \advance\@tempcnta by #2\relax}
```

So for example `\compute@modulus{12}{\thepage}` computes the page number modulo twelve.

We also work out how far to move the image on each page, dividing the difference between the text width and the image width by the number of pages:

```
\newcount\pagecount
\pagecount=100
\setbox0=\hbox{%
  \includegraphics[scale=0.5]{running.0}}
\newdimen\distance
\distance\textwidth
\advance\distance by -\wd0
\divide\distance by \pagecount
```

Finally, we use the `plain` page style, and put the right image in the right place on each page:

```
\newdimen\offset
\def\@oddfoot{%
  \offset=\distance
  \multiply\offset by \thepage
  \hskip\offset
  \compute@modulus{12}{\thepage}%
  \includegraphics[scale=0.5]%
    {running.\the\@tempcnta}%
  \hfil}
\let\@evenfoot\@oddfoot
```

All that remains is to generate the right number of pages:

```
\loop
  \mbox{} \newpage
\ifnum \pagecount>0
  \advance\pagecount by -1
\repeat
```

### References

[1] Boris Veytsman. Printing Envelopes and Labels in LaTeX 2ε: EnvLab Package User Guide, June 1996. Available on CTAN in `latex/macros/contrib/envlab`.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
`herries dot press (at) earthlink dot net`

## Brackets around anything

Timothy Hall

### Introduction

It is often the case when labeling terms in a figure that the label itself cannot be placed so that it refers to an unambiguous term in the figure. For example, consider the label $h$ in Figure 1.

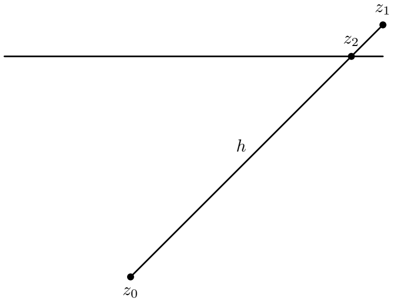


Figure 1: Ambiguous label $h$

Does the $h$ refer to the segment between points $z_0$ and $z_1$, or only between points $z_0$ and $z_2$? It would be clearer if there were a bracket explicitly delineating the term in the figure corresponding to $h$, such as that found in Figure 2.

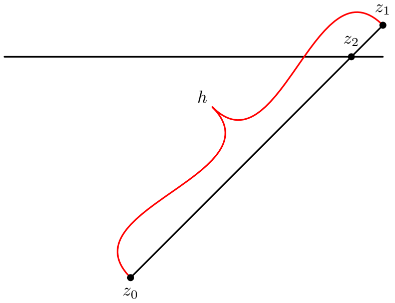


Figure 2: Clear label $h$

The placement of a bracket around terms of interest in a figure may be accomplished through the use of the following flexible and easy to use general purpose METAPOST[1] definition.

```
def sbrack(suffix r,s,d) text c=
   save a,v;
   numeric a;
   pair v;
   a=angle(z.r-z.s);
   v=0.5[z.r,z.s]+((d*pt,0)
                   rotated (a+90));
   draw z.r{dir (a+90)}
      ..{dir (a+90)}v{dir (a-90)}
      ..{dir (a-90)}z.s
      withcolor c;
enddef;
```

Its syntax is

`sbrack(#,#,size)color`

where the first # is the bracket's starting point, the second # is the bracket's ending point, and *size* is the distance (in standard METAPOST points) from the midpoint between the starting and ending points to the "vertex" of the bracket (the distance being measured perpendicular to the line segment[2] between the two points). The *color* refers to the drawing color[3] of the bracket, and may be specified by a standard name, such as "red," or by an RGB triple, such as $(1, 0, 1)$ for magenta, or by any other representation recognized by the METAPOST modifier `withcolor`.

Note that the order of the points listed in the definition is important. The `sbrack` utility always draws the bracket *to the right* as it progresses from the starting point to the ending point. So if the bracket in Figure 2 were drawn by

`sbrack(1,0,36)red;`

then the use of

`sbrack(0,1,36)red;`

would draw the bracket on the other side of the line (see Figure 3).

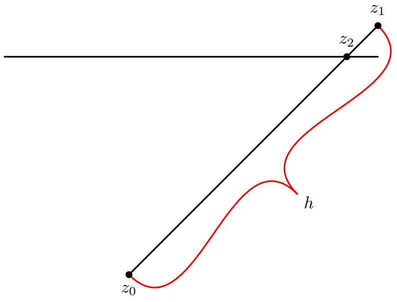


Figure 3: Bracket on other side

[1] Note that, except for the `withcolor` modifier, this definition could also be a METAFONT definition, should the need ever arise.

[2] A line segment need not be part of the figure for the midpoint to be calculated; see Figure 5.

[3] For black and white printers and displays, the `withcolor` modifier will result as a "shade of gray" rendering of the corresponding color.

### Placement of labels

The label $h$ in Figure 2 was placed by

```
label.lft(%
   btex $h$ etex,
   0.5[z0,z1]
   +((40pt,-6*sind(angle(z1-z0))*pt)
   rotated(angle(z1-z0)+90)));
```

Since the vertex of the bracket was placed 36 points from the line (in this particular case), this label is placed to the left of the vertex by an additional 4 points (for a total of 40 points). The term involving the sine (in degrees) ensures that the label will be "pointed to" by the curvature of the path around the vertex of the bracket.

The label $h$ in Figure 3 was placed by

```
label.lft(%
   btex $h$ etex,
   0.5[z0,z1]
   +((40pt,4*sind(angle(z0-z1))*pt)
   rotated(angle(z0-z1)-90)));
```

which only differs from before in the order of the points ($z_0$ and $z_1$ are interchanged), and that positive 4 is used instead of negative 6, and the rotation angle is 180 degrees from the previous form. The choice of 4 or 6 is subjective and depends on the particular circumstances where it is used[4] (and on the preferences of the user). However, the positive and negative signs, as well as the +90 degrees and −90 degrees choice in the rotation, must be used correctly depending on which side the bracket is drawn. Using the wrong values will result in (a) the label being on the opposite side as the bracket, or (b) the label being significantly misaligned, and therefore distracting.

The label itself may be rotated to align consistently with the bracket itself. For example, in Figure 4, the label was produced by

```
label.lft(%
   btex $\left\|\rho\right\|$ etex,z3)
   rotatedabout(z3,angle(z0-z1)-90);
```

where $z_3$ is the vertex of the bracket.[5] The complete METAPOST code for this example is instructional, as it combines several aspects of displaying a bracketed figure.

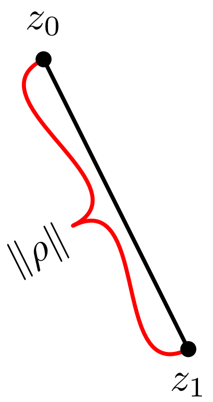


Figure 4: TeX Label

```
beginfig(4);
   pickup pencircle scaled1pt;

   z0=(0,0);
   z1-z0=(0.50in,-1.00in);
   z3=0.5[z1,z0]
     +((12pt,2*sind(angle(z0-z1))*pt)
      rotated(angle(z0-z1)+90));

   draw z0..0.5[z0,z1]..z1;
   sbrack(0,1,12)red;

   pickup pencircle scaled4pt;
   drawdot z0;
   drawdot z1;

   label.top(btex $z_0$ etex,z0);
   label.bot(btex $z_1$ etex,z1);
   label.lft(%
      btex $\left\|\rho\right\|$etex,z3)
      rotatedabout(z3,angle(z0-z1)-90);
endfig;
```

Straight line segments are not required for `sbrack` to work properly, since its definition only requires non-identical points, a spacing measure, and a drawing color (see Figure 5). None of these parameters have a default value. Note that it is up to the individual user to place the brackets and labels drawn by `sbrack` so that they do not interfere with any other terms in the figure.

### Summary example

Figure 6 summarizes the flexibility of the `sbrack` METAPOST definition. It is drawn by the following program:

[4] In the example given, the nature of the ascender on the left side of the letter "h" is the primary reason for using 4 rather than 6.

[5] Note `rotatedabout` or `rotatedaround` must be used, rather than `rotated`.

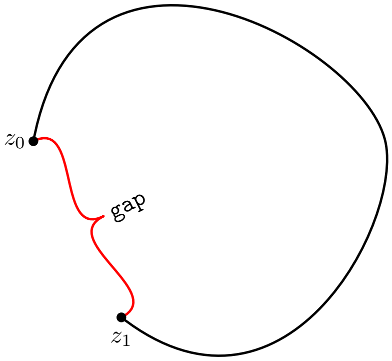


Figure 5: Indicator Label

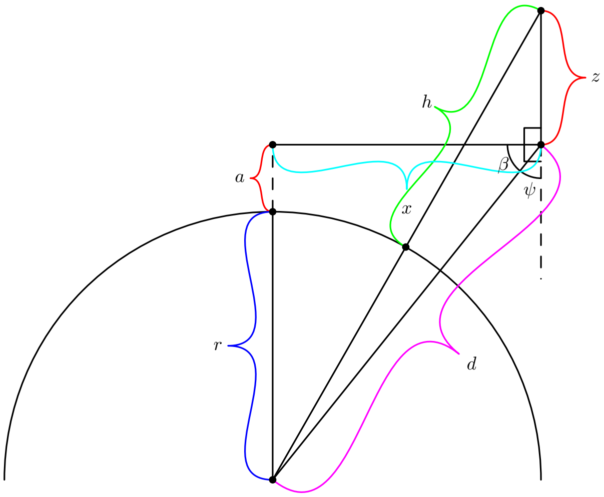


Figure 6: Summary Example

```
beginfig(6);
   path pth[];
   pickup pencircle scaled1pt;
   pth[1]=halfcircle scaled4in;

   z0=(0,0); z1-z0=(0,2in);
   z1'-z1=(0,0.5in); z1''=0.5[z1,z1'];
   z2-z1'=(2in,0);
   pth[2]=quartercircle scaled1/2in
          rotated180 shifted z2;
   z2'-z2=(0,-1in); z3-z2=(0,1in);
   z4=pth[1] intersectionpoint (z0--z3);
   z5'-z2=(-1/8in,0); z5''-z2=(0,1/8in);
   z5'''-z2=(0,-1/8in);
   z6=pth[2] intersectiontimes (z0--z2);
   pth[3]=subpath (0,x6) of pth[2];
   pth[4]=subpath (x6,2) of pth[2];

   draw halfcircle scaled 4in;
   draw z1'--z2; draw z0--z1;
   draw z0--z2; draw z2--z3;
   draw z0--z3;
   draw z2--z2' dashed evenly scaled2;
   draw z1--z1' dashed evenly scaled2;
   draw z5'--(x5',y5'')--z5'';
   draw z5'--(x5',y5''')--z5''';
   draw pth[3]; draw pth[4];
   sbrack(1',1,12)red;
   sbrack(3,4,24)green;
   sbrack(0,2,36)(1,0,1);
   sbrack(1',2,24)(0,1,1);
   sbrack(2,3,24)red;
   sbrack(1,0,24)blue;

   pickup pencircle scaled4pt;
   drawdot z0; drawdot z1; drawdot z1';
   drawdot z2; drawdot z3; drawdot z4;

   label.top(btex $x$ etex,
             0.5[z1',z2]+((40pt,
             -6*sind(angle(z1'-z2))*pt)
             rotated(angle(z1'-z2)+90)));
   label.rt(btex $d$ etex,
            0.5[z0,z2]+((40pt,
            4*sind(angle(z0-z2))*pt)
            rotated(angle(z0-z2)+90)));
   label.rt(btex $z$ etex,
            0.5[z2,z3]+(24pt,0));
   label.lft(btex $r$ etex,
             0.5[z0,z1]-(24pt,0));
   label.lft(btex $a$ etex,
             z1''-(12pt,0));
   label.lft(btex $h$ etex,
             0.5[z3,z4]+((24pt,
             -4*sind(angle(z3-z4))*pt)
             rotated(angle(z3-z4)+90)));
   label.lft(btex $\beta$ etex,
             point 3*x6/4 of pth[2]);
   label.bot(btex $\psi$ etex,
             point (1+x6/2) of pth[2]);
endfig;
```

### Technical note

The term $d$ in the sbrack definition is actually not a **suffix** as it is labeled. It is a **text** argument; indeed, a numeric that is joined with *pt to form the $x$-value of a **pair**. However, the syntax of sbrack usage is simplified, i.e., made to be of shortest character length, by placing $d$ with the suffixes, rather than as a separate text or expression. To wit:

```
    def sbrack(suffix r,s)(text d)text c =
```

is cumbersome, not only for the redundant use of **text**, but also because the syntax would then be

```
    sbrack(0,1)(24)red;
```

and for

```
    def sbrack(suffix r,s)(text d,c) =
```

the calling syntax would be

```
    sbrack(0,1)(24)(red);
```

both of which require more characters than the given definition. In fact, using **expr** in place of **text** would be inappropriate, since neither $d$ nor $c$ should be evaluated. However, if a further development or enhancement were made to the `sbrack` definition to allow $d$ to be a variable, then the use of **expr**, in this context, would be not only appropriate, but required.

## Further development and enhancements

A user may, of course, change any of the parameters found in the definition of `sbrack`, such as changing the relative position of the vertex (perhaps 75% of the way from the starting point to the ending point).

Another possibility is redefining the position of the label (relative to the vertex) to be on the opposite side of the vertex as is now given. In this way, the label would appear between the bracket and a line segment between the defining points.

Yet another value-added enhancement would be for the macro to calculate a default bracket spacing value, perhaps in such a way that the bracket does not "collide" with the term it is enclosing, and does not interfere with any other terms in the figure.

Some intriguing results come from adding *tension* statements to the curves generated by the *draw* command, as well as explicitly defining the control points of a curve in terms of the label position. Any such further developments and enhancements may be incorporated into the `sbrack` definition to meet any particular METAPOST (or METAFONT) need.

◇ Timothy Hall
PQI Consulting
P. O. Box 425616
Cambridge, MA 02142-0012
`http://www.pqic.com/TUG/baa.htm`

# Omega

## Typesetting Malayalam using Ω

Alex A.J.

### Abstract

This paper explains the installation and usage of a package for typesetting the Malayalam language using the Ω system. This package supports both the Traditional (Old Lipi) and Reformed (New Lipi) Malayalam scripts and provides two font families. The ΩTPs and macros are explained in detail.

### Introduction

Typesetting Malayalam with TEX was first implemented by Mr. Jeroen Hellingman, who developed a package for plain TEX. In this package, Malayalam text is written in an ASCII transliteration scheme, which in turn was converted into proper TEX source using a preprocessor written in C. He also developed a primitive METAFONT for both the Traditional and Reformed Malayalam scripts.

A new and improved package for LATEX was written by Alex A.J. in 2003. Two professional quality fonts were provided along with proper hyphenation and many other improvements. However, it still used the original preprocessor and transliteration scheme from Hellingman's package.

Using a preprocessor has several disadvantages. First of all, there are two source files, a `.mm` file (the LATEX source file with transliterated Malayalam text) and a `.tex` file which is produced by the preprocessor. There is no control over hyphenation; the preprocessor simply put discretionary hyphens after every character. It was necessary to use an external editor to delete unwanted hyphens.

The Ω system, developed by John Plaice and Yannis Haralambous, has proven to be a useful solution for Indic language typesetting. It supports the Unicode standard and accepts UTF-8 encoded text as input files. Thus the use of a preprocessor is entirely avoided and Malayalam text can be directly processed by Ω.

### Installation and usage

The package is available as a tarball and can be installed using the installation script provided with it. It can also be downloaded from CTAN. The package contains a number of ΩTPs, two font families in Type 1 format, and additional files supporting other commercial fonts etc.

To typeset Malayalam text, you simply use the command `\mal` inside a group, and enter your text using a UTF-8 enabled editor, such as Yudit (`http://yudit.org`).

### ΩTPs in detail

This is a technical description of the ΩTPs included in this package. It assumes some knowledge of Ω and Malayalam script.

**mal-uni01.otp** In this short ΩTP, a 'soft hyphen' (`@"0D4F`) is added to every Malayalam syllable, using lines like the following:

```
{consonant}{depA} => \1 \2 @"0D4F ;
{consonant}{depi} => \1 \2 @"0D4F ;
{consonant}{depI} => \1 \2 @"0D4F ;
...
```

The current Unicode standard does not provide slots for Malayalam 'Cillu letters'. The general practice among developers is to use ZWJ and ZWNJ to differentiate between Cillu letters and normal virama forms. The government of Kerala has proposed assigning code positions `0D7A`, `0D7B`, `0D7C`, `0D7D` and `0D7E` to the five Cillu letters. In this ΩTP, the ZWJ forms are mapped to the above locations using the following lines:

```
@"0D23 {virama}{zwj} => @"0D7A @"0D4F ;
@"0D28 {virama}{zwj} => @"0D7B @"0D4F ;
...
@"0D33 {virama}{zwj} => @"0D7E @"0D4F ;
```

**mal-uni02.otp** In this (also short) ΩTP, unwanted hyphen characters are removed in several places: before *anuswara*, *visarga*, Cillu letters, and others. Hyphens at the end of words are also removed.

**The final stage** In this stage, the Unicode locations are mapped to character positions in the actual fonts. Four ΩTPs are provided with the package enabling the use of many font families. Two of them, `mal-uni2keli.otp` and `mal-uni2rch.otp` provides typesetting in the Malayalam Reformed Script (പുതിയ ലിപി) using the Keli (കേളി) and Rachana (രചന) font families. `mal-uni2oldrch.otp` provides Traditional Malayalam script (പഴയ ലിപി) from the Rachana family. It is explained below in detail. The final ΩTP, `mal-uni2ism.otp`, supports over 35 Malayalam font families from the Indic language software 'CDAC ISM Publisher'.

**mal-uni2rch.otp** First of all, the ligature expressions (കൂട്ടക്ഷരങ്ങൾ) are identified and mapped to the corresponding glyph in the font. For example, the line:

```
{ka}{virama}{ka} => "\<183>" ;
```

identifies the sequence ‘ക + ് + ക’ and produces the ligature ‘ക്ക’, which is glyph 183 in the Rachana font.

One thing to note is that ligatures coming along with dependent vowels in which a character is placed to the left of the ligature (in the above example: ക്കെ, ക്കേ, ക്കൈ, ക്കൊ, and ക്കോ) must have entries before the plain ligature (ക്ക).

Similarly, all ligatures that are present in the font file have corresponding lines in this ΩTP.

Next, the dependent vowels are mapped to appropriate locations in the font with the following:

```
{depA} => "n" ;
{depi} => "o" ;
{depI} => "p" ;
{depu} => "q" ;
{depU} => "r" ;
{deprr} => "s" ;
...
{consonant}{depe} => <= "t" \1 ;
{consonant}{depE} => <= "u" \1 ;
...
```

The dependent form of ‘ര’ gets special treatment similar to the above (for example, ക + ് + ര = ക്ര).

Finally, the independent vowels and consonants are mapped to glyphs in the font with lines like the following:

```
{a} => "A" @"0D4F ;
{A} => "B" @"0D4F ;
{i} => "C" @"0D4F ;
{I} => "\<164>" @"0D4F ;
...
{ka} => "I" ;
{kha} => "J" ;
{ga} => "K" ;
{gha} => "L" ;
...
```

The fake Unicode character 0D4F, which is used as a soft hyphen, is replaced with a TEX discretionary hyphen.

```
@"0D4F => "\-" ;
```

## Producing UTF-8 Indic TEX files

One main objective of developing this package was that the user must be able to see Malayalam text as the source file is prepared.

Although OpenOffice supports Malayalam and produces decent UTF-8 text, the character display is very primitive; most of the dependent vowels are shown on the wrong side of the characters they are associated with. The same is true of the standard editor Vim.

A decent solution is the Yudit editor developed by Gaspar Sinai. It uses OpenType fonts for displaying text and comes with many transliteration schemes for almost all Indic languages. Writing a new transliteration scheme is also very easy. This package includes support for using Yudit to prepare Malayalam TEX source files using a phonetic transliteration scheme.

## Fonts

Two font families are included in this package in Type 1 format. The first one is Keli (കേളി) whose character set includes Malayalam Reformed script (New Lipi). The second one, Rachana (രചന) contains more than 900 glyphs in six font files. This font enables typesetting in the Traditional Malayalam script (Old Lipi).

## Conclusion

Ω seems to be capable of handling all the complexities involved in typesetting Indic languages. Having UTF-8 support makes it all the easier for developers to write ΩTP’s for any given font. Another advantage is that any UTF-8 compatible editor can be used for creating source files.

We hope this article will help developers to create support for other Indic languages.

This work was supported in part by the TEX Development Fund (http://tug.org/tc/devfund).

◇ Alex A.J.
Lilly Dale
Mukkolakkal, Nedumangad P.O.
Thiruvananthapuram-695541
India
indicTeX (at) gmail.com
http://sarovar.org/projects/malayalam

# TUG 2006 — schedule

| | | | |
|---|---|---|---|
| **Wednesday November 8** | 15:00 | *registration* | |
| | 16:00 | *tour around Marrakesh monuments in open bus* | |
| **Thursday November 9** | 08:00 | *registration* | |
| | 08:30 | **Khalid Sami**, UCAM-FSSM | *Welcome* |
| | 09:00 | *break* | |
| | | Morning session | chair: **Hans Hagen** |
| | 09:30 | **Thomas Feuerstack, Klaus Höppner** | *ProTeXt, a complete TeX system for beginners* |
| | 10:10 | **Taco Hoekwater** | *MetaPost developments — autumn 2006* |
| | 10:50 | **Jean-Michel Hufflen** | *Names in BibTeX and mlBibTeX* |
| | 11:25 | **Gerben Wierda, Renée Van Roode** | *TeX Live — Life with TeX* |
| | 12:00 | *lunch* | |
| | | Afternoon session | chair: **Nelson H. F. Beebe** |
| | 14:00 | **Hans Hagen** | *OpenMath and MathML in practice* |
| | 14:40 | **Jerzy B. Ludwichowski** | *TeX producing legal documents* |
| | 15:20 | **Zdeněk Wagner** | *Babel speaks Hindi* |
| | 16:00 | *break* | |
| | 17:30 | *guided tour of Marrakesh medina/old city and Jamee El-Fna place* | |
| **Friday November 10** | | Morning session | chair: **Jonathan Kew** |
| | 08:30 | **Yannis Haralambous** | *Infrastructure for typesetting high-quality Arabic* |
| | 09:15 | **Youssef Jabri** | *The Arabi system — TeX writes in Arabic and Farsi* |
| | 10:00 | *break* | |
| | 10:15 | **Hossam A.H. Fahmy** | *AlQalam for typesetting traditional Arabic texts* |
| | 10:50 | **Mustapha Eddahibi**, Azzeddine Lazrek, Khalid Sami | *DadTeX — A full Arabic interface* |
| | 12:00 | *lunch* | |
| | | Afternoon session | chair: **Barbara Beeton** |
| | 14:40 | **Jonathan Kew** | *Unicode and multilingual typesetting with XeTeX* |
| | 15:20 | Mohamed Jamal Eddine Benatia, **Mohamed Elyaakoubi**, Azzeddine Lazrek | *Arabic text justification* |
| | 16:00 | *break* | |
| | 16:15 | round table: *Arabic typography* | moderator: **Yannis Haralambous** |
| | 17:30 | *city tour of Marrakesh* | |
| **Saturday November 11** | | Morning session | chair: **Robin Laakso** |
| | 08:30 | **Claudio Beccari** | *LaTeX2$_\varepsilon$, `pict2e` and complex numbers* |
| | 09:15 | **Morten Høgholm** | *Page design in LaTeX3* |
| | 10:00 | *break* | |
| | 10:15 | Hans Hagen, **Jerzy Ludwichowski**, Volker RW Schaa | *The New Font Project: TeX Gyre* |
| | 10:50 | **Karel Píška** | *Outline font extensions for Arabic typesetting* |
| | 11:25 | **Chris Rowley**, Frank Mittelbach | *Everything we want to know about modern font technologies* |
| | 12:00 | *lunch* | |
| | | Afternoon session | chair: **Klaus Höppner** |
| | 14:00 | **Elena Smirnova, Stephen M. Watt** | *Generating TeX from mathematical content with respect to notational settings* |
| | 14:40 | **Adrian Frischauf**, Paul Libbrecht | *dvi2svg: Using LaTeX layout on the Web* |
| | 15:20 | **Hans Hagen** | *LuaTeX and ConTeXt versions* |
| | 16:00 | *break* | |
| | 16:15 | **Gyöngyi Bujdosó** | *TeX, typography & art together* |
| | 16:30 | **Barbara Beeton** | *How to create a TeX journal: A personal journey* |
| | 16:15 | round table: *Mathematics on the Web* | moderator: **Stephen M. Watt** |
| | 19:00 | *banquet at Chez Ali* | |
| **Sunday November 12** | 08:00 | *Excursion to Essaouira* | |

(Speakers are in **bold**.)

# TUG 2006

## Sponsors

**Cadi Ayyad University ▪ CSTUG ▪ DANTE e.V. ▪ GUTenberg ▪ Faculty of Sciences ▪ Pole of Competences STIC ▪ TeX Users Group**

*Thanks to all! Thanks also to all the speakers and participants, without whom there would be no conference. Special thanks to the FSSM Communication Service and Computer Sciences Department for providing local assistance, and to Duane Bibby for the (as always) excellent drawing.*

## Program committee

C. Beccari, PT, Italy
K. Berry, TUG, USA
H. Hagen, Pragma ADE, Netherlands
Y. Haralambous, ENSTB, France
K. Höppner, DANTE, Germany
B. Hughes, UM, Australia
A. Lazrek, UCAM-FSSM, Morocco
A. Lindsay, Lancs Uni, UK
S. Peter, Beech Stave Press, USA
J. Plaice, UNSW, Australia
B. Raichle, US, Germany
K. Sami, UCAM-FSSM, Morocco
V. RW Schaa, DANTE, Germany
C. Swanepoel, UNISA, South Africa
A. Syropoulos, GTF, Greece

## Organizing committee

K. Berry, R. Laakso, TUG, USA
M. El Adnani, A. Lazrek,
K. Sami, UCAM-FSSM, Morocco

## Participants

*Claudio Beccari,* Politecnico di Torino, Italy `claudio dot beccari (at) polito dot it`

*Nelson H. F. Beebe,* University of Utah, USA `beebe (at) math dot utah dot edu`

*Barbara Beeton,* American Mathematical Society, USA `bnb (at) ams dot org`

*Mohamed Jamal Eddine Benatia,* Ibn Youssef School, Morocco `benatiamje (at) yahoo dot fr`

*Gyöngyi Bujdosó,* University of Debrecen, Hungary `bujdoso (at) inf dot unideb dot hu`

*Marc Dehon,* Ibn Ghazi School, Morocco `majdi_sam (at) yahoo dot fr`

*Mustapha Eddahibi,* Cadi Ayyad University, Morocco `m dot eddahibi (at) ucam dot ac dot ma`

*Mohamed Elyaakoubi,* Cadi Ayyad University, Morocco `m dot elyaakoubi (at) ucam dot ac dot ma`

*Hossam A. H. Fahmy,* Cairo University, Egypt `hfahmy (at) arith dot stanford dot edu`

*Thomas Feuerstack,* FernUniversität, Germany `Thomas dot Feuerstack (at) FernUni-Hagen dot de`

*Adrian Frischauf,* German Research Center for Artificial Intelligence, Germany `adrianf (at) activemath dot org`

*Steve Grathwohl,* Duke University Press, USA `grath (at) duke dot edu`

*Hans Hagen,* Pragma ADE, The Netherlands `pragma (at) wxs dot nl`

*Yannis Haralambous,* ENSTB, France `yannis.haralambous (at) enst-bretagne dot fr`

*Hartmut Henkel,* Von Hoerner & Sulger, Germany `hartmut_henkel (at) gmx dot de`

*Taco Hoekwater,* Elvenkind BV, The Netherlands `taco (at) elvenkind dot com`

*Morten Høgholm,* LaTeX3 Project, Denmark `morten dot hoegholm (at) latex-project dot org`

*Klaus Höppner,* DANTE e.V., Germany `klaus (at) dante dot de`

*Brian Housley,* University of Berne, Switzerland `brian dot housley (at) gccs dot ch`

*Jean-Michel Hufflen,* University of Franche-Comté, France `hufflen (at) lifc dot univ-fcomte dot fr`

*Youssef Jabri,* Mohammed I University, Morocco `yjabri (at) ensa dot univ-oujda dot ac dot ma`

*Steffen Kernstock,* WPT Kernstock, Germany `s dot kernstock (at) kernstock dot com`

*Jonathan Kew,* SIL International, UK `jonathan_kew (at) sil dot org`

*Reinhard Kotucha,* Hannover, Germany `reinhard dot kotucha (at) web dot de`

*Robin Laakso,* TeX Users Group, USA `office (at) tug dot org`

*Azzeddine Lazrek,* Cadi Ayyad University, Morocco `lazrek (at) ucam dot ac dot ma`

*Jerzy B. Ludwichowski,* Nicolas Copernicus University, Poland `Jerzy.Ludwichowski (at) uni dot torun dot pl`

*Karel Píška,* Academy of Sciences, The Czech Republic `piska (at) fzu dot cz`

*Arthur Reutenauer,* GUTenberg, France `arthur dot reutenauer (at) gmail dot com`

*Jon Riding,* Oxford Brookes University, UK `jon dot riding (at) plenumorganum dot org dot uk`

*Renée van Roode,* R & A, The Netherlands `Renee dot van dot Roode (at) rna dot nl`

*Chris Rowley,* Open University, UK `C dot A dot Rowley (at) open dot ac dot uk`

*Khalid Sami,* Cadi Ayyad University, Morocco `k_sami (at) ucam dot ac dot ma`

*Volker RW Schaa,* DANTE e.V., Germany `volker (at) dante dot de`

*Elena Smirnova,* University of Western Ontario, Canada `elena (at) orcca dot on dot ca`

*Vytas Statulevicius,* VTeX Ltd., Lithuania `vytas (at) vtex dot lt`

*Ina Talandiene,* VTeX Ltd., Lithuania `ina (at) vtex dot lt`

*Sigitas Tolusis,* VTeX Ltd., Lithuania `sigitas (at) vtex dot lt`

*Zdeněk Wagner,* Ice Beare Soft, The Czech Republic `wagner (at) cesnet dot cz`

*Stephen M. Watt,* University of Western Ontario, Canada `watt (at) csd dot uwo dot ca`

*Gerben Wierda,* R & A, The Netherlands `Gerben.Wierda (at) rna dot nl`

## Conference information

The TEX Users Group's twenty-seventh annual meeting and conference is organized in collaboration with the Computer Science Department of the Faculty of Sciences Semlalia, Cadi Ayyad University, located in Marrakesh, Morocco. About thirty talks are expected to be held over three days, starting on November 9th, 2006. More than forty people are expected to attend the conference, coming from diverse places: Algeria, Canada, the Czech Republic, Denmark, Egypt, France, Germany, Greece, Hungary, Italy, Lithuania, Morocco, the Netherlands, Poland, Switzerland, Syria, the United Kingdom, and the United States.

## Topics

After TUG 2003 in USA (Hawaii), TUG 2004 in Europe (Xanthi, Greece), TUG 2005 in Asia (Wuhan, China), TUG 2006 is, for the first time, held in North Africa (Marrakesh, Morocco). The theme of the meeting is **Digital Typography & Electronic Publishing: Localization & Internationalization**.

Our goal is to focus on the new challenges presented by the rapidly growing desire for best localization. Indeed, processing of multilingual e-documents is now needed beyond the limits of traditional cultural areas. The TEX environment's localization of languages using an Arabic-alphabet based script are examples of such concerns. Of course, a wealth of material on various other topics and projects will also be presented. This theme gave us the opportunity to welcome contributions outside the strict scope of TEX.

## Talks and proceedings

We tried to organize the sessions by gathering talks in similar areas of interest. Day one will focus on aspects of the TEX overall environment, discussing various distributions, important programs, and applications. For the second day, we concentrate on the topic of Arabic typography and typesetting. The final day brings together topics from the web, fonts, and LATEX.

Preprints will be available at the meeting. The final proceedings will be an issue of TUGboat, as usual, and will be published shortly after the conference.

## The drawing

Duane Bibby's TEX lion and its small computer companion help set the context of the conference. The Lion is dressed as an Arab Calligrapher entering a dome in the island of Africa from a large Andalusian gate. The word TEX is mirrored and some Arabic Indic numbers are there to remind that the conference is held in a country which writes in Arabic.

We can also notice the Koutoubia tower (1184–1199). The minaret (tower) is the most famous monument of Marrakesh. It was used as the architectural model for the famous Giralda tower of Sevilla (Spain) and for the Hassan tower of Rabat also. The tower's name is derived from the Arabic al-Koutoubiyyin (librarians), since it was surrounded by sellers of manuscripts and calligraphers.

Of course, the palm tree adds a final grace note.

## Thanks

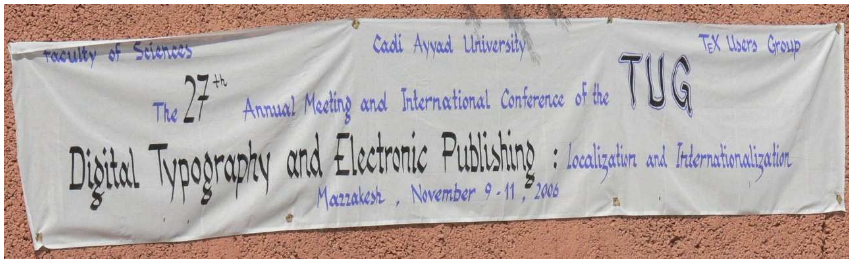
First, we must thank all the members of the organizing and programme committees for their kind and helpful discussions and encouragement. We also thank Karl Berry for his work preparing the conference (he was unfortunately unable to attend, and sends his regrets). We also owe thanks to Barbara Beeton — despite not being listed on the formal committees, she was always present and always helpful. Finally, we thank Khalid Sami for his invaluable help.

Best regards,

Azzeddine Lazrek
Conference chair

# TUG 2006 Report

Taco Hoekwater
taco (at) elvenkind dot com


Faculty of Sciences
Cadi Ayyad University
TeX Users Group
The 27th Annual Meeting and International Conference of the TUG
Digital Typography and Electronic Publishing : Localization and Internationalization
Marrakesh , November 9 - 11 , 2006

## Abstract

After TUG 2003 in America (Hawaii, USA), TUG 2004 in Europe (Xanthi, Greece), TUG 2005 in Asia (Wuhan, China), TUG 2006 was held in Africa, more precisely in Marrakesh. Processing multilingual e-documents went beyond the limits of its traditional cultural areas and new horizons in the internationalization of TeX were explored.

*Keywords:* TUG 2006, Marrakesh, Arabic, international user meeting

### Introduction

The TeX User Group's 27th annual meeting and conference was organized in collaboration with the Computer Science Department of the Faculty of Sciences Semlalia, Cadi Ayyad University, located in Marrakesh, Morocco. About thirty talks were given over three days, starting on November 9th, 2006. The conference was attended by more than forty people from all over the world.

The main topic and subtitle of the conference was: 'Digital Typography & Electronic Publishing: Localization & Internationalization'. It follows that there was a large focus on Arabic typography, but even so, a number of interesting presentations on altogether different subjects were given. A short day by day overview follows.

### Wednesday November 8, arrival

Hans Hagen and I arrived (together with a few other people) at the airport of Marrakesh late in the morning, where we were picked up and chauffeur-driven to our hotel. Luckily, the hotel was only a few hundred meters away from the conference location, but that did not stop us from taking more than an hour to walk over there.

There was no official program for this day, but because many of the international attendants arrived in the morning and early afternoon, the local organization had thoughtfully scheduled a tour of Marrakesh city by (open) double-decker bus. As luck would have it, during this tour we had the only rainfall of the entire week, so we all got wet and had to hurry inside the bus. But from then on, we had a steady plus 20 degrees Celsius and lots of sunshine, so no complaints about the weather.

### Thursday November 9, day 1

The day started early, with the official welcoming speeches starting at half past eight. We were welcomed by Boumedine Tanouti of UCAM (the university itself) and Mohssine Belkoura from FSSM (the hosting faculty), and of course by TUG.

The first talk of the day was by Thomas Feuerstack and Klaus Höppner, who presented "ProTeXt, a complete TeX system for beginners". Most Windows-using readers will have seen the ProTeXt disk in previous years' TeX Collection. The very easy to install and use TeX system based on MiKTeX and TeXnicCenter will of course also be included this year. They are still looking for a Dutch translator for the installation manual, so if you want to help them out, please send me a message.

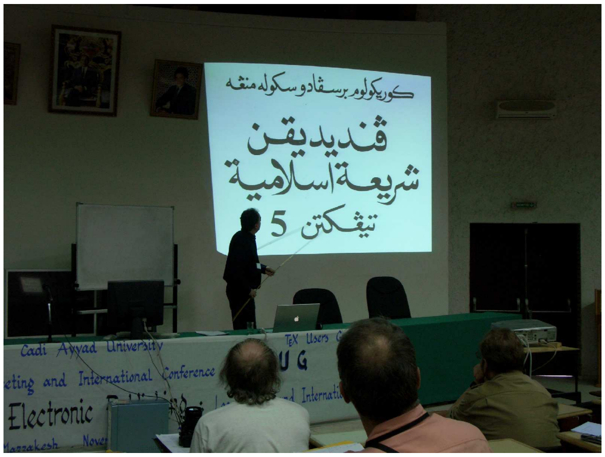
Next was my presentation of the new developments for the upcoming MetaPost release. The most important news is that MetaPost can now do font re-encoding and subset-ting. You can read about that in the previous (EuroTeX) issue of *TUGboat*.

Jean-Michel Hufflen talked about the use of person names in his program MlBibTeX versus the traditional BibTeX. MlBibTeX ('Ml' stands for 'Multilingual') features much improved input and handling of people's names. His quite colorful slides demonstrated a number of problematic names along with their usage in MlBibTeX.

Coding math
in XML
Ayyad University
TEX Users Group
d International Conference of the TUG
ronic Publishing : Localization and Internationalization
Nove 2006

Hans Hagen, OpenMath

The Souks



Yannis Haralambous

The other two Dutch people present were Renée van Roode and Gerben Wierda, and together they talked about 'TEX Live — Life with TEX'. Renée gave a very nice overview of the past six years that Gerben has been working on the i-Installer and i-Packages for gwTEX. After that, Gerben took over with a more technical explanation of what the i-Installer does. It came as bit of a shock to most of us that the last slide was titled 'I Quit'. Other people will continue to work on TEX for Mac OS X, but Gerben will be sorely missed.

Moroccan lunches are not be taken lightly. Or perhaps I should say: Moroccan lunches cannot be taken in lightly. The food was excellent all through the conference, and especially so the lunches. These were served in a restaurant that was situated behind an unadorned garden door in a residential street. Considering the outstanding quality (and quantity) of the food served there, it is no surprise that they keep it a well guarded secret!

The afternoon program began with Hans Hagen telling everybody about the Mathadore project. The Dutch Mathadore project uses OpenMath to provide courseware for secondary education, and this content is typeset by ConTEXt after an intermediate conversion to MathML using an XSLT script that is part of the ConTEXt distribution.

Next up, Jerzy Ludwichowski talked about how Copernicus University in Toruń uses TEX to handle the admission forms for students. These legal documents are generated automatically from the base data in the administrative system, handling tens of thousands of admissions each year, with a minimum of effort.

The last talk of the day was Zdeněk Wagner: Babel speaks Hindi. The presentation described how the Babel module for Hindi was made, and it showed some of the problems associated with typesetting Hindi. For instance, in this script letters are reordered in the output, creating the need for a preprocessor to facilitate the input.

In the evening, there followed another visit to the old city of Marrakesh. We did a walk through the Souks, and returned to the hotel by means of horse and carriage.

### Friday November 10, day 2

Friday was completely focused on Arabic typography. The first talk was by Yannis Haralambous. His talk, 'Infrastructure for high-quality Arabic typesetting' kicked off the day by unveiling the plans for Arabic typesetting support in the coming $\Omega_2$. He showed us the planned data structure and the algorithmic steps that will be used to produce high quality output in a large number of different languages that use the Arabic script.

Youssef Jabri then talked about 'The Arabi system — TEX writes in Arabic and Farsi'. Arabi is a newly arrived package that provides Arabic script support for LaTeX without the need for an external preprocessor or a special TeX extension. It adds support for the Arabic and Farsi languages to Babel, it comes with a set of suitable fonts, and it understands a number of different input encodings. Support for Tifinagh, Syriac and even Urdu is planned for the near future.

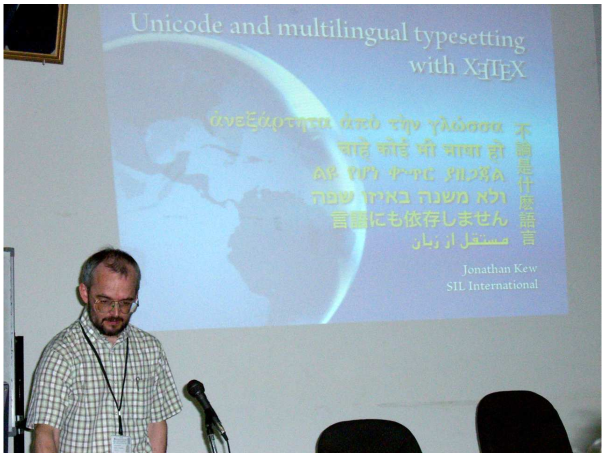
Karel Píška demonstrated some interesting techniques that can be used to improve existing font technologies. The focus of 'Outline font extensions for Arabic typesetting' was on using a dynamic representation of the glyph shape so as to allow runtime generation of curvilinear kashida connections instead of the rule-based fillers that are common today.

After the break, Hossam Fahmy introduced us to his system that aims at typesetting the Qur'an: AlQalam ("the pen" in Arabic). The system evolved out of modifications to ArabTeX. Much work is still needed, especially in the area of fonts, but also when it comes to low-level support from the typesetting program. Nevertheless, the intermediate results are already quite impressive.

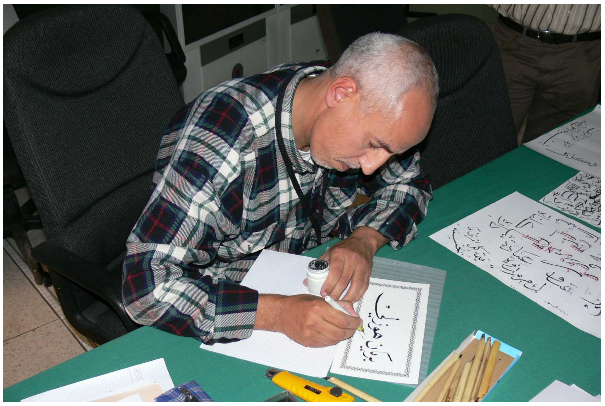
Have you ever felt the need to key in LaTeX commands in your native language? Well, if you are speaking Arabic, now you can! Mustapha Eddahibi presented 'DadTeX — A full Arabic interface'. This project makes it possible for Arabic-speaking people who are not familiar with the English language to use LaTeX. This project also involves Azzeddine Lazrek and Khalid Sami from UCAM.

Next in line was Taco Hoekwater, acting as stand-in for Idris Hamid. He summarized the Oriental TeX project, its objectives and time-line. He also explained the relationship between this project, which is sponsored by Colorado State University (USA), and the pdfTeX successor LuaTeX, as well as upcoming MetaPost versions.

After yet another impressive lunch, Jonathan Kew demonstrated some of the multi-lingual capabilities of XƎTEX. Besides a set of 'prepared earlier' slides, he also gave an impressive live demonstration. The second part of his talk was all about the growing Unicode math support in XƎTEX. This will no longer need special TeX fonts but is now able to use the new Unicode support found in the latest OpenType fonts.



Jonathan Kew



Mohamed Benatia, calligraphing participants' names

Mohamed Elyaakoubi presented research on the justification of Arabic text. Hyphenation of words has been forbidden in Arabic for centuries. Various other methods of justification are used instead, like stretching intermediate connectors between letters and the use of an elaborate system of ligatures and alternative letter forms. The paper corresponding to this talk was a joint effort with Azzeddine Lazrek and the calligrapher Mohamed Benatia, who kindly calligraphed the phonetic representation of all of our names in Arabic.

Officially, this was to have been the last talk of the day, but to kick off the round table discussion that followed, Yannis Haralambous presented a set of slides showing all the various languages that use Arabic script and the enormous amount of variation that can be found therein. After that, the round table discussion followed. This was attended by a fair number of local students as well as TeX-ies.

## Saturday November 11, day 3

Claudio Beccari began the final day of talks, talking about the use of `pict2e`, a fairly recent addition to the LaTeX repertoire of drawing packages. The focus was on how you can use complex numbers to simplify drawing.

LaTeX3, and especially page design, was highlighted by Morten Høgholm. Because he received many general questions in preceding days, he also explained quite a bit about the project infrastructure and the internals of the upcoming LaTeX3. Although there is still no release date, but the code is ready enough that interested people are requested to beta-test the new system.

After the coffee break, there was a session on fonts. It started with Jerzy Ludwichowski who presented the TeX Gyre font project.

Macintosh users can be proud of the fact that the i-Installer already has support for the first three font families that came out of this project, because in the next slot, Gerben actually built the required i-Packages as a demonstration of how to use i-Installer 'from the other side'.

At this point, a presentation was squeezed in. Yannis Haralambous made a brave attempt to replace Apostolos Syropoulos based on his slides entitled 'LaTeX as a tool for the typographic reproduction of ancient texts'. For a large number of ancient scripts, this talk explained some of the problems related to the script and how well it can be typeset using current LaTeX. Yannis worked hard to compensate the missing author by showing examples he had found of the scripts in question.

For 'Everything we want to know about modern font technologies', Chris Rowley turned his presentation into a panel session, made possible by the presence of many font experts. He had prepared some questions and after discussing them briefly, a lively discussion followed. Among the topics discussed were the fundamental differences between the typesetting of Western scripts (by pasting together glyphs) and the Arabic script which is still strongly rooted in the calligraphic world, something not trivial to do with computers. The discussion ended with musings about the potential of using MetaPost as an integral part of TeX, enabling the use of dynamically generated fonts.

Hurray, after that it was time for lunch again! Following that, Elena Smirnova talked to us about generating TeX from mathematical content while honoring notational preferences. To this end, they convert from an extended form of Content MathML to Presentation MathML or TeX input, using a specific notation style that can be selected by the user using a simple GUI interface. The conversion application itself is controlled by an XML-based catalog of possible presentations and conversions.

'DVI2SVG: Using LaTeX layout on the web' was presented by Adrian Frischauf. He convincingly demonstrated that you can create really good looking math in web pages by converting the TeX-generated DVI file into SVG, and then serving the SVG to the web browser. The biggest (perhaps only) drawback of this approach is that right now, it needs a separate plug-in to be installed in order to view the page. The built-in support inside web browsers is not yet good enough to handle complex text native.

One of the many fantastic lunches.

Hans Hagen talked about the impact of LuaTeX on ConTeXt. He gave an overview of the status quo of the LuaTeX project and gave some examples of how and where Lua comes into play.

After the coffee break and the traditional group photograph, the final session started with a continuation of Gyöngyi Bujdosó's talk from EuroTeX 2006. 'TeX, typography & art together' focused on the typographical side of a TeX- and typographical e-learning system that is being developed in Hungary. This second half of the system revolves around a database containing typefaces, paintings, page layouts, and information about their designers.

More than twenty-five years of *TUGboat* history were summarized by Barbara Beeton in the final presentation of TUG 2006. Barbara presented a large number of examples that were scanned in from previous volumes as an aid in explaining how the current layout came to be.

Several talks had dealt with the more trendy ways of coding (XML) and presenting math (web, dynamic, right-left). In the final discussion Stephen Watt challenged the audience to come up with visions and requests concerning coding and representing math as well as current and future demands of accessing math on the web. Being a member of the forums that discuss these items in relation to standards, the author triggered discussion and also invited the audience to bombard him with emails regarding the subject.

That night was the banquet, held at a special place called 'Chez Ali'. After dinner, a special performance was held in the center courtyard of the

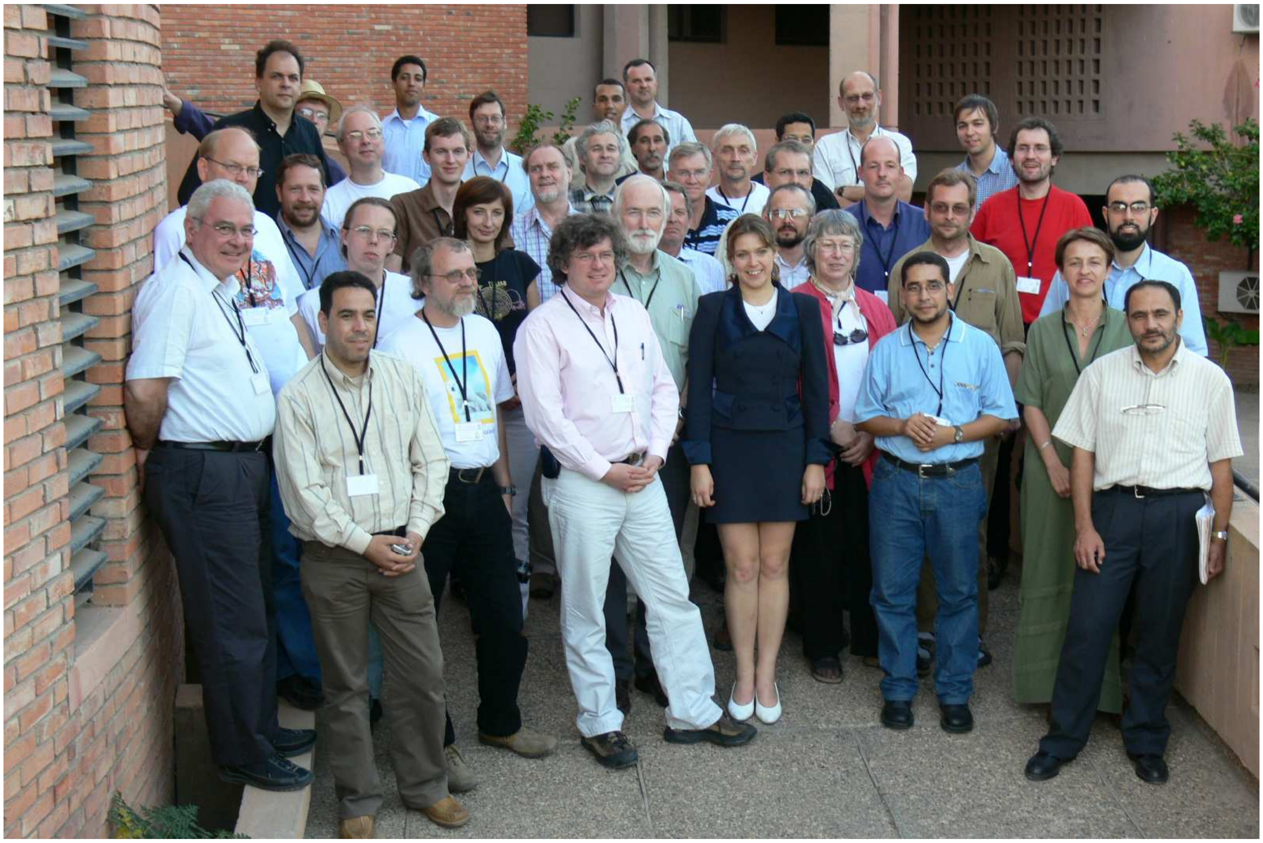
complex. At the end of that there were fireworks and a ritual burning of the TEX Users Group logo!

**Sunday November 12, excursion and closing**

Even though the last talk was on Saturday, almost everybody stayed around for the excursion to Essaouira on Sunday. Perhaps in part because the banquet was the night before, so that is a trick worth remembering. After the crowded and metropolitan Marrakesh, it was nice to see a town that was perhaps just as commercial, but much, much smaller in scale.

When we got back to the hotel in the evening, the time had come to officially close the conference. Many words of thanks were spoken there already, but let me repeat again: a big thank you to the local organization and especially Azzeddine Lazrek. It has been a great TUG conference!

(A sampling of conference photos are below, courtesy of Hartmut Henkel and Volker Schaa. Many more are at `http://bel.gsi.de/tug2006/`. *Ed.*)



*Back row:* Yannis Haralambous, Arthur Reutenauer, Hartmut Henkel, Mustapha Eddahibi, Jon Riding, local participant, Sigitas Tolusis, Vytas Statulevicius, Adrian Frischauf.

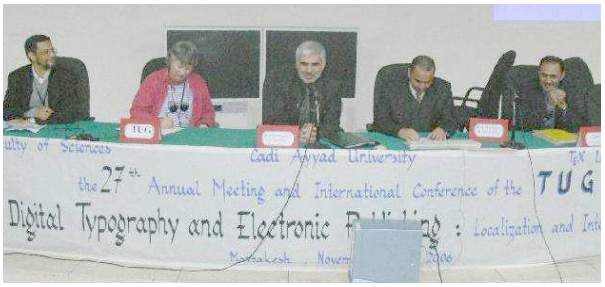
*Second from back row:* Hans Hagen, Steffen Kernstock, Morten Høgholm, Chris Rowley, Karel Píška, local participant, Steve Grathwohl, Jerzy Ludwichowski, Khalid Sami, Thomas Feuerstack, Klaus Höppner.

*Second from front row:* Claudio Beccari, Taco Hoekwater, Ina Talandiene, Nelson H.F. Beebe, Brian Housley, Jonathan Kew, Jean-Michel Hufflen, Reinhard Kotucha, Hossam Fahmy.

*Front row:* Youssef Jabri, Zdeněk Wagner, Stephen Watt, Elena Smirnova, Barbara Beeton, Mohamed Elyaakoubi, Gyöngyi Bujdosó, Azzeddine Lazrek.

Other participants not in the group photo:
Volker RW Schaa (first in back row),
Robin Laakso (last in front row), Marc Dehon,
Mohamed Jamal Eddine Benatia.

Conversing after another fabulous lunch.


ulty of Sciences
Cadi Ayyad University
the 27th Annual Meeting and International Conference of the TUG
Digital Typography and Electronic P
: Localization and Int
Marrakesh Novem 2006

Official opening: Khalid Sami (organizing committee), Barbara Beeton (TUG), Boumedine Tanouti (UCAM vice-president), Mohssine Belkoura (FSSM Vice-doyen), and Azzeddine Lazrek (conference chair).

Double-decker bus tour on arrival day.

Jemaa el-Fna, guided tour of the old city.

Excursion to Essaouira, on the coast.

Closing remarks and thank-yous to all.

# Arabic text justification

Mohamed Jamal Eddine Benatia,
Mohamed Elyaakoubi and Azzeddine Lazrek
Department of Computer Science,
Faculty of Science, University Cadi Ayyad
P.O. Box 2390, Marrakesh, Morocco
Phone: +212 24 43 46 49 Fax: +212 24 43 74 09
lazrek (at) ucam dot ac dot ma
http://www.ucam.ac.ma/fssm/rydarab

**Abstract**

Justification of Latin script based texts is carried out by handling of hyphenation and insertion of inter-word glue, which can be stretched or shrunk to some extent. Due to the cursive nature of Arabic writing, text justification in Arabic has a quite different logic. So, the classical algorithms of text justification must be completely revised. Justification in Arabic typography has traditional processes inspired by calligraphy manuals. Words are flexible: on the one hand, a word can be resized through stretching some letters in a curvilinear way; on the other hand, it can be shrunk via ligatures so that the width of a given letter group is decreased.

محاذاة النص العربي

**ملخص**: لاجراء محاذاة النص في الكتابة بالحرف اللاتيني أو ما شابه تعتمد برامجيات معالجة النص تقطيع الكلمات المحاذية للهامش الايسر، طبقا لقواعد نحوية خاصة بلغة النص، وزرع فراغات قابلة للتمطيط أو التقليص. أما الخط العربي فهو ممشوق لا يسمح بفصل حروف الكلمة عن بعضها ما دامت قواعد الكتابة تقضي بذلك كما أن تقطيع الكلمات أمر غير مألوف فيه ولكنه بالمقابل يتيح مرونة تمطيط حجم الكلمات وتمديد الحروف بشكل انسيابي بواسطة استعمال الكشيدة كما أنه يتيح تقليص طول الكلمات بواسطة تراكب الحروف وتداخلها عموديا. كل هذا يعني أن أسلوب تحقيق محاذاة النص في الكتابة بالخط العربي يختلف تماما عن نظيره في الكتابة بالحرف اللاتيني وأن مراجعة لوغارثمات المحاذاة من أصلها أمر لا مناص منه.

## 1 Justification in Latin typography

A good many methods to make paragraphs on a page be visually homogeneous have been developed. The majority of these methods have already been implemented in TeX. The hyphenation and justification algorithm divides a paragraph into lines in an optimal way, as regards time complexity as well as the visual result obtained [6]. The processing spreads out beyond the paragraphs, to reach the level of the page in its totality.

### 1.1 Typographical hyphenation

Typographical hyphenation is the breaking of words when they come at the end of a line and would overflow into the margin. In general, word breaking happens at syllable boundaries.

In the beginning, the hyphenation of words was done by hand. In 1983, F. M. Liang [8] published a sophisticated method to find nearly all the suitable places to insert hyphens in a Latin script based written word. The method is controlled by an organized tree structure of *tries*, containing a list of hyphenation patterns. Combinations of letters which allow, or prohibit, the word-breaking are listed, and priorities to breakpoints in letter groups are assigned. Patterns reflect the hyphenation rules of a given language. So there will be as many pattern tree structures as there are normative languages. This is the algorithm Donald E. Knuth chose to implement in TeX.

### 1.2 Spacing

To give a line the flexibility it needs, a space or its equivalent can presumably be inserted between each pair of words. Therefore, some of these spaces are transformed into line ends. Others are transformed into variable sized spaces called glue. The glue has a normal size that can be stretched or shrunk. When a paragraph intended to have a justified right margin

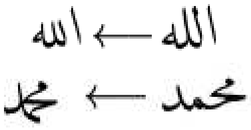


Figure 1: Special morphology

is composed with TeX, the glue widths of each line are adjusted so that lines end almost at the right margin. Generally, the last line of a paragraph is an exception, and does not have to end at the margin.

Given this, it is always an extremely difficult task to obtain a uniform typographical gray. The main reason is the impossibility of ensuring an equal inter-word space in different lines. The composition accidents of *rivers* and *alleys* are uncomfortable for the reader, and irregular spaces catch his attention.

A solution for such problems has been implemented by Hàn Thế Thành [11]. Instead of (or as well as) changing inter-word spaces to justify text lines, the widths of characters are slightly modified. So, better inter-word spacing can be obtained and space elasticity can be limited. This width modification is implemented through horizontal scaling of fonts in pdfTeX. If it is employed parsimoniously and wisely, this method can appreciably improve appearance of the typography produced by TeX.

## 2 Justification in Arabic typography

Arabic writing is cursive in its printed form as well as in its handwritten form. The letters' morphology changes according to their position in the word, according to the surrounding letters, and in some cases, according to the word's meaning (for example, ALLAH and Mohamed when it indicates the prophet's name in Figure 1). The alternative positions then depend on the typeset words. The end of a given glyph is tied to the beginning of the following glyph, with no possible break.

**Remark**:
All figures and the table are written from right to left according to Arabic writing direction.

### 2.1 Kashida

The genuine connections between Arabic letters are curvilinear strokes that can be stretched or shrunk according to the writing context. Such curve is called *kashida*, *tamdid*, *madda*, *maT*, *taTwil*, or *iTalah*. This variable-sized connection between letters is specific to Arabic alphabet based writing. The kashida is used in various circumstances:

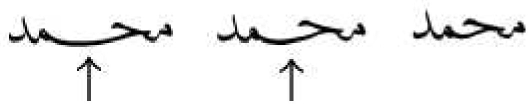


Figure 2: An Arabic word with kashida

- **emphasis**: to mark an important piece of a word. The kashida will then mark the sound elongation;
- **legibility**: to give a better character layout on the baseline, and to lessen the cluttering at the joint point between two successive letters of the same word;
- **aesthetics**, to embellish the writing of a word;
- **justification**, to justify a text line.

The example in Figure 2 shows a composition of an Arabic word; the arrows indicate the kashidas, with various degrees of extensibility.

There are mandatory elongations, allowed elongations and prohibited elongations. The typographical quality of a text is determined, among other things, by the absence of mandatory elongations or the presence of prohibited elongations.

In terms of Arabic text justification, the kashida is a typographical effect that allows the lengthening of letters in some carefully selected points of the line, with determined parameters, in order to produce the left alignment of a paragraph. The good selection of characters to be stretched is called *tansil*.

### 2.2 Current typesetting systems

In terms of text processing tools, the curvilinear kashida is, generally, still beyond what the majority of typesetting systems can afford. The kashida is not a character in itself, but an elongation of some character parts while keeping rigid the body's character. It is not a simple horizontal scaling to widen character width. Instead of performing a kashida, the majority of typesetting systems proceed by inserting a rectilinear segment between letters; the resulting typographical quality is unpleasant. Due to the lack of adequate tools, the solution consists of inserting a glyph, that is, an element of a font. So, rather than computing (say) parameterized Bézier curves in real time, a ready-to-use character is inserted. Moreover, whenever stretching is performed by means of a parameterized glyph coming from an external dynamic font, the current font context is changed.

Curvilinear extensibility of characters can be afforded by certain systems through the a priori generation of curvilinear glyphs for some predefined sizes.

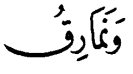


Figure 3: Presence of Noon-Meem ligature
(Source: Holy Quran written by the calligrapher Alhaj Hafez Mohamed Amin Alrochdi, scrutinized and revised by General Directorate of Endowments, Baghdad, p. 649.)

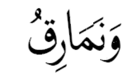


Figure 4: Absence of Noon-Meem ligature

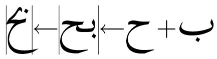


Figure 5: Contextual and aesthetic transformations

Beyond these sizes, the system will choose curvilinear primitive and linear fragments. Of course, this will violate the curvilinear shape of letters and symbols composed at large sizes [3, 9, 10].

A better approach consists of building a dynamic font [4, 7], through parameterizations of the composition procedure of each letter. The introduced parameters indicate the extensibility size or degree. To handle the elongations, a letter is decomposed into two parts: the static body of the letter and the dynamic part, capable of stretching.

### 2.3 Ligatures

The cursive nature of Arabic writing implies, among other things, a wide use of ligatures [5]. Indeed, Arabic writing is rich in ligatures. Some ligatures are mandatory and obey grammatical and contextual rules [5]. Others are optional and exist only for aesthetic reasons, legibility and/or justification. Moreover, the connection of letters, in the course of writing cursively, can lead to the introduction of implicit contextual ligatures. An explicit ligature is the fusion of two, three, or even more, graphemes.

Some aesthetic ligatures result in some reading ambiguity. So, precise texts such as the Holy Quran are sometimes written without such ligatures. Figure 3 shows an example of such ambiguity: the Noon-Meem ligature can be confused with the initial form of the letter Ghain. Figure 4 shows the text without the ligature.

Generally, the ligature width is shorter than the width of the fused grapheme group. For example, the aesthetic ligature بح in Figure 5 is 9.65031pt wide, whereas the ligature given by simple contextual substitutions بح is rather wide (14.754pt).

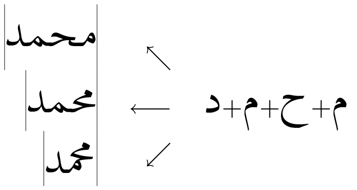


Figure 6: Various levels of ligatures

The control of ligature behavior, by the conversion of implicit ligatures into aesthetic ones, brings some flexibility to the word. So, it can be adapted to the available space on the line. The example in Figure 6 shows three ligature levels: mandatory simple substitutions, aesthetic ligatures of second degree, and finally, aesthetic ligatures of third degree. The two last ligature levels provide shrinking possibilities of the same word.

The use of aesthetic ligatures of second and third degree has to take into consideration the constraints of legibility.

A typesetting system should take into account three levels of recourse to ligatures. In the first level, there are only implicit contextual ligatures and mandatory grammatical ligatures of second degree. This level is recommended for textbook composition, where it is necessary to avoid any collision between characters and/or any reading ambiguity. In a second level, some aesthetic ligatures of second degree can be used. This level is recommended for composition of books for the general public. The third level, where the use of aesthetic ligatures of higher degrees is allowed and liberties in graphic expressions can be taken, is possible in special circumstances.

The use (or not) of the explicit ligatures to improve the justification should take into account the graphic environment and the block regularity of the concerned text. In calligraphy, when an aesthetic ligature is used, there is no obligation to use this ligature in all text occurrences. The justification problem can be resolved by the use of kashidas in a text containing only implicit ligatures.

The use of ligatures to justify lines is not solely an Arabic writing characteristic. Adolf Wild [12], the Gutenberg museum conservator in Mainz, examined the Gutenberg Bible from a typographical point of view. At the level of lines, justification was made via ligatures, instead of today's variable spaces.

### 2.4 Other practices

The baseline (*satr alkitabah*) is the line on which letters stand to form words and therefore phrases. In Arabic writing, the baseline is a virtual line. The characters as well as the words do not stay directly

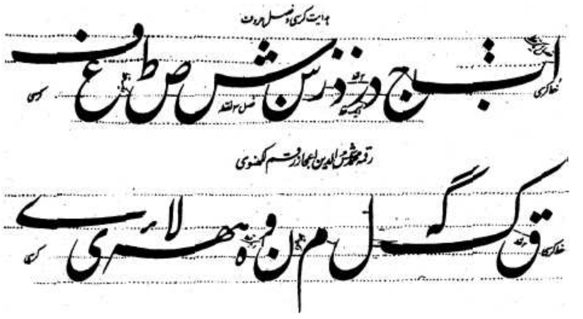


Figure 7: Kursy in ta‘lyq style (Source: Badai‘ Alkhat Al‘arabi, p. 372, Naji Zine Eddine Almasref)

on a static line with a fixed reference. The letters flow in an interactive way with the surrounding letters. They change not only their shape, but also their position with respect to an imaginary baseline. Letters as well as words are free from any absolute reference. Thereby, the letters as well as the words may be tangled. According to Ibn Muqlah [1], good composition implies three types of interacting forces among letters and words:

- *tarsif*, the junction of the conjoint letters, where each letter seeks the point of join with the preceding letter;
- *ta'lif*, the addition of letters in isolated form, where the isolated letter seeks to fill the blank left by the typeface of their neighboring letters;
- *tastir*, the words' ruling, so that they are held horizontally.

From another standpoint, positioning variations of letters and words are at the origin of the *chairs kursy-s* [2] theory, that introduces a number of bearing lines. According to different versions of this theory, the number of these lines can vary from two to five, or even seven lines. The heights between lines vary according to authors of this theory and can not be respected in practice. Apparently, this theory is used more in style ta‘lyq, as in Figure 7. Thereby, we can wonder to what extent we can formalize this concept for a possible automation thereof.

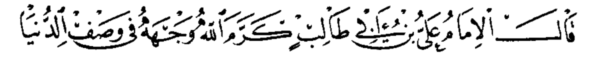


Figure 8: Words tangle

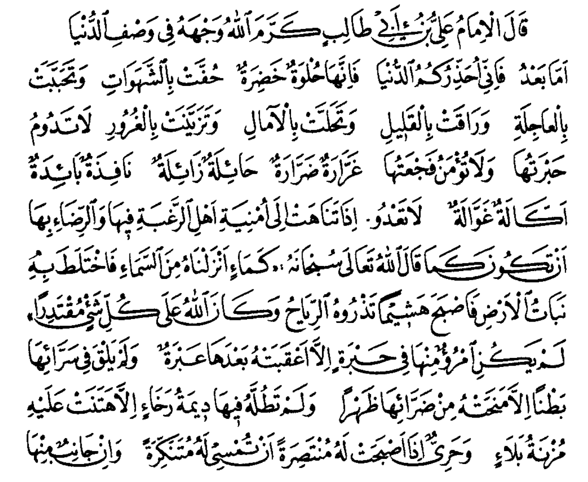


Figure 9: Justification favoring the use of increasing spaces

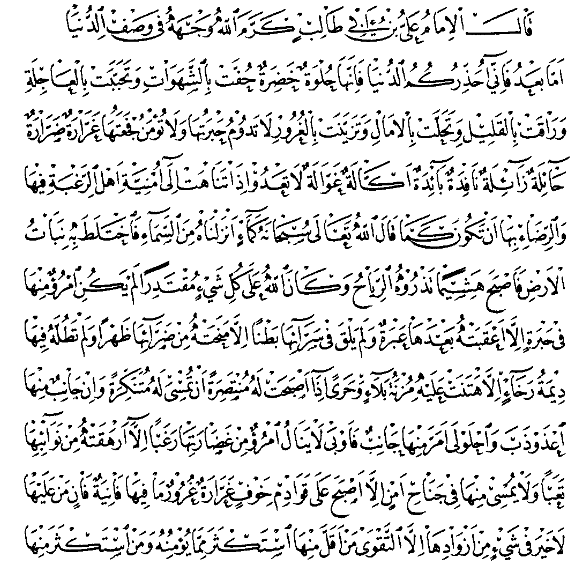


Figure 10: Justification favoring the use of kashida

Nevertheless, we can say that for justification, calligraphers do use the tangling possibilities of letters and words, as in Figure 8, and inter-word space shortening as additional or alternative methods instead of kashidas. The same text was written by two calligraphers and won a prize in the IRCICA[1] 2004 competition. The composition in Figure 9 by M. T. Alubaidy used more spacing and entanglement, while A. Alabdo in Figure 10 favored more kashida.

Calligraphers also build on other practices for justification, such as:

- word heaping; this consists of putting certain words above others — especially, the word Allah above the preceding word (see the end of the seventh line of Figure 11);

[1] `http://www.ircica.org/`

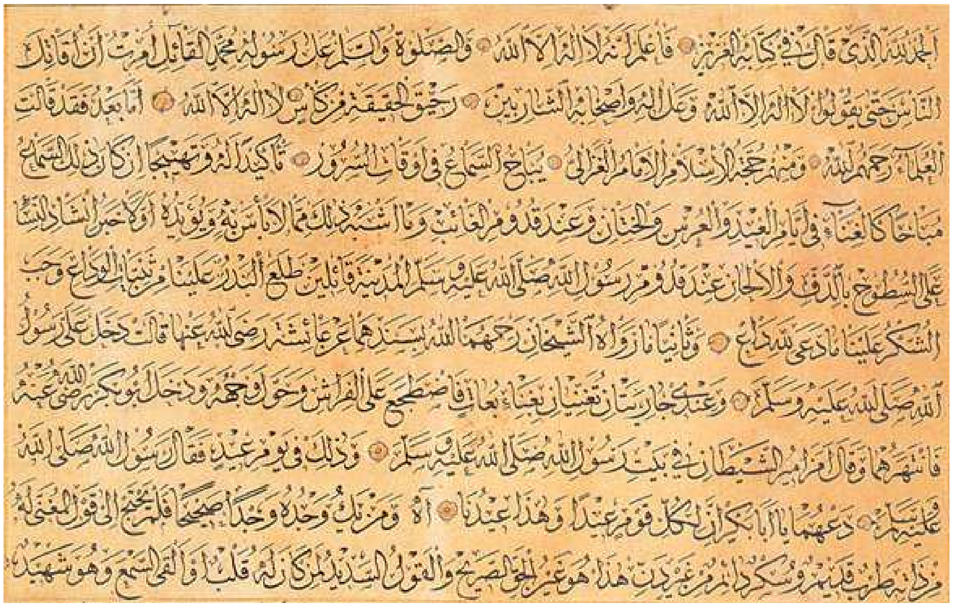


Figure 11: Heaping words and hyphenating fragments

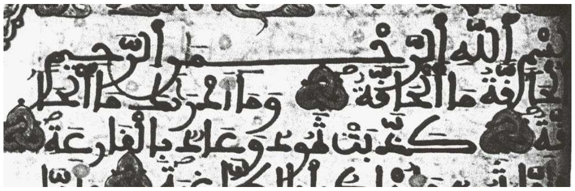


Figure 12: Word hyphenation

- moving the broken fragment above the hyphenated word, as in the last word of the fourth line in Figure 11;
- word hyphenation, as in Figure 12;
- word hyphenation in margin; this is an expulsion of hyphenated fragments to the line's margin, instead of the following line, as in Figure 13;
- decreasing of some words at the end of a line, as in Figures 14 and 15;
- curving of the baseline, as in Figure 16.

Since the 10th century, hyphenations at ends of lines have been strictly prohibited. This was probably due to the cursive structure of the Arabic language, and to the possible absence of vocalization signs. There are even sentences which it is advised not to hyphenate (ex. *salla ALLAH elyh w sallam*). If word hyphenation was accepted in Arabic, legibility would be considerably affected.

Nowadays, only the famous caesura at hemstitch boundary is allowed. Regarded as a pause,

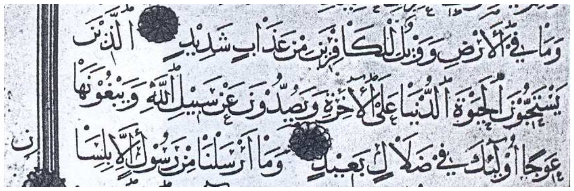


Figure 13: Expulsion in margin of hyphenated fragment (Source: Holy Quran written by Yakout Almusta‘simy with style Ryhany and Kufy in Iran museum)

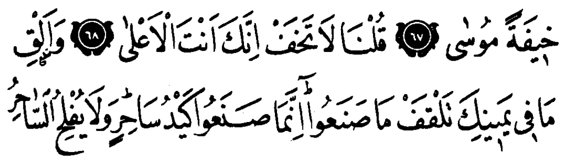

خيفة موسى ۝ قلنا لا تخف إنك أنت الأعلى ۝ وألق
ما في يمينك تلقف ما صنعوا إنما صنعوا كيد ساحر ولا يفلح الساحر

Figure 14: Decreasing of words at end of line

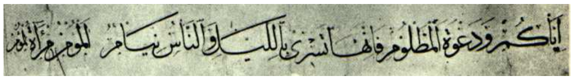


Figure 15: Decreasing of words at end of line (Source: Badai‘ Alkhat Al‘arabi, p. 372, Naji Zine Eddine Almasref)

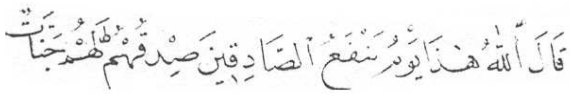
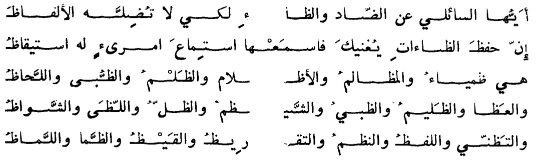
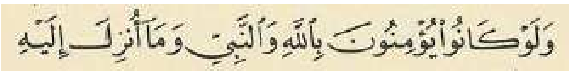


Figure 16: Curving of the baseline



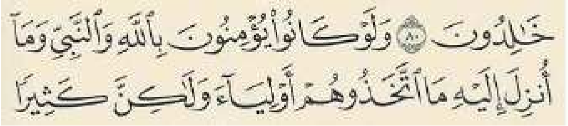
أَيُّها السائلي عن الضّاد والظا ءِ لكي لا تُضِلَّه الألفاظُ
إنّ حفظَ الظاءاتِ يُغنيكَ فاسمَعْها استماعَ امرىءٍ له استيقاظُ
هي ظمياءُ والمظالمُ والأظـ ـلام والظَلْمُ والظُّبى واللَّحاظُ
والعَظا والظَليمُ والظبيُ والشَّيـ ـظمُ والظلُّ واللّظَى والشَّواظُ
والتَظنّي واللفظُ والنظمُ والتقـ ـريظُ والقَيْظُ والظّما واللَّماظُ

Figure 17: Caesura at hemstitch boundary



وَلَوْ كَانُوا يُؤْمِنُونَ بِاللَّهِ وَالنَّبِيِّ وَمَا أُنزِلَ إِلَيْهِ

Figure 18: Stretching Lam



خَالِدُونَ ﴿٨٠﴾ وَلَوْ كَانُوا يُؤْمِنُونَ بِاللَّهِ وَالنَّبِيِّ وَمَا
أُنزِلَ إِلَيْهِ مَا اتَّخَذُوهُمْ أَوْلِيَاءَ وَلَٰكِنَّ كَثِيرًا

Figure 19: Simple Lam

this does not affect legibility since the hyphenated word is still held on the same line, as shown in Figure 17. There is also a stretching letter, shown in Figure 18, which is not allowed anymore, Figure 19.

### 2.5 Diacritic/metric dot

The qalam used to produce the calligraphic style Naskh is a piece of reed like a flute mouthpiece, shown in Figure 20. The reed's head shape corresponds to the graphic style that the qalam or feather will offer. For example, in the Naskh style the shape is beveled, and in the Maghreby style it is pointed. The feather's head in Naskh style is a flat rectangle. During the operation of writing, it has to be maintained with an inclination angle of approximately 70° with a virtual baseline.

The diacritic dot which appears as a tilted square is the typographical unit marked by the feather in use.

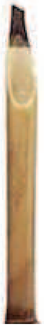
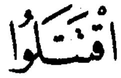


Figure 20: Qalam of Arabic calligraphy

The system of diacritic dots plays a leading semantic role. Indeed, certain letters are characterized by the presence, number and positions of dots. The basic glyph ٮ gives rise to several letters according to the number of diacritic dots which appear above or below: ب, ت, and ث. Similarly, the glyph ح provokes several letters according to presence and position of the dot with respect to the basic glyph: ح, خ and ج (noted as [ج] in Figure 22). In the case of a succession of letters carrying two diacritic dots, the dots may interfere with each other. They can thus be placed horizontally one next to the other, or vertically one above the other, according to the available space above or below the basic glyph, as in Figure 21, or — even better — to stretch letters for better spacing. Moreover, according to the style, the size of the diacritic sign also changes according to the stretching size, also seen in Figure 21.



اقْتَتَلُوا

Figure 21: Diacritic dots positioning
(Source: Holy Quran written by the calligrapher Alhaj Hafez Mohamed Amin Alrochdi, Scrutinized and revised by General Directorate of Endowments, Baghdad, p. 43.)

{ب ت ث} = [ب]
{ج ح خ} = [ج]
{د ذ} = [د]
{ر ز} = [ر]
{س ش} = [س]
{ص ض} = [ص]
{ط ظ} = [ط]
{ع غ} = [ع]
{ب ت ث ن ي} = [بَ]

Figure 22: Letter clusters

The importance of the diacritic dot goes beyond its phonetic role. This dot is also the measure unit used for regularizing the dimensions and the metrics of glyphs. Ibn Muqlah[2] specified letter measurements in metric dots. In order to give

[2] Abou Ali Ben Mohamed Ben Ali bnou Muqlat [272–328 A.H./886–940 A.D.], a native of Shiraz and a minister of the Abbasid caliph in Baghdad, was one of the first theorists of Arabic calligraphy. His contribution to this art was not the invention of a new script but the application of systematic rules that determine the surface area and proportions of the individual letter-shapes with respect to one another.

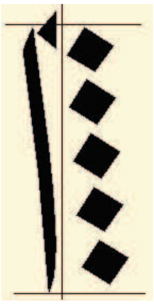


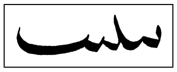
Figure 23: Arabic letter Alef metrics

homogeneity to letters, he includes all the Arabic letters in a circle of six dots diameter, the Alef letter height. Figure 23 shows the metric of the Arabic letter Alef in metric dot. Of course, this notion remains relative and approximate; the dot position in the line and the position of dots among them lead to differences among calligraphers.

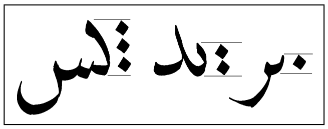
### 2.6 Allographs

The allographs are the various shapes that a letter can take while keeping its place in the word: isolated, initial, median or final. Letters can have additional shapes even though, grammatically, there are only four shapes. Allometry is then the study of allographs: shape, position, context, etc. Generally, the election of an allograph responds to aesthetic needs. So, this choice is left to the writer. However, the use of an allograph is sometimes desired and even recommended. The letter's shape should change according to the letters in the neighborhood and, in some cases, according to the presence of kashida. Some examples:

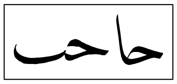
- the median form of the letter Beh should take a more acute shape if it interposes two spine letters:



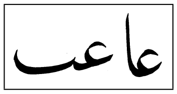
- the initial form of Beh can take three allograph shapes, according to its following letter:



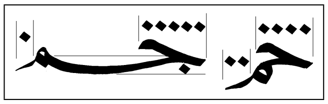
- the initial form of the letter Hah may take the shape of lawzi Hah whenever it precedes an ascending letter:



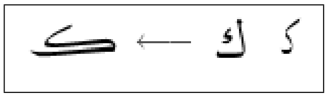
- the initial form of the letter Ain should take the shape of finjani Ain when it is followed by an ascending letter:



- the initial form of the letter Hah as well as the final form of the letter Meem change their morphological forms in the presence of kashida:



- the letter Kaf changes its morphological shape in the case of stretching and should be changed into zinadi Kaf:



## 3 Proposition for Arabic justification

As we have seen, to perform text justification, Arabic writing provides various techniques coming from its handwriting traditions. These techniques are not based on the insertion of variable spaces between words. So, unpleasant spaces are avoided and a balanced aspect can result.

The present study will be restricted to a formalism of the problem of text justification in the calligraphic style Naskh. For reasons of legibility, the Naskh style has been adopted in typesetting since the first attempts of computerization and standardization of Arabic typography. The Naskh style stands between the difficult Thuluth style and easy Ruq‘a style. Our choice for the Naskh style is also motivated by the fact that some calligraphic styles, such as the Ruq‘a style, do not allow the use of kashida. Other styles, such as the *kufi* style, have a geometrical structure which goes beyond the rules of feather and metric dot, while still other styles use letters stretching only for aesthetic purposes. On the other hand, the Nasta‘liq and Diwani styles are more generous with number of kashidas: a word of three letters can indeed receive two elongations.

### 3.1 Breaking a paragraph into lines

Before using ligatures or kashida to justify lines, a system has to first break paragraphs into lines. A legitimate place to break a sequence of words into lines is a position in a place that gives a good length. Therefore, it may be the border of the last word of a horizontal list which generates a width equal to or less than the value of the line width. Or the border of the last word of a horizontal list which generates a width higher than the value of the line width and, through the use of ligatures of second degree, the width of the list can be decreased to result in a value equal to or less than the line width. In general, there are several legitimate breaking places, and the badness of each possibility should be considered so that the best one can be chosen.

### 3.2 Elongation presence

In the following, we will focus on justification using kashidas. Making a system eager to stretch a letter leads to overcoming constraints of determination of the extensibility places with the required parameters. In this context, letter stretching obeys a set of calligraphic rules. These rules can be found in [1] and/or some Arabic treatises on calligraphy and practices.

The Arabic alphabet is composed of 28 letters:

أ ب ت ث ج ح خ د ذ
ر ز س ش ص ض ط ظ
ع غ ف ق ك ل م ن ه و ي

Before giving an account of the rules in detail, let's look first at some general rules of kashida:

- The first letter of a word composed only of two letters may not be stretched, but the second letter can be stretched;
- No more than one letter in the same word can be stretched;
- On a given line, generally only one word can be stretched (the albasmalah sentence in Figure 24 is an exception — there, the first word is always stretched);
- The use of the kashida is recommended where there is a succession of spine letters, to remove reading ambiguity. In particular, the letter Seen should always be followed by an elongation of two metric points;
- A word ending with the letter Yeh [ى][3] cannot be stretched. The presence of kashida in this case generates a new letter *qantara*. On the other hand, it is better to stretch a word when it ends with a pronoun: final letter Heh [ـه]. In this case, the kashida should be placed before the last character.

Now, let us consider the rules concerning where to use kashidas, their number and their degree of extensibility. According to the most general rules we have found, it is advised to use the kashida in the middle of the word. In words composed of four or six letters, the kashida is to be put down on the level of the second or third letter of the word respectively. In the following, we proceed first by generalizations; we classify words according to their number of letters, and enumerate occurrences where stretching is allowed.

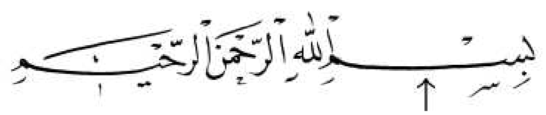


Figure 24: Stretching of letter Seen in albasmalah

**Words with two letters:** It is strictly prohibited to stretch a word with two letters, except in the case of سر or شر. Though it represents only one letter, the initial form of the letter Seen [سـ] can be regarded as a succession of three glyphs: [ـد] [ـد] [د].

**Words with three letters:** Usually, the kashida is omitted in a word composed of three letters. Words such as عسى and فتى cannot be stretched. On the other hand, in the *albasmalah*, the word *bismi* بسم should always be stretched in the level of its letter Seen, as in Figure 24; this elongation was not imposed previously, as in Figure 12.
According to Imad Eddine Ibn Al'afif's[4] opinions, one can stretch a word composed with three letters when the first letter is a ع, ط, [س] or [ج]. According to Azzeftaoui,[5] one can proceed with an elongation if the last letter is an Alef or a Lam. If we decide to stretch such a word, the kashida should be performed on the level of the second letter.

**Words with four letters:** Words with four letters are the most susceptible to be stretched, and it is preferable to put down kashida on the level of the second letter. However, one should not stretch words like: خبير, نمير, and تغلب.

**Words with five letters:** For this case also, the treatises show differences. While Ibn Al'afif forbids the use of kashida in a word of five letters

[3] Brackets are used throughout to indicate a *letter family*, standing for any glyph which shares this root form, differing only in the number of dots.

[4] The opinions of 'afif Eddine Mohamed Alhalabi Achirazi were a reference in the Alqalqashandi's treatise of calligraphy. One of his disciples was his son Imad Eddine Mohamed.

[5] Abou Mohamed Ben Ahmed Azzeftaoui was born in 750 A.H. and passed away in 806 A.H. He was taught calligraphy by Chams Eddine Mohamed Ben Ali Bnou Abi Raquiba.

| Precede | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| و | ه | ن | م | ل | ك | ق | ف | [ع] | [ط] | [ص] | [س] | [ر] | [د] | [ج] | [ب] | أ | Isolated | |
| | 1 | 1 | 1 | −1 | | | | | +1 | | | −1 | −1 | −1 | | 1 | 5 | [ب] |
| −1 | −1 | −1 | −1 | −1 | −1 | | | −1 | +1 | | | −1 | −1 | −1 | | −1 | | [ج] |
| −1 | −1 | 2 | −1 | 2 | 2 | −1 | −1 | −1 | 1 | 2 | 2 | 1 | −1 | −1 | 2 | 1 | 3 | [ط][ص][س] |
| | −1 | −1 | −1 | −1 | | | | −1 | +1 | | | −1 | −1 | −1 | | −1 | | [ع] |
| −1 | | | −1 | −1 | | | | −1 | +1 | | | −1 | −1 | −1 | | 1 | 1 | فق |
| −1 | −1 | −1 | −1 | −1 | −1 | | | | | −1 | | 1 | 1 | −1 | | −1 | 3 | ك |
| | −1 | −1 | −1 | | | | | −1 | | | | | −1 | −1 | | | 3 | ل |
| −1 | −1 | −1 | −1 | −1 | −1 | | | −1 | 1 | | | 1 | 1 | −1 | | −1 | | م |
| | −1 | | | | | | | | | | −1 | −1 | −1 | | 1 | | | ه |

Figure 25: Chart of allowed extensions, by context

as such word cannot be divided into two equal parts, Ibn Khalouf affirms that elongation on the second letter of the word in this case is advised and necessary.

**Words with more than five letters:** For this case, the general rules of letter successions in Figure 25 are the norm.

### 3.3 Elongation places

Some special care is necessary to examine historical calligraphic compositions from a typographical point of view. Many calligraphers went beyond some established rules. They used kashidas generously when they were remunerated by the page so that they were better paid [2].

The heavy use of elongations at the line end is the preferred method for Ibn Wahid.[6] For Ibn Al'afif [1], elongation is best at the end of a line, it can be tolerated in the beginning, and it is forbidden in the middle.

The propositions of the treatises on the legitimate kashida places show differences in some details. While Ibn Al'afif allows letter elongation in the middle of line only when absolutely necessary, Alchirazi does not have objections.

In terms of justification, the heavy use of the kashida at the end of a line by Ibn Wahid and Ibn Al'afif is due to a particular reason: calligraphers can estimate the elongation only when nearing the limit of a line. The kashida is triggered by the distance to the end of line.

The superposition of two elongations on two consecutive lines can be seen only as a defect, even by Ibn Al'afif. To avoid the "stairs" effect resulting from such superposition, a uniform typographical grid can be advised.

### 3.4 Degree of extensibility

The degree of extensibility of stretchable letters depends on some contextual elements:

- the nature of the letter to stretch;
- the position of the letter in the word;
- the position of the word in the line.

In order to speed up the processing, the letters will be grouped into families. The glyphs of each family undergo the same contextual treatment as well as the same extensibility rules.

Figure 25 gives the degree of extensibility in metric dot of each letter according to its context. For example: "The elongation of letter Beh is authorized, if it is followed by Dal, and it is prohibited, if it follows Seen".

Let us recall that the letters {أ, [د], [ر], و} are never stretchable.

In some cases, several possibilities of treatment arise. We will focus on the most widespread and simplest ones.

Notice that:

- in boxes, each number $\imath$ represents the interval $[\imath,12]$ in metric dots;
- an empty box represents a prohibited elongation;

[6] Charaf Eddine Mohamed Bnou Charif Bnou Youssef Azzary, known as Ibn Wahid, was born in Damascus in 647 A.H. He studied in Iraq and lived in Egypt. He was one of Yacout Almousta'simi's disciples.

- the plus sign means an allowed and approved elongation;
- the minus sign means an allowed but not approved elongation, in a number of these cases, one could use ligatures instead of kashidas.

### 3.5 Process

Here is our proposed method for justifying Arabic text:

- break the paragraph into lines with a specific width;
- compute the badness of the individual line;
- refer to the contextual table to determine all legitimate kashida places;
- establish priorities for kashida points;
- distribute the lines' badness as parameters of kashida;
- generate curvilinear kashida with determined parameters;
- stretch letters in the selected points.

## 4 Conclusion

The justification in Arabic writing differs from other writing systems on at least two points. First, for centuries, hyphenation of words has not been allowed. Second, Arabic is endowed with a particular tool for justification, the kashida. If historical treatises on calligraphy provide us with a list of technical specifications, our work consists of checking the agreement among these treatises and the general use in current manuscripts, and then attempting to build a formalism of the degrees of the letters' extensibility according to their context.

**Acknowledgements**:
Thanks to Barbara Beeton, Karl Berry, and Khalid Sami for their editorial corrections and contributions.

### References

[1] Abi Al-abas Ahmed Ibn Ali Al-qanqachandi (821h/1418). *Sobh al-a'cha fi sna't al-incha.* Version copied from edition Al-amiriat with corrections and a detailed indexing. Ministry for the culture and national orientation, general Egyptian institution of production, edition, impression and distribution.

[2] Vlad Atanasiu. *Le phénomène calligraphique à l'époque du sultanat mamluk*. PhD thesis, 2003. `http://www.atanasiu.freesurf.fr/thesis`.

[3] Zeev Becker and Daniel Berry. `triroff`, an adaptation of the device-independent `troff` for formatting tri-directional text. *Electronic Publishing — Origination Dissemination and Design*, 2(3):119–142, October 1989.

[4] Daniel M. Berry. Stretching letter and slanted-baseline formatting for Arabic, Hebrew and Persian with `ditroff/ffortid` and dynamic PostScript fonts. In *Software — Practice & Experience*, number 29:15, pages 1417–1457, 1999.

[5] Yannis Haralambous. Tour du monde des ligatures. *Cahiers Gutenberg*, 22:69–80, 1995.

[6] Yannis Haralambous. Voyage au centre de TEX : composition, paragraphage, césure. *Cahiers Gutenberg*, 44-45:3–53, 2004.

[7] Azzeddine Lazrek. CurExt, Typesetting variable-sized curved symbols. In *EuroTEX 2003: 14th European TEX Conference*, *TUGboat* 24:3, pages 323–327, Brest, France, 2003. `http://www.ucam.ac.ma/fssm/rydarab/doc/communic/curext.pdf`.

[8] Liang, Franklin Mark. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, 1983. `http://tug.org/docs/liang`.

[9] Thomas Milo. ALI-BABA and the 4.0 Unicode Characters — Towards the Ideal Arabic Working Environment, New input output concepts under Unicode. In *EuroTEX 2003: 14th European TEX Conference*, *TUGboat* 24:3, pages 502–511, Brest, France, 2003.

[10] Johny Srouji and Daniel M. Berry. Arabic formatting with `ditroff/ffortid`. In *Electronic Publishing — Origination Dissemination and Design*, volume 5, pages 163–208, 1992.

[11] Hàn Thế Thành. Améliorer la typographie de TEX. *Cahiers Gutenberg, actes du congrès GUT'99*, 32, 1999.

[12] Adolf Wild. La typographie de la Bible de Gutenberg. *Cahiers Gutenberg*, 22:5–15, 1995.

# The Arabi system نظام ﴿العربي﴾ — TEX writes in Arabic and Farsi

Youssef Jabri
École Nationale des Sciences Appliquées,
Oujda, Morocco
yjabri (at) ensa dot univ-oujda dot ac dot ma

### Abstract

In this paper, we will present a newly arrived package on CTAN that provides Arabic script support for TEX without the need for an external pre-processor. The Arabi package adds one of the last major multilingual typesetting capabilities to Babel by adding support for the *Arabic* عربي and *Farsi* فارسي languages. Other languages using the Arabic script should also be more or less easily implementable.

Arabi comes with many good quality free fonts, Arabic and Farsi, and may also use commercial fonts. It supports many 8-bit input encodings (namely, CP-1256, ISO-8859-6 and Unicode UTF-8) and can typeset *classical Arabic poetry.*

The package is distributed under the LATEX Project Public License (LPPL), and has the LPPL maintenance status "author-maintained". It can be used *freely* (including commercially) to produce beautiful texts that mix Arabic, Farsi and Latin (or other) characters.

### ملخص

رزمة العربي نظام يتيح إمكانية استعمال الحروف العربية واللاتينية جنباً إلى جنب في مستند واحد باستعمال نظام « تيخ » TEX لتصفيف الحروف.

رزمة العربي تضيف إمكانية استعمال اللغتين(عربي و فارسي) مع نظام تيخ ومنذ البداية ، فهذا النظام يتميز بكونه محمولا ويتمتع بقدر كبير من المرونة ، لأنه قابل للاستعمال مع معظم ما تم إنجازه مناإضافات . إضافة إلى أنه لا يحتاج إلى أي معالج خارجي لتحديد أشكال الحروف في الكلمة . يقدُمُ العربي حالياً مصحوباً بمجموعة خطوط حرة الاستعمال كما يمكنه استعمال عدد من الخطوط التي تأتي مع نظاٍم ويندوز مثلاً . كما هو الحال بالنسبة لنظام تيخ ، فإن العربي مجاني ولا يكلف مُستعمِله إلا عناء الاستعمال .

## 1 Introduction

The development of Arabi[1] was a response to the absence of a package that manipulates the Arabic script and fulfills the following requirements:

1. LATEX 2ε and Babel compliant, this combination format/package being the most widely used in our opinion when mixing different languages.
2. The possibility of using 8-bit input text including already existing Arabic texts, on different systems.
3. Able to use existing, commercial and free, beautiful Arabic fonts.
4. Free (as in freedom), meaning a license like the GNU GPL or LPPL.

Arabi comes with an extensive user manual; this article gives a general overview of the system.

## 2 Typesetting Arabic with TEX: the existing possibilities

TEX and the Arabic script have a long history.

One might imagine that enabling TEX to write in both directions Right-to-Left (R2L) and Left-to-Right (L2R) with an Arabic font suffices to typeset Arabic with TEX.

Unfortunately, although such an extended TEX may perhaps be used to typeset a R2L language like Hebrew, this is far from sufficient for a complex script like Arabic, where the shapes of the glyphs

[1] The name of the package should not be misunderstood. It is not designed to support only the Arabic language, but all languages that use the *Arabic script*. Technically speaking, for Babel, they will all be considered as *dialects* of Arabic.

depend on the context, and may take many forms (at least four forms for the majority of Arabic characters even in the simplest[2] cases).

Many early attempts have been made; they all relied on a preprocessor that does the contextual analysis (also known as the *shaping algorithm*).

One attempt, not widely known, due to Terry Regier from the University of California, Berkeley, dating from December 1990, relied on the famous macros of D. Knuth and P. MacKay:

```
%The lines below are from Knuth and MacKay
%     TUGboat vol.8, #1, page 14.
\font\revrm=xbmc10    \hyphenchar\revrm=-1
\catcode`\|=\active
\def|#1|{{\revrm \reflect#1\empty\tcelfer}}
\def\reflect#1#2\tcelfer{\ifx#1\empty\else%
\reflect#2\tcelfer#1\fi}
```

to do the reflection, after a preprocessor has done a rough contextual analysis.

The pioneering work by Knuth and MacKay [11], who implemented the TEX bidirectional algorithm (which is unrelated to the Unicode Bidirectional Algorithm; the latter implicitly chooses the directions of the text) and added to TEX the four primitives (\beginL, \endL, \beginR and \endR) made things much better!

Some early attempts were also carried out by Y. Haralambous, who used the new extended engine TEX--XET. This includes the non-free Al-Amal[3] (1992, [6]), and the free ArabiTEX [4] (April 1995).

The most widely used system at present is probably K. Lagally's ArabTEX [13]. It is a package for writing Arabic in several languages using the Arabic script. It consists of a TEX macro package and *one* Arabic Naskhi-like font. ArabTEX will run with Plain TEX and LATEX; and work with any TEX engine, because it uses *its own* bidirectional algorithm. So, no preprocessor is needed! This makes it a little slow but with today's computer power, this is not really a problem. Its real drawback lies in the fact that the macros apparently *depend heavily* on the glyphs of the font it uses, making it quite impossible to use any other fonts that may be available to the user.

For courageous users, there also exist two more powerful systems

- Ω by Y. Haralambous and J. Plaice, and
- XƎTEX by J. Kew, if you have the right system and the right fonts.

## 3 Arabic script specifics

The Arabic script is one of the most widely used scripts on earth. It dominates in Arabic countries, of course, but has a special place for all Muslims because it's the script used to write the Koran, the holy book of Muslims.

The Arabic script, like all other Semitic languages, is written from *Right-to-Left*.

Another important aspect of the Arabic script is that *no hyphenation* is needed, or allowed at all. So, *no hyphenation patterns* are needed for any languages that uses the Arabic script. In very old Arabic documents, words could be split after a non-connecting character, while characters that connect were never split. In modern Arabic, hyphenation is forbidden completely. This makes it more difficult to get justification when long words occur at the end of a line, but Arabic is also cursive and has (in modern fonts mimicking the handwritten forms) a special character called *kashida* or *tatweel* (*keshideh* is a Farsi word that means *stretch*) that may be used between adjoining characters to make the word become longer. An example is the following word: **مثال** that may be written to occupy longer **مثـــال** and longer **مـــثـــال** and much more longer space **مــثـــــــال**.

### 3.1 The Arabic alphabet

The Arabic alphabet is caseless, but most letters have either two or four forms. The different forms are used according to the letter's position in the word (*initial*, *medial*, *final* and *isolated*). The alphabet is constituted in its basic form by

- 28 *consonants* (29 if we count the *hamza*). But the number of 28 characters can exceed easily 1000 glyphs per font if all ligatures are present!

| Isolated | Initial | Medial | Final |
|---|---|---|---|
| ب | ﺑ | ﺒ | ﺐ |
| ج | ﺟ | ﺠ | ﺞ |
| ع | ﻋ | ﻌ | ﻊ |
| ه | ﻫ | ﻬ | ﻪ |

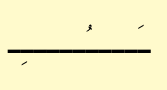
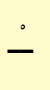
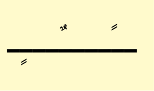
Table 1: Some characters' contextual forms

- Seven diacritical marks specifying the *vowels*. They are not used in typical Arabic texts but appear in poetry, textbooks for people learning the Arabic language, and some religious texts. They can be typed and then at the moment of compiling the document, can be either included or omitted according to the author's wish! The

[2] Through typographical simplifications. Some aspects of traditional Arabic typography are described in [5].

[3] We did not review it, as it was not available to the public as far as we know.

[4] The source and a DOS executable of the preprocessor were available through the French TUG.

three basic ones are called *fatha*, *damma* and *kasra*: ; the *sukun*  is used for the absence of vowels; and there are three *tanwin* forms written by doubling the three basic ones: .

The vowel marks are written somewhat like accents in the Latin script. Above, the drawn line represents the baseline, with the vowels that appear above the line being typeset above letters, while those below the line are typeset below letters.

### 3.2 Arabic typography

This aspect of Arabic merits much investigation and so much can be said about it. But in order not to be too lengthy, we will just cite three points.

In the classical Arabic literature, there are no typographical styles like bold, italic, etc. Different classical typefaces are used instead (req'a, naskhi, thuluth, etc.) to distinguish between different logical parts of the text. In modern literature, that depends heavily on computers made by people who are either unaware of the rules of Arabic typography or do not have enough time or money to develop such possibilities, we use more and more boldface and italics (slanted to the wrong side many times, unfortunately).

Concerning spacing and punctuation, there is a lot of change between books published early in this century by mechanical means and some more recent ones typeset using computer programs. It seems that different editors adopted different rules. Some use English or French rules, while others insert space before and after each sign — which was the rule in the older texts!

In general, in Arabic texts, enumerated lists use the `abjad` system using letters, in a particular order, instead of numbers, but numbered lists are used also.

## 4 The Arabi system

The two main problems faced when typesetting Arabic with TEX are managed by Arabi as follows.

1. The *bi-directional capability* supposes that the user has a TEX engine providing the four primitives `\beginR`, `\endR`, `\beginL` and `\endL`. This is the case with the TEX--XET and ε-TEX engines.
2. The contextual analysis does not need/use any pre-processor; this is done completely in the fonts, using the (quite limited) ligature possibilities of METAFONT.

This second point is the whole secret of Arabi's compatibility with most available packages. We tried to shorten TEX coding to deal with the specifics of the Arabic script as much as we could, to avoid eventual conflicts and clashes with other code.

The system is also *compatible* with *all other formats*, such as plain or ConTEXt. This too is because the whole contextual analysis is done in the fonts!

### 4.1 Input and font encodings

Typesetting Arabic and Farsi texts with TEX implies the use of special *input* and *output encodings*, so we need to use the standard packages `inputenc` and `fontenc`.

We use two special font encodings. For Arabic, we use `LAE` for *Local Arabic Encoding*, while for Farsi we use `LFE` that stands for *Local Farsi Encoding.* These two encoding *are not final.* Some character positions may change, and some empty slots will be filled with new characters.

Concerning the input encoding, the user simply creates an ordinary LATEX file, in which he can use 8-bit Arabic characters, typed visually on some system that supports the Arabic script.

For now, the Arabi system supports the following input code pages:

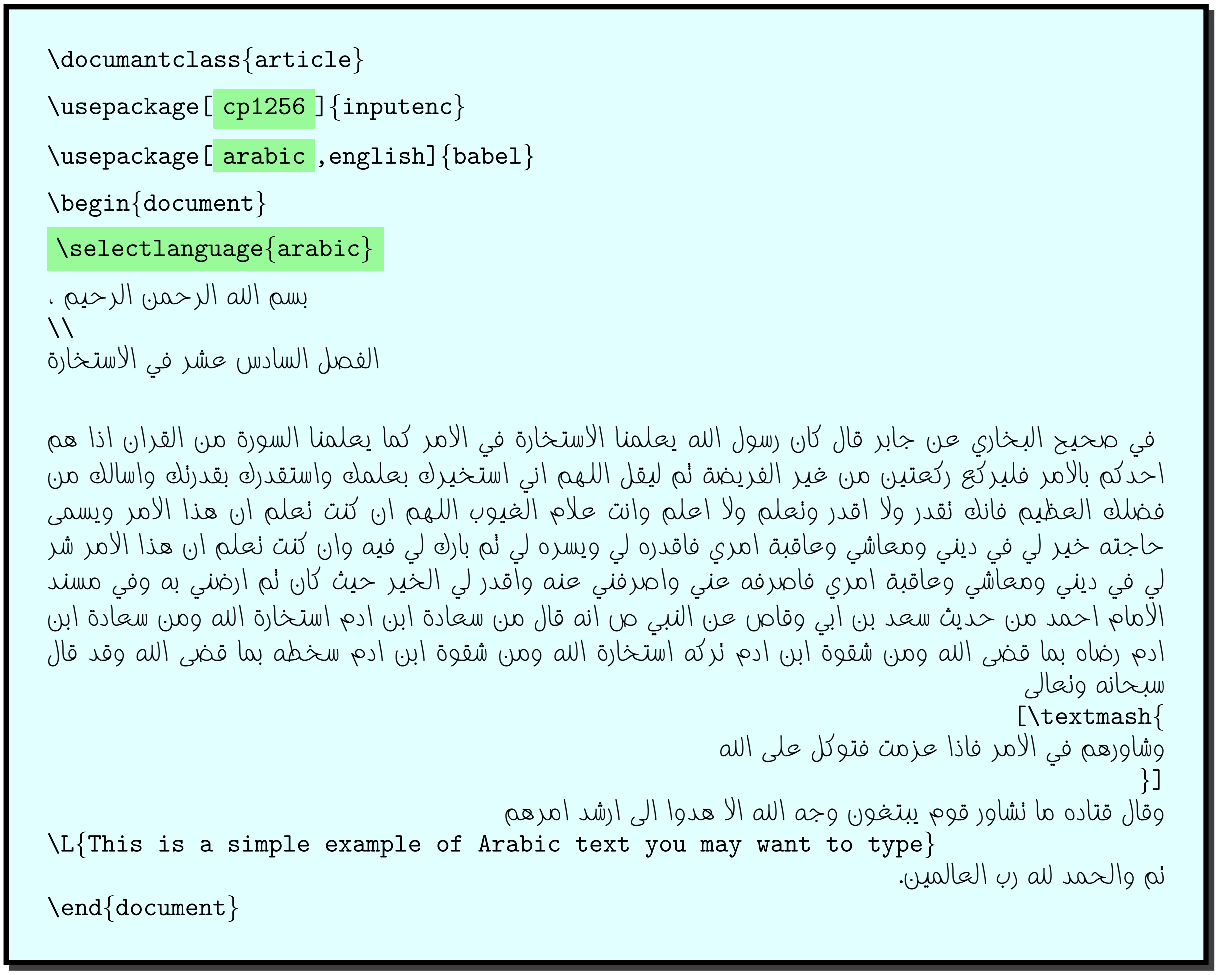
1. Arabic Windows CP-1256 for Arabic and Farsi.
2. ISO-8859-6 for Arabic, not suitable for Farsi because many Farsi characters are missing.
3. The multibyte Unicode UTF-8 (ISO-10646) for Arabic and Farsi.

### 4.2 What has been done so far?

Currently, with Arabi you can typeset correctly, while mixing the Arabic and Latin scripts, according to the context:

- Footnotes, appearing on the right side of the page.
- Lists, both itemized and enumerated. The standard `enumerate` environment uses the `abjad` system mentioned earlier.
- Floats are typeset with the right caption form and the appropriate entry is added to the table of contents.

Moreover, Arabi takes care of the bidirectional formatting of sectioning, chapters, (sub-)sections, etc., according to the context. And the tables (of contents, figures and tables) are typeset all according to one global text direction, which is the main text direction as specified by the user. This is meant to



```
\documantclass{article}
\usepackage[ cp1256 ]{inputenc}
\usepackage[ arabic ,english]{babel}
\begin{document}
\selectlanguage{arabic}
بسم الله الرحمن الرحيم .
\\
الفصل السادس عشر في الاستخارة

في صحيح البخاري عن جابر قال كان رسول الله يعلمنا الاستخارة في الامر كما يعلمنا السورة من القران اذا هم
احدكم بالامر فليركع ركعتين من غير الفريضة ثم ليقل اللهم اني استخيرك بعلمك واستقدرك بقدرتك واسالك من
فضلك العظيم فانك تقدر ولا اقدر وتعلم ولا اعلم وانت علام الغيوب اللهم ان كنت تعلم ان هذا الامر ويسمى
حاجته خير لي في ديني ومعاشي وعاقبة امري فاقدره لي ويسره لي ثم بارك لي فيه وان كنت تعلم ان هذا الامر شر
لي في ديني ومعاشي وعاقبة امري فاصرفه عني واصرفني عنه واقدر لي الخير حيث كان ثم ارضني به وفي مسند
الامام احمد من حديث سعد بن ابي وقاص عن النبي ص انه قال من سعادة ابن ادم استخارة الله ومن سعادة ابن
ادم رضاه بما قضى الله ومن شقوة ابن ادم تركه استخارة الله ومن شقوة ابن ادم سخطه بما قضى الله وقد قال
سبحانه وتعالى
[\textmash{
وشاورهم في الامر فاذا عزمت فتوكل على الله
}]
وقال قتاده ما تشاور قوم يبتغون وجه الله الا هدوا الى ارشد امرهم
\L{This is a simple example of Arabic text you may want to type}
تم والحمد لله رب العالمين.
\end{document}
```

Figure 1: Sample Arabi input

be the one that dominates your text, either an Arabic (script) document with small amounts of Latin text included, or a Latin one that contains Arabic.

Arabi has also a *limited*, but almost good, capability of vocalizing. Some more work needs to be done in that direction. Things would have been certainly better if METAFONT had more powerful ligature possibilities! But if you use XeTeX and have the *right* fonts, then things are certainly better.

The package also comes with extensive, and, we hope, clear documentation.

### 4.3 Current status

At the time of this writing, Arabi is at version 1.0, and already included in some distributions like MikTeX and BakomaTeX.

The latest version is always available from the CTAN archives. You should find it at `CTAN:tex-archive/language/arabic/arabi`

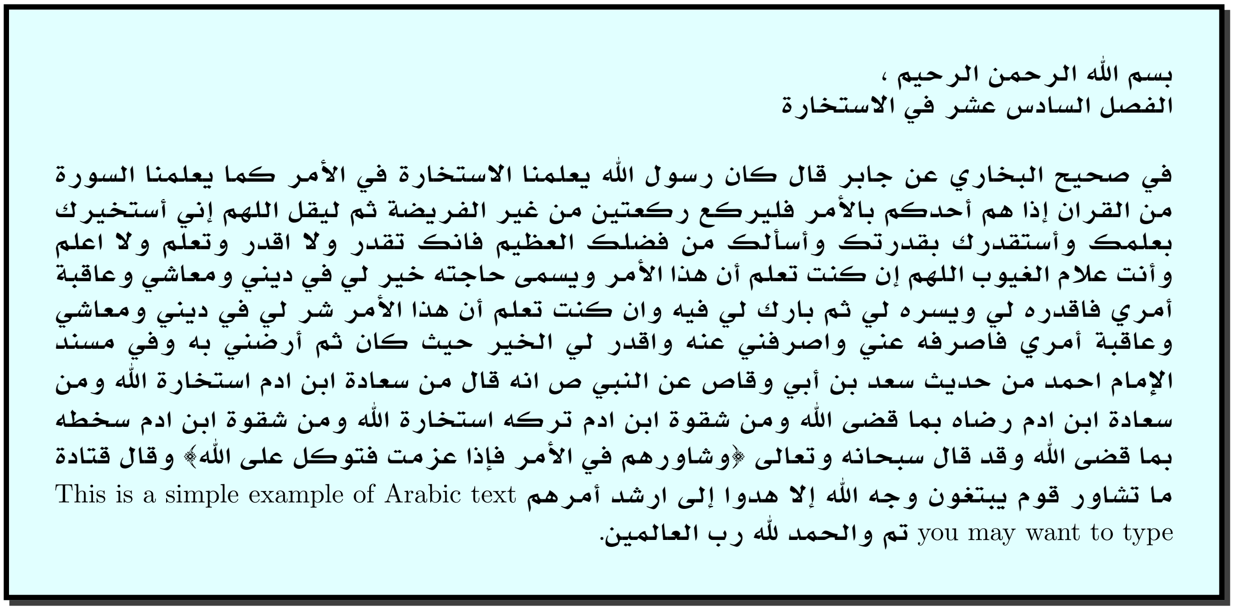
As mentioned earlier, the package is distributed under the LPPL, and has the status "author-maintained". It can be used *freely* (including commercially) to produce beautiful texts that mix Arabic with characters from other scripts.

Figure 1 shows a sample Arabi input document, and figure 2 the corresponding output.

### 4.4 Babel compliance

Arabi is fully LaTeX $2_\varepsilon$ and Babel compliant. It provides almost all the language-dependent strings for the *Arabic* and *Farsi* languages and can generate automatically the official *Jalali calendar*. The Farsi captions and the code for the Farsi date are from the FarsiTeX system.[5] Moreover, all Babel language switching commands apply.

[5] The FarsiTeX system seems unfortunately still not available with LaTeX $2_\varepsilon$. We hope that the Farsi support offered by Arabi and the Farsi fonts from the Farsi web project that come with Arabi will be useful to all Farsi users.



بسم الله الرحمن الرحيم ،
الفصل السادس عشر في الاستخارة

في صحيح البخاري عن جابر قال كان رسول الله يعلمنا الاستخارة في الأمر كما يعلمنا السورة من القران إذا هم أحدكم بالأمر فليركع ركعتين من غير الفريضة ثم ليقل اللهم إني أستخيرك بعلمك وأستقدرك بقدرتك وأسألك من فضلك العظيم فانك تقدر ولا اقدر وتعلم ولا اعلم وأنت علام الغيوب اللهم إن كنت تعلم أن هذا الأمر ويسمى حاجته خير لي في ديني ومعاشي وعاقبة أمري فاقدره لي ويسره لي ثم بارك لي فيه وان كنت تعلم أن هذا الأمر شر لي في ديني ومعاشي وعاقبة أمري فاصرفه عني واصرفني عنه واقدر لي الخير حيث كان ثم أرضني به وفي مسند الإمام احمد من حديث سعد بن أبي وقاص عن النبي ص انه قال من سعادة ابن ادم استخارة الله ومن سعادة ابن ادم رضاه بما قضى الله ومن شقوة ابن ادم تركه استخارة الله ومن شقوة ابن ادم سخطه بما قضى الله وقد قال سبحانه وتعالى ﴿وشاورهم في الأمر فإذا عزمت فتوكل على الله﴾ وقال قتادة ما تشاور قوم يبتغون وجه الله إلا هدوا إلى ارشد أمرهم This is a simple example of Arabic text you may want to type تم والحمد لله رب العالمين.

Figure 2: Sample Arabi output

### 4.5 Compatibility

The Arabi package has been tested successfully with packages such as `parshape`, `poster`, `pstricks` (and many of its derivatives), and, to our great surprise and pleasure, ArabTEX. It has been tested also on a Mac OS X system with the teTEX distribution and TeXShop (see figure 3).

### 4.6 Arabic fonts

One of the good features in Arabi is its ability to use any existing fonts that the underlying TEX engine can access. Arabi comes with a collection of Arabic and Farsi GNU fonts from, respectively, the Arabeyes and Farsi Web projects. The TFM files of some widely available commercial fonts are also included in the distribution, but the user still has to manage telling his engine where to find the corresponding font.

One remark to make here is that when preparing the vector encoding files for the different fonts, we learned that there is *no* standard. Even some corporations who produce and distribute applications and fonts that support the Arabic script for many years use so many names for the same glyphs that we arrived at the conclusion that one can never know what will be found when the font is opened!

## 5 Some bells and whistles

Arabi comes also with an *experimental* module that produces a *transliteration* of `Arabic texts`. No counterpart has been done for Farsi yet.

When texts are in general not fully vowelized, the transliteration cannot be expected to be correct. Moreover, when writing using some 8-bit input encoding there is absolutely no way to distinguish between long vowels ا ي و and the letters *alif*, *yaa* and *waw*. Neither, it is possible to write correctly the *hamza* when on *ʾalif*, *wāw*, or *yāʾ*.

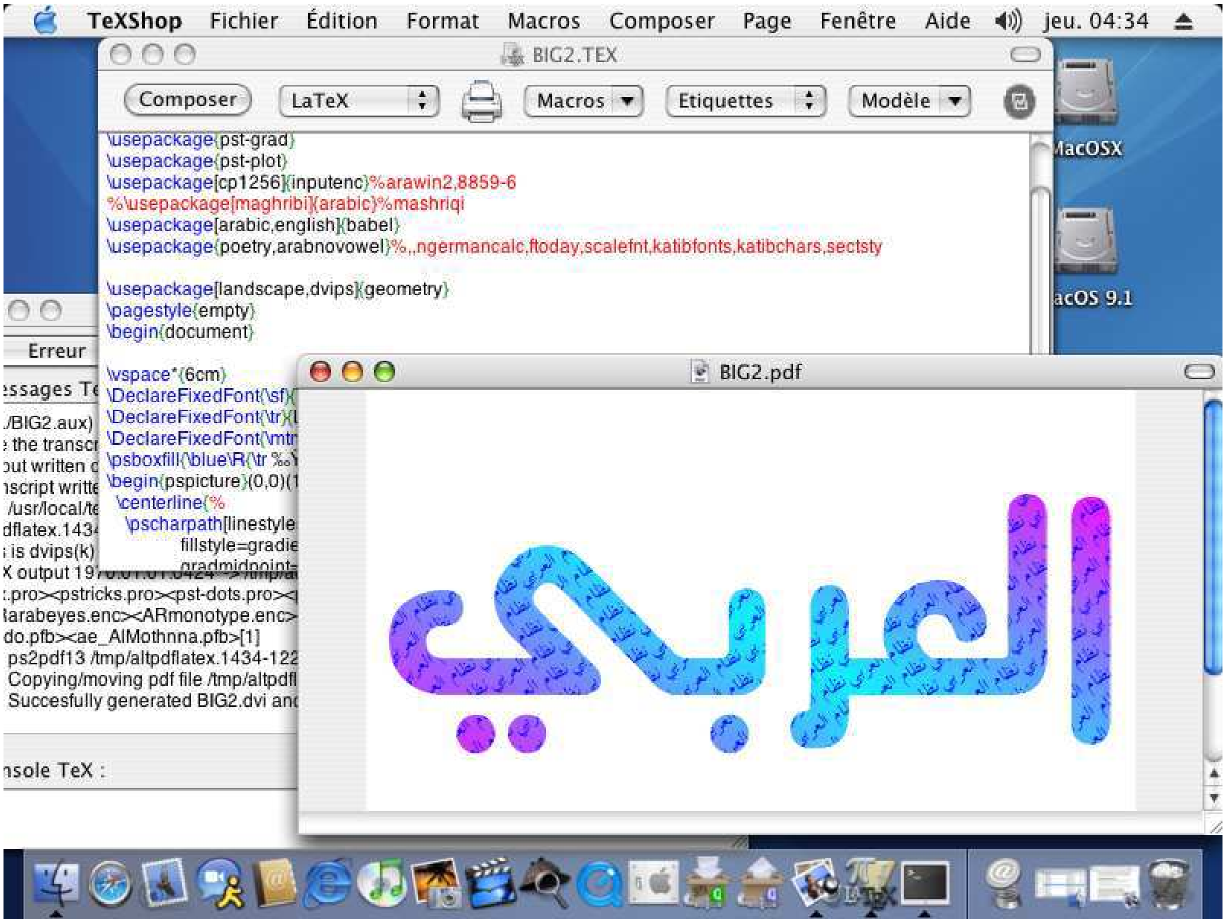
To use it, just load the package `translit` as with any other package, and type Arabic text in 8 bits in a Latin context, that is, without issuing a command that switches to the Arabic language.

| | | | |
|---|---|---|---|
| 1 | *ʾabw ālʿlāʾ ālmʿry* | أبو العلاء المعري | 1 |
| 2 | *matnuN mubārakuN* | مَتنٌ مُبارَكٌ | 2 |
| 3 | *ḥǧ mbrwr* | حج مبرور | 3 |

Table 2: A little example of transliteration

Classical poetry, in both Arabic and Farsi, is formatted in two "parallel" verses that begin and end at the same positions. When verses are too short, they are written closer to the (vertical) center of the page, as in the next example. Arabi relies on the same idea[6] of spreading the *keshida* glyph used by ArabTEX.

[6] A contribution by the author to ArabTEX a long time ago. ArabTEX uses a variable width horizontal "line" while we stack the *kashida* glyph the necessary number of times to get the right width!



```
\usepackage{pst-grad}
\usepackage{pst-plot}
\usepackage[cp1256]{inputenc}%arawin2,8859-6
%\usepackage[maghribi]{arabic}%mashriqi
\usepackage[arabic,english]{babel}
\usepackage{poetry,arabnovowel}%,,ngermancalc,ftoday,scalefnt,katibfonts,katibchars,sectsty

\usepackage[landscape,dvips]{geometry}
\pagestyle{empty}
\begin{document}

\vspace*{6cm}
\DeclareFixedFont{\sf}{
\DeclareFixedFont{\tr}{
\DeclareFixedFont{\mtr
\psboxfill{\blue\R{\tr %
\begin{pspicture}(0,0)(1
 \centerline{%
  \pscharpath[linestyle
           fillstyle=gradie
           gradmidpoint=
```

```
./BIG2.aux)
 the transcr
out written o
nscript writte
 /usr/local/te
dflatex.1434
 is dvips(k)
X output 1970.01.01:0424 -> /tmp/a
:.pro><pstricks.pro><pst-dots.pro><
Farabeyes.enc><ARmonotype.enc>
do.pfb><ae_AlMothnna.pfb>[1]
 ps2pdf13 /tmp/altpdflatex.1434-122
 Copying/moving pdf file /tmp/altpdfl
 Succesfully generated BIG2.dvi and
```

Figure 3: Arabi running on Mac OS

﴿ قال الطغرائي في لامية العجم ﴾

حب السلامة يثني هم صاحبه
عن المعالي ويغري المرء بالكسل
فإن جنحت إليه فاتخذ نفقا
في الأرض أو سلما في الجو واعتزل
لو كان في شرف المأوى بلوغ منى
لم تبرح الشمس يوما دارة الحمل
وشأن صدقك عند الناس كذبهم
وهل يطابق معوج بمعتدل
ملك القناعة لا يخشى عليه ولا
يحتاج فيه إلى الأنصار والخول
ترجو البقاء بدار لا ثبات لها
فهل سمعت بظل غير منتقل

Figure 4: Some Arabic poetry

## 6 The limits and the problems

The main limit seems to be the capacity of the TFM format:

- First in its 256-glyph limit, which certainly is not a sufficient number for a modern font, not to talk about an Arabic one!
- And second in the very limiting way it handles ligatures. In a script like Arabic, three-character ligatures are the rule, while there are even four letter ligatures, e.g., محمد. But if we also want to manage diacritics, which we can recall play in Arabic the role of vowels in Latin languages, things become even worse.

There is also an important ε-TEX issue, that the R2L direction is not supported in Mathematics. So we have to rely on some script *à la* Knuth and MacKay to reverse the characters and the words.

## 7 The future for Arabi

Concerning the future developments of Arabi. In these early times, we focus on keeping it alive and bringing it to maturity by correcting any bug that appears and completing the already existing functions, as no one is perfect. Let us cite the poet *al-mutanabbi* : **قال أبو الطيب المتنبي**

**ولم أر في عيوب الناس عيبـــــاً**
**كنقص القادرين على التمـــــام**

Please do not hesitate to forward suggestions, questions, or comments on Arabi. Thanks for your interest.

### References

[1] B. Esfahbod and R. Pournader. FarsiTEX and the Iranian TEX community. *TUGboat* 23(1), 41–45, 2002.

[2] J. Braams. Babel, a multilingual style-option system for use with LATEX's standard document styles. *TUGboat* 12(2), 291–301, 1992.

[3] J. Braams. An update on the Babel system. *TUGboat* 14(1), 60–61, 1993.

[4] Michel Goossens and Frank Mittelbach, with Johannes Braams, David Carlisle, and Chris Rowley. *The LATEX Companion*. Addison-Wesley, 2nd edition, 2004.

[5] Y. Haralambous. Towards the revival of traditional Arabic typography ... through TEX. Proceedings of the EuroTEX'92 conference, Prague, 1992.

[6] Y. Haralambous. Typesetting the Holy Qur'ān with TEX. Proceedings of the 2nd International Conference on Multilingual Computing (Latin and Arabic script), Durham, 1992.

[7] A. Hoenig. *TEX Unbound: Strategies for Fonts, Graphics, and More*. Oxford University Press, 1998.

[8] D.E. Knuth. *The TEXbook*. Addison-Wesley, Reading, MA, USA, 1986.

[9] D.E. Knuth. *The METAFONTbook*. Addison-Wesley, Reading, MA, USA, 1986.

[10] D.E. Knuth. Virtual Fonts: More fun for grand wizards. *TUGboat* 11(1), 13–23, April 1990.

[11] D.E. Knuth and P. MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat* 8(1), 14–25, April 1987.

[12] A. Lakhdar-Ghazal. *Caractères arabes diacritiques selon l'ASV-CODAR (pour imprimer les langues arabes)*. Institut d'Études et de Recherches pour l'Arabisation, Rabat, 1993.

[13] K. Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. Proceedings of the EuroTEX92 conference, Prague, 1992

[14] K. Lagally. *ArabTEX Arabic and Hebrew, (Draft) User Manual Version 4.00*. March 11, 2004.

[15] L. Lamport. *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994.

[16] P. MacKay. Typesetting problem scripts. BYTE 11(2), 201–218, 1986.

[17] The FarsiTEX Project. `http://www.farsitex.org/`

[18] The FarsiWeb Project. `http://www.farsiweb.info/`

[19] Institute of Standards and Industrial Research of Iran. `http://www.isiri.com`

[20] Microsoft. Free download of the Arabic font pack, `arafonts.exe`. `http://office.microsoft.com/arabicregion/Downloads/2000/arafonts.aspx`

[21] XƎTEX web site and mailing list. `http://scripts.sil.org/xetex`

[22] The Unicode standard. `http://www.unicode.org/`

# DadTEX — A full Arabic interface

Mustapha Eddahibi, Azzeddine Lazrek and Khalid Sami
Department of Computer Science,
Faculty of Science, University Cadi Ayyad
P.O. Box 2390, Marrakesh, Morocco
Phone: +212 24 43 46 49 Fax: +212 24 43 74 09
lazrek (at) ucam dot ac dot ma
http://www.ucam.ac.ma/fssm/rydarab

**Abstract**

This paper presents a TEX-based way to localize LATEX documents for natural languages with a script based on the Arabic alphabet. So, native speakers of such natural languages who do not know English can use and understand LATEX.

ضادتخ - واجهة عربية خالصة

**ملخص:** يعرض هذا المقال تعريبا كاملا لنظام تخ لتنضيد النصوص الشهير. يتيح هذا التعريب إمكانية تحرير النص المدخل بلغة عربية خالصة. الامر الذي يتيح للذين لا يستطيعون استعمال اللغة الانجليزية إمكانية استعمال برنامج ليتخ كنظرائهم والحصول على نصوص ذوات جودة تنضيد رفيعة على غرار ما يقدمه البرنامج الاصلي.

## 1 Introduction

Although the typesetting system TEX was originally designed especially for composition of mathematical documents, it has become a very fine system for typesetting documents in many other fields. The Arabic alphabet–based scripts are some of the linguistic contexts where the use of TEX has been progressively adapted. Many projects, including ArabTEX[1] [4], MlTEX,[2] Omega[3] [3], ε-TEX,[4] Aleph, and others, as multilingual systems, have been interested in typesetting documents containing simple Arabic text. The system RyDArab[5] has been dedicated to the composition of mathematical expressions in various notations used in Arabic, depending on the cultural context.

At first, TEX was intended for document composition in English. It was based on the ASCII encoding. To allow direct data entry for particular letters of Latin languages, a set of packages (inputenc, Babel,[6] ...) have been proposed. The application encTEX[7] [5], compatible with the standard 8-bit TEX, allows the encoding of input/output in UTF-8.

Initially, Arabic document composition was performed through Arabic/Latin transliterations. That is, the input of Arabic text was done with Latin characters corresponding to Arabic alphabet glyphs. The glyph transliterations are accompanied by the application of a set of contextual operations to find suitable glyphs. This is necessary because, besides its right-to-left writing direction, Arabic writing is cursive and letters have several shapes according to their positions in the word: initial, median, final, isolated (e.g., ج ـج ـجـ جـ). The Omega and ArabTEX systems were interested in direct Arabic text input; i.e., text is directly typed in an Arabic text editor. This operation is performed using transliteration tables to translate Arabic text from the text editor encoding to the encoding used by the typesetting system.

[1] ArabTEX is a multilingual computer typesetting system developed by Klaus Lagally: http://www.informatik.uni-stuttgart.de/ifi/bs/research/arab_e.html

[2] MlTEX was developed by Michael Ferguson

[3] Omega is a 16-bit enhanced version of TEX developed by John Plaice and Yannis Haralambous: http://omega.enstb.org/

[4] ε-TEX is an extended TEX including the features of TEX--XET, developed as part of the 𝒩𝒯𝒮 project by Peter Breitenlohner under the aegis of DANTE e.V. during 1992: http://www.ctan.org/tex-archive/systems/e-tex/

[5] RyDArab is a system for typesetting mathematics in an Arabic notation: http://www.ucam.ac.ma/fssm/rydarab/system/zip/rydarab.zip

[6] Babel provides multilingual support for TEX. It's developed by Johannes L. Braams: http://www.ctan.org/tex-archive/macros/latex/required/babel/babel.pdf

[7] encTEX allows full UTF-8 processing in standard 8-bit TEX. It's developed by Petr Olšák: http://www.olsak.net/enctex.html

Until now, however, there is no system that allows composing Arabic documents completely in Arabic. So, the development of a mechanism completely based on TeX to do so was an open question. Therefore, we developed an interface, called DadTeX,[8] that allows creation of LaTeX documents in Arabic. The goal was to allow Arabic users, who don't know English, to compose LaTeX documents containing *only* Arabic letters, with the addition of control characters like `\`, `$`, ... and of course with Latin text for bilingual documents.

Mathematical documents in an Arabic notation, composed with RyDArab and CurExt,[9] can so be typeset directly. In order to save the semantic meaning of mathematical commands and environments, it was natural to translate the RyDArab and CurExt macros. Of course, those translations are not enough. In TeX, besides the commands for mathematical objects, there are many commands that specify the mathematical content of the text, such as the `theorem` environment.

With these translations, Arabic users now hope that one need not use commands like `\chapter`, in a standard article. Clearly, if commands are also localized, it will be easy to understand their meaning; further, customization of command names allows TeX to be tailored to local pedagogical approaches.

Also, using English based commands with Arabic text is a source of ambiguity. Some problems with TeX source editing with bidirectional lines are:

Text selection: selections can be made either in logical or visual mode. Selections in logical mode are accompanied by discontinuities which is a direct consequence of bidirectionality and that leave the user feeling uncomfortable, while visual mode selections lead to content ambiguity.

Character positions: in a bidirectional line, the end and beginning of a run of Arabic text are visually in the same place.

Semantic ambiguity: characters without explicit direction inherit the direction of the preceding text. This misleads the reader of the source about the meaning of the expression. E.g., the command `\catcode‘\ب=11` will be displayed as the incorrect `\catcode‘\11=ب`.

With DadTeX,[10] such problems can be avoided by using only Arabic text and minimizing bidirectional text in source documents.

The RyDArab system has already made some steps in the way of TeX localization, by adding commands for automatic date generation with Arabic month names and Arabic numbers. Further steps to be done in this field include exploiting the linguistic and regional properties from the system settings and getting notational preferences (such as units, currency, ... ).

## 2 DadTeX system

### 2.1 Process

At the beginning of this work, we thought of creating an application to convert Arabic text in a source document into its transliterated version using the same correspondences as those used in the transliterations. The program would read the source text character by character and translate each element into its equivalent. When a control character is met, the program would operate differently, by seeking in a commands dictionary the equivalent of this command.

This mechanism is similar to the one used in FarsiTeX[11] [1]. FarsiTeX translates Persian text to transliterated text via an application program `ftx2tex`. Commands are written with Latin characters but are rendered from right to left. For example, `\begin{document}` is written as `{document}begin\`.

This method, based on the use of an application to translate a localized source file, presents several drawbacks. For instance, it is not direct; it generates a supplementary file to be processed instead of the original source file. So, the compilation time is increased. Likewise, additional memory and free space are required. Therefore, the method adopted in DadTeX is based on a direct use of TeX through translation of commands into Arabic. The source file is compiled directly. This makes it possible to avoid the use of a supplemental application that could eventually limit the use of DadTeX across platforms.

Such command name translation is not sufficient when using some systems, such as RyDArab.

[8] Dad is an Arabic letter. Its sound doesn't exist in many languages, showing the Arabic language's phonetic precision. In the Arabic literature, the Arabic language is often called the Dad language.

[9] CurExt is a variable-sized curved symbols typesetting system: `http://www.ucam.ac.ma/fssm/rydarab/system/zip/curext.zip`

[10] DadTeX is developed within the framework of global project "Al-khawarizmy", acting in the general field of localization of e-documents and tools: `http://www.ucam.ac.ma/fssm/rydarab/system/zip/dadtex.zip`

[11] FarsiTeX is a Persian/English bidirectional typesetting system: `http://www.farsitex.org/`

Indeed, RyDArab is based on transliteration of individual letter-based alphabetical symbols [2]. Therefore, it becomes necessary to redefine the correspondences using Arabic characters. This operation has to take into account the particular encoding used.

Various systems of numbers are used in Arabic alphabet-based writings: the Arabic numbers notation system called Arabic or occidental numbers, used in North African Arabic countries; Arabic-Indic numbers, used in Middle Eastern Arabic countries; and Persian numbers, used in Iran. In TEX, numbers are used to control document component sizes, an action's frequency, etc. In this work, only the syntactic aspect of numbers will be discussed, the semantic would be in the scope of a future DadTEX version.

The method used in DadTEX is not specific to Arabic language, or languages with an Arabic alphabet–based script. It can be generalized on one hand to any right-to-left writing system, and on the other hand to any non-ASCII encoded documents. This system would alleviate the learning burden imposed on non-English speaking people.

### 2.2 Structure

For purposes of simplification, the first version of DadTEX is intended to be used with ArabTEX or Omega. Documents using DadTEX should be encoded either in the ISO-8859-6 encoding system for ArabTEX or UTF-8 for Omega. Other encoding systems can be used, as long as they are compatible with ArabTEX or Omega.

DadTEX is composed of four files:

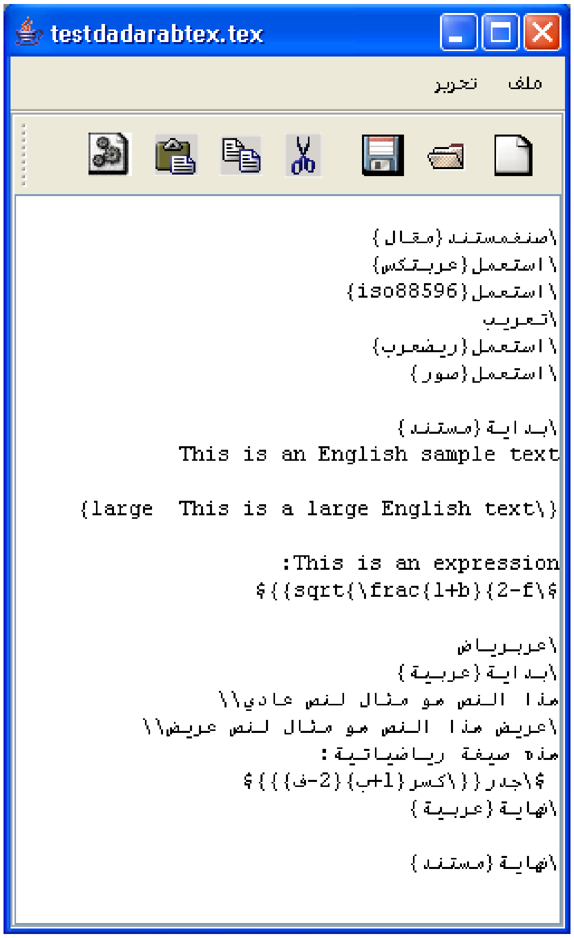
`dadalpha.tex`: here, ISO-8859-6 Arabic characters are declared as letters using the primitive `\catcode`. In TEX, command names are composed of a control character \ followed by characters with category code equal to 11.

`dadbody.tex`: a set of commands used in document bodies are declared.

`dadpreamb.sty`: a set of commands used in document preambles are declared. The encoding system in use is defined here.

`dadadapt.tex`: DadTEX users can add their own commands. It is like a dictionary containing vocabulary that can be augmented whenever new keywords are established. Putting them in a separate dictionary allows for flexible translation.

DadTEX is platform independent; it can be used in both Windows and Linux. It is extensible, i.e., the user can add new commands. It is also flexible; all command names can be modified. The core of TEX and LATEX are not modified. In fact, DadTEX operates like an interface between the Arabic script user and the typesetting system. The mechanism used makes it possible to translate command names for all TEX packages available without limitations.



```
\صنفمستند{مقال}
\استعمل{عربتكس}
\استعمل{iso88596}
\تعريب
\استعمل{ريضمرب}
\استعمل{صور}

\بداية{مستند}
This is an English sample text

{\large This is a large English text}

This is an expression:
$\sqrt{\frac{1+b}{2-f}}$

\عربرياض
\بداية{عربية}
هذا النص هو مثال لنص عادي\\
\عريض هذا النص هو مثال لنص عريض\\
هذه صيغة رياضياتية:
$\جدر{\كسر{1+ب}{2-ف}}$
\نهاية{عربية}

\نهاية{مستند}
```

Figure 1: Arabic document source using ArabTEX

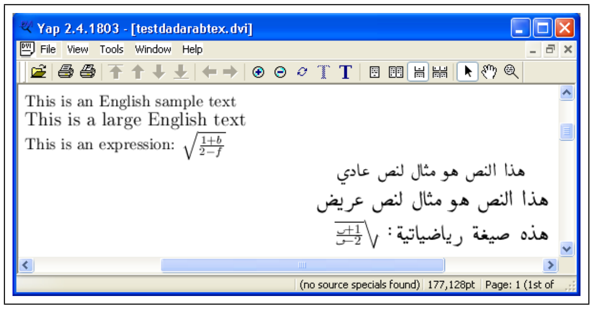
Figures 1–4 show examples of a TEX document both with and without DadTEX.

### 2.3 In action

Besides the DadTEX system, an adapted text editor, named DadNass,[12] has been developed to simplify typing of document source, calling TEX commands and running applications.

In spite of the Arabic interface for composition of TEX documents, some problems related to the debugger and to error messages remain. Error messages from the TEX core or from loaded packages are still displayed in English. Since DadTEX is based on

[12] `http://www.ucam.ac.ma/fssm/rydarab/system/zip/dadnass.zip`



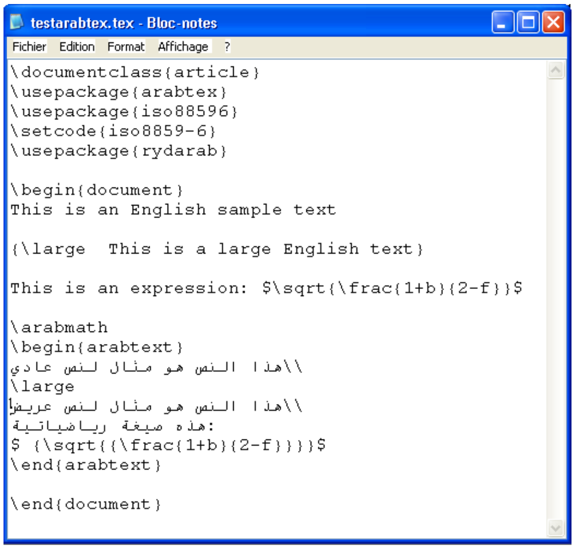
Yap 2.4.1803 - [testdadarabtex.dvi]

This is an English sample text
This is a large English text
This is an expression: $\sqrt{\frac{1+b}{2-f}}$

هذا النص هو مثال لنص عادي
هذا النص هو مثال لنص عريض
هذه صيغة رياضياتية : $\sqrt{\frac{1+ب}{2-ف}}$

(no source specials found) 177,128pt Page: 1 (1st of

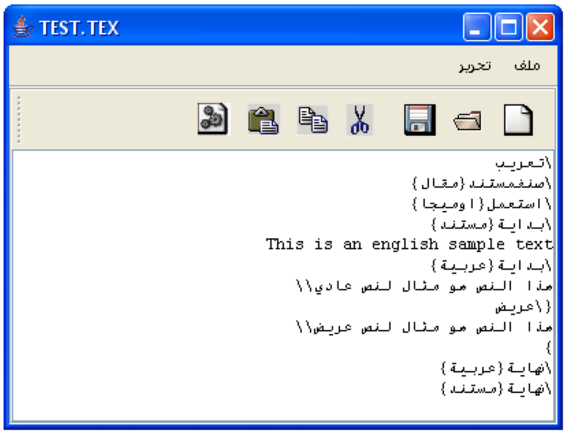
Figure 2: Output document in DVI



testarabtex.tex - Bloc-notes

```
\documentclass{article}
\usepackage{arabtex}
\usepackage{iso88596}
\setcode{iso8859-6}
\usepackage{rydarab}

\begin{document}
This is an English sample text

{\large  This is a large English text}

This is an expression: $\sqrt{\frac{1+b}{2-f}}$

\arabmath
\begin{arabtext}
\\هذا النص هو مثال لنص عادي
\large
\\هذا النص هو مثال لنص عريضا
هذه صيغة رياضياتية:
$ {\sqrt{{\frac{1+b}{2-f}}}}$
\end{arabtext}

\end{document}
```

Figure 3: Normal equivalent of the document source



TEST.TEX

```
\تعريب
\صنفمستند{مقال}
\استعمل{اوميجا}
\بداية{مستند}
This is an english sample text
\بداية{عربية}
هذا النص هو مثال لنص عادي\\
{\عريض
هذا النص هو مثال لنص عريض\\
}
\نهاية{عربية}
\نهاية{مستند}
```

Figure 4: Simple Arabic document source using Omega

translation of command names only, error messages are always the same.

## 2.4 Encoding

DadTEX can be used with any encoding system capable of representing the Arabic alphabet, as long as the encoding is supported by the packages in use. In this version, we used ISO-8859-6 instead of UTF-8 because the ArabTEX system still has some problems when it is used with UTF-8. The encoding system recommended[13] to represent Arabic text under Unix is ISO-8859-6.

In ISO-8859-6, Arabic characters are encoded in one byte. It is thus easy to set their category code into 11. The assignment is done by using a text editor that supports ISO-8859-6 encoding (e.g., `\catcode‘\ب=11`). If the encoding used to encode Arabic text is UTF-8, then Arabic characters are presented using two bytes, and the use of a text editor that supports UTF-8 will lead to errors, because `\catcode` is intended to be used with only one-byte characters. For example, instead of using the command as follows `\catcode‘\ا=11`, characters should be divided into two visible bytes (Ø and § in the case of the Alef) using an ASCII text editor. Thus, we use the following: `\catcode‘\Ø=11` and `\catcode‘\§=11`. However, in the case of the Omega system, the hexadecimal code can be used directly: `\catcode‘^^^^0627=11`.

Currently available multilingual TEX typesetting systems require declaring a text's linguistic environment. For example, the Arabic text begins with the command `\begin{arab}` and ends with `\end{arab}`. This way allows TEX to use the suitable font and change the writing direction. This is in contrast with browsers, where no specific declaration of the language is required, thanks to an advanced level of use of Unicode's bidirectionality algorithm, and to the possibility of using available fonts instead of being tied to only one font. This is due to the multilingual nature of TrueType fonts. Indeed, all Unicode characters can be presented in a single TrueType font.

## 2.5 Document structure

The structure of DadTEX documents is similar to that used in LATEX.

The document starts with the Arabic command `\صنفمستند`, equivalent to `\documentclass` with the translation of the class name (e.g., `article` to

[13] "Arabization of graphical user interfaces", by Franck Portaneri and Fethi Amara: `http://www.langbox.com/staff/arastub.html`

مقال ). These classes and options specify the basic structure for a document.

Next, we can find commands like \استعمل, equivalent to the command `\usepackage`, which instructs LaTeX to load some external commands gathered in a package.

One can possibly use other commands that will be applied before `\begin{document}`, for example, commands that act on the document layout.

The command \بداية{مستند} is the Arabic translation of the command `\begin{document}` which marks the end of the preamble and declares, as its name indicates, that the document starts here.

The DadTeX system is not intended to be limited to Arabic script composition packages; all TeX commands can be translated. Indeed, a document can simultaneously contain text in Arabic and in other languages. For this reason, the beginning of the document is distinguished from the beginning of the Arabic environment. The command \بداية{عربية} is used to declare the beginning of the Arabic environment. For each command \بداية (`\begin`) there is a corresponding \نهاية (`\end`) command that indicates the end of the environment.

For the Omega-based DadTeX version, the first line of the document is not the Arabic equivalent of the command `\documentclass`. Instead, the Arabic source begins with the command \تعريب, equivalent to the commands that change ΩCP list, which will allow to conversion of the input document encoding into that used by the font, e.g., OT1 or T1. However, there are still some technical problems with using Omega and DadTeX with RyDArab to compose Arabic mathematical expressions.

### 2.6 Compilation

Instead of directly compiling the source document composed by the user, we use the combination of following compilation options:

```
latex \RequirePackage{adapt}
      \input myfile.tex -job-name myfile
```

## 3 Conclusion

DadTeX allows the TeX user who does not know English to use command names and parameters written in his mother tongue. So, TeX can be more user-friendly and understandable and, therefore, more widely used among Arabic-speaking people. Conversely, such users can thus be more concerned with and interested in TeX development.

In a future version of DadTeX, it will be very interesting to add support for the Arabi system [6]. Indeed, compared to ArabTeX, Arabi can be used with other packages with fewer restrictions.

**Acknowledgements**:

Thanks to Barbara Beeton and Karl Berry for their editorial corrections and contributions.

### References

[1] Behdad Esfahbod and Roozbeh Pournader. FarsiTeX and the Iranian TeX Community. *TUGboat* 23:1, pages 41–45, Proceedings of the 2002 TUG Annual Meeting, Trivandrum, India, 2002.

[2] Mostafa Banouni, Azzeddine Lazrek et Khalid Sami. Une translittération arabe/roman pour un e-document. In *$5^e$ Colloque International sur le Document Électronique*, pages 123–137, Conférence Fédérative sur le Document, Hammamet, Tunisi, 2002.

[3] Yannis Haralambous and John Plaice. Multilingual Typesetting with Ω, a Case Study: Arabic. In *Proceedings of the International Symposium on Multilingual Information Processing*, pages 137–154, Tsukuba, 1997.

[4] Klaus Lagally. ArabTeX — Typesetting Arabic with vowels and ligatures. In *EuroTeX'92*, Brno, Czechoslovakia, 1992.

[5] Petr Olšák. Second Version of encTeX: UTF-8 Support. *TUGboat* 24:3, pages 499–501, Proceedings of the EuroTeX 2003 Meeting, Brest, France, 2003.

[6] Youssef Jabri. The Arabi system — TeX writes in Arabic and Farsi. In this volume, pp. 147–153.

# AlQalam for typesetting traditional Arabic texts[*]

Hossam A. H. Fahmy
Electronics and Communications Department,
Faculty of Engineering, Cairo University, Egypt
hfahmy (at) arith dot stanford dot edu

### Abstract

AlQalam ("the pen" in Arabic) is our freely available system intended for typesetting the Qur'an, other traditional texts, and any publications in the languages using the Arabic script. From a typographical point of view, the Qur'an is one of the most demanding texts. However, there is a long historical record of excellent quality materials (manuscripts and recent printings) to guide the work on a system to typeset it. Such a system, once complete, can easily typeset any work using the Arabic script, including those with mixed languages.

## 1 Characteristics of Arabic typography

The Arabic alphabet has been adopted for use by many languages in Africa and Asia, including Arabic, Dari, Farsi, Jawi, Kashmiri, Pashto, Punjabi, Sindhi, Urdu, and Uyghur. The Arabic script is also used for a number of other languages either to present how the language used to be written historically (as for Turkish) or how some write it in an unofficial manner (as for Hausa in western Africa).

Similar to the Latin alphabet, with its adoption by several languages, the Arabic alphabet has acquired new symbols to represent the sounds that do not exist in Arabic. In contrast to the Latin alphabet that has dots only on the 'i' and 'j', the Arabic alphabet uses dots extensively, both above and below the letter shapes, to distinguish the different characters. This explicit distinction between the different letters in written text using dots was in itself an addition to the original script, which had no dots. In the original Arabic script, the distinction between بيت (house) and بنت (girl) when represented as ٮٮٮ was understood from the context. In general, the symbols developed for other languages follow the same idea as Arabic and use more dots (up to four) and special marks on the original shapes of the Arabic letters.

Arabic being a semitic language, usually only the consonants are written in a word. The equivalent of short vowel sounds are written as additional marks on top of the letters. Obviously, the different languages have different vowels and need different symbols to mark them. In addition to that, since the geographical area covered by the Arabic script historically is quite vast, different regions of the world developed different symbols. The result that we see today is a plethora of additional marks developed historically.

A beginner learns that Arabic is written from *right to left* and must practice writing each letter and its connection rules to other letters. Because of this cursive nature, any letter may connect to the previous and following letters. Hence, a beginner learns the general *four basic forms* of a letter: at the start of a word, at the middle, at the end, and isolated. The simplest example of this rule is the equivalent of 'b' in Arabic: بـ ـبـ ـب ب. A reader with a sensitive eye will notice that the four shapes of the same letter differ in their width, height above the line, and depth below it. The same structural shape is used for the equivalent of 't' (تـ ـتـ ـت ت), and 'th' (ثـ ـثـ ـث ث). The equivalent of 'n' and 'y' share the same shapes as 'b' in the initial and medial forms (نـ ـنـ and يـ ـيـ) but not in the final or the isolated forms (ـن ن and ـي ي). In many writing styles, both traditional calligraphic scripts and typographical fonts, the final and isolated forms of 'y' are written without dots as (ـى ى).

In traditional Arabic writing styles (but with the exception of thuluth, riqāʿ, and tawqīʿ [12]), the letters ا د ر و and their siblings with dots or marks do not connect to the following letter but only to the preceding one.

The cursive nature of the Arabic script adds another characteristic: many letters combine together to produce new shapes as in الحج becoming الحج. In the Latin script this phenomena occurs infrequently and when it happens, a *ligature* is used to improve the appearance of the problematic letter combinations such as 'ff' and 'ffl'. In Arabic typography, on the other hand, the presence of combined letters is abundant *but optional* in many cases.

[*] A project under the author's supervision. This paper combines the material presented at two conferences: EuroTEX 2006 and TUG 2006.

Haralambous [1] gives a long list (still not exhaustive) of possible ‘ligatures’ in Arabic. While speaking about the history of Arabic typography, Milo [8] explains that “each letter can have a different appearance in *any* combination, something that can only be crudely imitated with ligatures”. According to Milo [8], most modern books present the connected letter groups “as ‘ligatures’ and ‘artistic expressions’ without so much as a hint at traditional morphographic rules”. In 1990, MacKay [7] discussed the range of context evaluation in Arabic and concluded that clusters of four, five, six, and sometimes more letters may combine into a unique shape. MacKay then proposed the use of virtual fonts with TEX as an adequate solution.

Another feature in the Arabic script is its reliance on subtle changes to the letter shape to aid the reader in identifying the beginning and end of each letter within a combination. The letter سـ has three “teeth” (vertical pen strokes) similar to the teeth in بـ and تـ. When سـ is connected to بـ, the tail of the سـ may be elongated to alert the reader to the correct grouping of the teeth. Furthermore, the two words سبع and تسع present different heights for the teeth of the بـ and تـ to help the reader as well. This difference in width and height is a type of encoding to prevent a misreading of the word and to aid the trained eye in quickly catching the letter combination.

That encoding helps in other cases as well. If for any reason the dots fade away, a reader faced with ٮسع can guess its correct origin. This encoding to emphasize the different letters by raising some teeth is essential in words such as تبينت and تسببت. However, the raising of the teeth is only possible by investigating the group of letters. So the height of the tooth for ـنـ in تبينت and تبنت is not a feature of the individual letter but of the whole combination. The same goes for the height of the dot on top of that same letter as in سننجح or سنن .

In traditional (manual) writing, the “skeleton” of the letter combination is written first, then the dots and the other marks are provided. So, a writer probably progresses from the skeleton to the dotted to the vocalized form as سٮں → سنن → سُنَنٌ .

In the Arabic script, a good calligrapher justifies the lines mainly by using optional ligatures or wider forms of some letters and not by stretching the spaces between the words, as is done with the Latin script. The use of optional ligatures and wider forms is the preferred method in high quality works. Another method is to add an elongation to the tail of some letters by using the taṭwīl or kashīdah symbol ‘ـ’ such as ســــبب instead of سبب. This second method has been widely abused in newspapers and low quality materials using mechanical typewriters.

The Arabic script has a large number of writing styles that were developed historically to accommodate the different languages and different purposes. Latin scripts use bold, italic, or larger fonts for section headings and for emphasis. Traditional Arabic writings vary the typeface instead. The printings of the Qur’an as well as of most books almost always use the naskh typeface for the main body. The headings, the introductory materials, and the back materials frequently use other typefaces such as the thuluth, ta‘līq, and ruq‘ah.

To summarize, the Arabic script has its own unique requirements for typesetting:

1. Arabic is written from right to left.
2. Characters in general have four different forms (initial, medial, final, and isolated).
3. These forms are of different width, height, and depth.
4. The shape of a specific form depends on its context. For example, the height of the teeth (the vertical stroke at the start of the character) changes to help the reader to distinguish this character from its neighbors.
5. There are additional marks (mostly for short vowels) that are put on top or below the character.
6. The horizontal and vertical location of the dots and marks on the characters is not always at the same position but depends on the character *and* its context.
7. Almost any letter of the script may enter into a ligature and those ligatures may be up to six letters long.
8. Ligatures and variable width forms of the letters are used to justify the lines.
9. Several typefaces are needed for special materials in a work of good quality.

With all of these issues, to find a suitable position for the dots and marks on the letter combinations is sometimes a real challenge even for a human, let alone a machine.

## 2 Automated typesetting of Arabic

The Arabic script does not enjoy the same luxury that Latin script has when it comes to automated typesetting on computers. Milo correctly asserts [8] that the use of individual letters as the building block is not suitable for Arabic. Both MacKay [7] and Milo [9] argue that a layered approach is a better solution. In such a layered approach, some basic elements are provided in the font to represent the skeleton of some letter combinations, an individual letter's skeleton, or even a part of a letter. These elements are combined first to give the correct skeleton of the word with all the needed shaping for the teeth or other style requirements. On top of that skeleton, a second layer for the dots is added. Then, the vowel marks and any other marks come in subsequent layers. Milo [10] developed a system with a layered approach for his company, DecoType. It is a proprietary system used by a number of commercial software tools. Due to its proprietary nature, the full details and the extent of the capabilities of this system are not widely known.

Our goal is to provide a freely available system capable of typesetting the Qur'an, other traditional texts, and any publications in the languages using the Arabic script. From a typographical point of view, the Qur'an is one of the most demanding texts. However, there is a long historical record of excellent quality materials (manuscripts and recent printings) to guide the work on a system to typeset it. Such a system, once complete, can easily typeset any work using the Arabic script including those with mixed languages.

Knuth and MacKay [4] were the first to present a working solution for including right-to-left text (for Arabic and Hebrew) in the TEX family. Their proposed TEX-XET system is an extension of TEX that produces an extended DVI file. The enhanced mode of $\varepsilon$-TEX allows bidirectional text processing and produces regular DVI files, but $\varepsilon$-TEX does not provide any Arabic fonts or any specific functionalities that ease the typesetting of Arabic books. Within the TEX extensions, both Ω [3] and ArabTEX [5, 6] have been used for Arabic and have met some of the basic requirements to varying degrees.

With the historical trend to extend TEX, Ω evolved as an implementation allowing multilingual text processing. As an offshoot of the work on Ω, the Al-Amal system [1] was an early attempt to typeset the Qur'an specifically. Unfortunately, it is not freely available and its output (as shown in the example published in the paper describing it) falls short of the desires of a native reader.

Due to its various attractive features, Ω was the first choice to achieve our goal. However, the result of our early experiments with the available Arabic font provided with Ω and with the system itself were not satisfactory. Ω in its current state does not easily lend itself to the layered approach described earlier. The modification of Ω is not an easy task since it is a very large system and such a modification means the creation of a new system that is not compatible with the existing base of TEX. The newer developments to Ω [2] — once they are stable, widely available, and documented — should be a great help in implementing the layered approach necessary for typesetting high quality texts in Arabic.

Lagally in ArabTEX [6] preferred to stay within the stable TEX standard and perform all the necessary processing with TEX macros. That decision allowed ArabTEX to be portable to any TEX implementation. However, ArabTEX had to compromise on the issue of line breaking. Although not a simple program, ArabTEX is confined to a number of style files each performing a specific task. ArabTEX implements a layered approach where each character is represented by a skeleton and additional modifiers (dots and vowels). For high quality work, the font provided with ArabTEX still needs improvements but it is an acceptable start. ArabTEX uses the LATEX license and hence we changed the name of our work to AlQalam ('the pen' in Arabic).

## 3 Implementation

As a start, AlQalam grows out of modifications to ArabTEX [5, 6]. Hence, it inherits ArabTEX's good features:

- All the necessary processing is done with TEX macros which allows it to be portable to any TEX implementation.
- Although still in need of many improvements, the font provided with ArabTEX is the best available within the TEX family.
- The shapes of the letters change with their context (teeth are raised and automatic detection of many ligatures).
- A layered approach is used.

Although the use of TEX macros brings the virtue of portability it also brings severe limitations:

- ArabTEX had to compromise on the issue of line breaking and justification. For right-to-left text, ArabTEX is forced to handle the line breaking itself, using a slow and complicated algorithm, thus bypassing one of the best parts of TEX for the Latin script.
- ArabTEX analyzes the character combinations to decide on ligatures using TEX macros as well. This analysis using macros is
  - less efficient than the ligature tables used for the Latin script in METAFONT; and

   - limits the extent of the search for alternative letter combinations. In Arabic, four, five, six, and sometimes more letters may combine into a unique shape [7].

## 4 Specific needs of the Qur'an

Our goal of typesetting the Qur'an and traditional texts implies a few more challenging requirements in addition to those for general Arabic typesetting. To assist the reader in recitation, several indicators for vowels, joints, text structure, and pausing locations have historically been added to the text of the Qur'an. We present here a few symbols.

**Signs of pause (علامات الوقف) :**

- The ۗ sign: the reader may continue but it is better to pause.
- The ۖ sign: it is allowed to pause but it is better to continue.
- The waqf jā'iz sign ۚ: equally good to pause or continue.

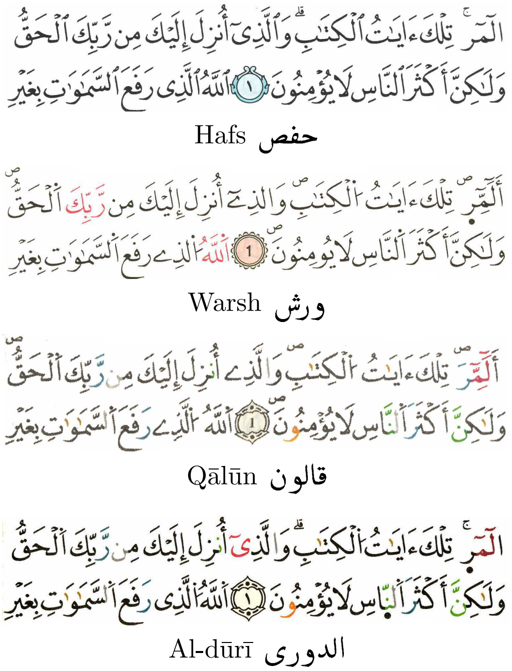
**Additional diacritics :**

- Ra's khā' ـۡـ corresponds to sukūn.
- The madda 'ۤ' appears in the Qur'an on many letters such as in كٓهيعٓصٓ.
- The small 'ۦ' in ﴿أَن يُشۡرَكَ بِهِۦ﴾ and 'ۥ' in ﴿وَٱللَّهُ عِندَهُۥ حُسۡنُ﴾.

Furthermore, there are different "narrations" of the Qur'an that differ in the pronunciation in some locations and hence lead to a plethora of additional marks needed. The vast majority of the printed copies of the Qur'an are in the narration known as Hafs. Only three other narrations (with their special marks for the special pronunciations) are printed in the whole Muslim world. The remaining narrations (sixteen remaining for a total of twenty) are still in manuscript form.

Fig. 1 shows an example of the four narrations that exist in print. To make the comparison easier, we present the same two lines from the four narrations written by the same calligrapher. A simple look at the first word (top right in each example) reveals some of the different symbols needed. Those additional symbols fit well in a layered approach but would be quite difficult to accommodate otherwise. The ۚ symbol appearing on the first word of the topmost narration is a pausing sign.

The use of transliteration for a purely Arabic document that is several hundred pages long is obviously neither practical nor desirable. Hence, the default assumption for AlQalam is an input file with the characters coded in Unicode. Bi-directional editors such as emacs and gedit (we used both) are good options. Editors have their own limitations though. If a symbol has a Unicode point associated with it but there is no key combination mapped to that symbol or no glyph in the editor's font to represent it, another facility must be used. In a case such as ۖ the user might supply the Unicode value as `^^db^^96`. AlQalam interprets this code as belonging to the "signs of pause" category, then raises it to its correct position such as in ٱلۡحَقُّۖ. The improvement of the input method is one of the major steps in future developments.



Hafs حفص

Warsh ورش

Qālūn قالون

Al-dūrī الدوری

**Figure 1**: The first two lines of surat al-ra'd: an example from the four printed narrations.

Another feature of typesetting the Qur'an is the use of colors. In some printings, certain letters, marks, or sometimes complete words take a different color usually to remind the reader of a pronunciation rule. Educational texts for young children also often use color encoding schemes to stress new reading concepts and to help train their eyes in picking up the distinctive features of the script. A complete system for dealing with the Arabic text should be able to color a piece of a letter combination or some specific marks.

## 5 New features in AlQalam

The first version of AlQalam, which became available by the end of 2005 and was presented at EuroTEX 2006, introduced many additional symbols to the font to enable the typesetting of quotations from the Qur'an. It also added an additional layer in typesetting Arabic text for the pausing marks of the Qur'an. Three features must exist in this layer:

- the correct vertical and horizontal positioning of those marks on the underlying word;
- the ability to stack some marks on top of each other; and
- the scalability of those marks when the size of the underlying text is scaled.

The first version of AlQalam implemented the concept of the additional layer but was deficient in regards to the three features just mentioned.

### 5.1 Positioning the marks

The different pausing marks vary in their sizes and shapes. They are raised on top of words that vary in their heights as well. The second sample from the top in Fig. 1 has four words followed by the same pausing sign ص. Its vertical position in

أُلْحَقُّ and ٱلْكِتَٰبِ

is different because of the underlying text.

Two primitive algorithms existed in the initial version of AlQalam.

1. Raise the pausing sign at a predefined height from the baseline regardless of the height of the underlying text. The worst case is when a sign with a descender such as ج comes on top of a diacritic mark on a high letter. With a fixed height we get:

   الٓمٓۚ صَبَرُواْۖ تَرَوْنَهَاۖ الْقَمَرَۖ → الٓمٓ صَبَرُواْ تَرَوْنَهَا الْقَمَرَ

   using this first method.

2. Raise the pausing sign by a fixed height *above* the diacritics on top of the character. This second method results in

   بِرَبِّهِمْ تَرَوْنَهَا الْقَمَرَ أُلْحَقُّ → بِرَبِّهِمْ تَرَوْنَهَا الْقَمَرَ أُلْحَقُّ

   where ص has a varying vertical position.

Human calligraphers, in contrast, use neither of these methods. In the current implementation, we attempt to come closer to what is done in the best of the art. Calligraphers never lower the pausing marks below certain limits. Hence, AlQalam now starts by raising any pausing sign a minimum height depending on the current font size. If the underlying text is high enough so that an overlap occurs, the pausing mark is raised further. It is important to note that AlQalam now allows more than one mark to appear on top of a word. The new algorithm is not linked to the diacritic mark, as in the second method mentioned above, but can handle *any* underlying text, be it a diacritic mark or another pausing mark. The new algorithm thus yields:

بِرَبِّهِمْ تَرَوْنَهَاۖ الْقَمَرَۖ الٓمٓۚ ٱلْكِتَٰبِۗ أُلْحَقُّ صَبَرُواْۖ

for the case of a single mark and

أَعْمَىٰۚ مَّرْقَدِنَاۜۗ

for the case of multiple marks.

As for the third new feature of scaling, if the user writes

```
\RL{\vsmaller الْقَمَرَ^^db^^96 \larger
الْقَمَرَ^^db^^96 \larger الْقَمَرَ^^db^^96 \larger
الْقَمَرَ^^db^^96 \larger الْقَمَرَ^^db^^96 \larger
الْقَمَرَ^^db^^96 \larger الْقَمَرَ^^db^^96 \larger
الْقَمَرَ}
```

the output is

الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ الْقَمَرَۖ

which is easily achieved, since the height at which the pausing mark is positioned depends on the font size.

It is obviously not desirable to type sequences such as `^^db^^96` throughout the input file. An editor capable of understanding user-defined shortcuts may be used to ease this task. The user can just type the shortcut key and the editor puts the correct UTF-8 code into the file. Furthermore, to help all users, we also assigned some shortcuts for marks that appear frequently in the Qur'an, such as the dagger alif ـٰ for which the user types '`!`' instead of '`^^d9^^b0`'. Our system now translates the '`!`' and the other shortcuts on the fly to the corresponding UTF-8 code before processing the file. Here are the shortcuts currently enabled in AlQalam:

| type | ← | mark | type | ← | mark | type | ← | mark |
|---|---|---|---|---|---|---|---|---|
| 3 | ← | ـۚ | 2 | ← | ـۗ | 1 | ← | ـۖ |
| 6 | ← | ـ۟ | 5 | ← | ـۙ | 4 | ← | ـۘ |
| 9 | ← | ـۛ | 8 | ← | ـۭ | 7 | ← | ـۢ |
| ! | ← | ـٰ | . | ← | ـ̣ | 0 | ← | ـ۠ |
| ^ | ← | ۩ | * | ← | ۞ | + | ← | ۝ |

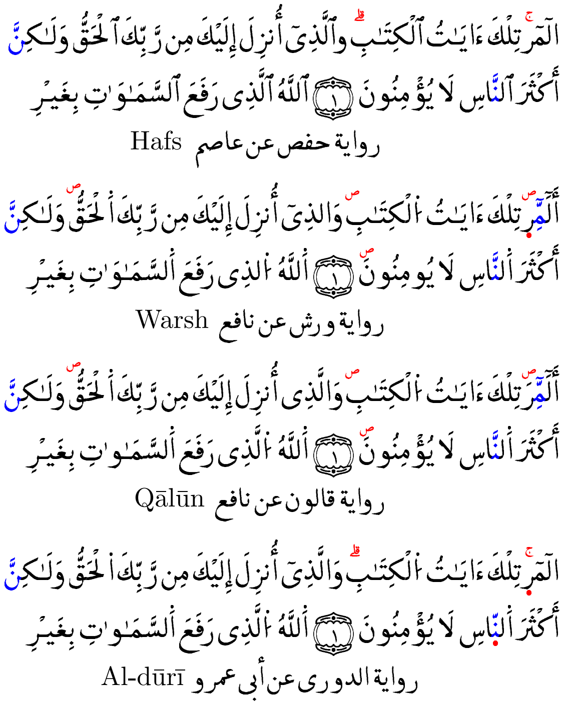

Hafs رواية حفص عن عاصم
Warsh رواية ورش عن نافع
Qālūn رواية قالون عن نافع
Al-dūrī رواية الدورى عن أبى عمرو

**Figure 2**: The same first two lines of surat al-ra'd as in Fig. 1, typeset with AlQalam.

Fig. 2 shows the same quotes presented in Fig. 1 as typeset by the current version of AlQalam. This example reveals a few more new features. The interword spacing in high quality Arabic typography is much smaller than in Latin-based typography. In some cases it is even completely absent. The reader relies on the fact that letters at the end of a word have a different shape in order to separate the words. A minimal spacing within a right-to-left environment is the default now, compare:

هذا مثال للمسافات المتروكة بين الكلمات العربية

to

هذا مثال للمسافات المتروكة بين الكلمات العربية

The command `\newspacefalse` was used in the second case to retain a large spacing as in ArabTEX and regular Latin script.

The earlier version of AlQalam allows hamzat-alwasl (ٱ) to appear only at the start of a word. However, in the Qur'anic text, it may appear in a medial form as in

وَٱلَّذِى    فَٱدَّٰرَءْتُمْ    وَٱتَّقُواْ    بِٱسْمِ

which is now possible in the current version.

Another new feature concerns the positioning of diacritic marks on top of the letters. Compare أَسْمَاء to أَسْمَاء. The first is the default, while the second with the raised mark is achieved by the command `\hightrue` within the Arabic script environment.

The current algorithm handles the small dagger alif according to its context. It is considered a separate character that appears on its own in cases such as ٱلسَّمَٰوَٰتِ ءَايَٰتُ رَوَٰسِىَ. On the other hand, it is considered a mark on top of the underlying character in cases such as ٱسْتَوَىٰ ٱلصَّلَوٰةَ ٱلْحَيَوٰةَ. If the dagger alif is a mark, its positioning on the character is similar to that of the short vowels.

The contextual analysis for the dagger alif is conceptually simple, although a bit elaborate to program using TEX macros. The dagger alif is a mark modifying the underlying character if it is

- on the 'final' ى as in ٱسْتَوَىٰ, even if it is followed by a connected pronoun as in سَوَّىٰهُ تَقْوَىٰهَا زَكَّىٰهَا, or
- on a و which is pronounced as an ا as in ٱلرِّبَوٰاْ or ٱلصَّلَوٰةَ.

## 5.2 Coloring

The coloring scheme has been improved as well. In Fig. 2, the color of every م or ن that has a shaddah on top of it is different, to indicate its special pronunciation in the Qur'an. The user chooses the color by `\colmnshadd{`*colorname*`}` (a few other commands for various rules were also added). The issue of coloring while maintaining the contextual analysis to decide on the correct form (initial, medial, final, or isolated) of the letter is not easily done.

In the Latin script, it is easy to get 'text' via '`te\textcolor{blue}{x}t`' — as long as no special ligatures or kerning are needed between the 'x' and the 'e' or 't'. However, in Arabic script the command sequence in the middle of the word breaks the contextual analysis and the ligature formation.

In our case, since the coloring rules are known a priori, we code them in the system. Thus, after all the contextual analysis to decide on the appropriate letter form and the ligatures is done, but before the complete word is fed to the output, we intervene and check for the existence of the requested letter sequence. If a part of the word matches the pattern, we color it. Such a hard-coded "programmatic" way is not suitable for arbitary coloring that the user may wish to introduce into regular text outside of Qur'anic quotations.

## 5.3 Other improvements

A large number of glyphs in the current font are improvements of what ArabTEX or the first version of AlQalam provided. ArabTEX uses a circular pen for the diacritic marks. We found that a rotated

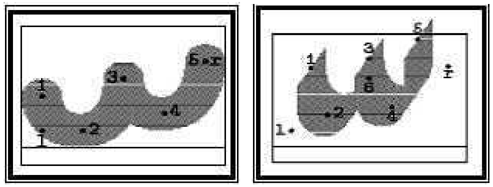
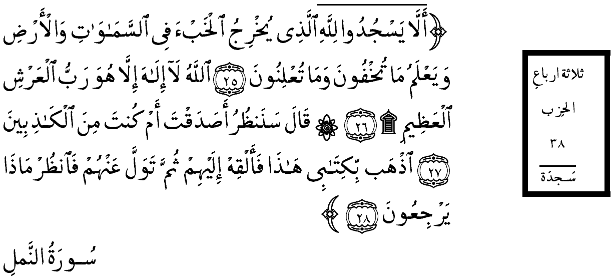


**Figure 3**: Effect of changing the pen and improving the shape on the "shaddah".



﴿أَلَّا يَسْجُدُوا لِلَّهِ ٱلَّذِى يُخْرِجُ ٱلْخَبْءَ فِى ٱلسَّمَٰوَٰتِ وَٱلْأَرْضِ
وَيَعْلَمُ مَا تُخْفُونَ وَمَا تُعْلِنُونَ ٢٥ ٱللَّهُ لَآ إِلَٰهَ إِلَّا هُوَ رَبُّ ٱلْعَرْشِ
ٱلْعَظِيمِ ۩ ٢٦ ۞ قَالَ سَنَنظُرُ أَصَدَقْتَ أَمْ كُنتَ مِنَ ٱلْكَٰذِبِينَ
٢٧ ٱذْهَب بِّكِتَٰبِى هَٰذَا فَأَلْقِهْ إِلَيْهِمْ ثُمَّ تَوَلَّ عَنْهُمْ فَٱنظُرْ مَاذَا
يَرْجِعُونَ ٢٨﴾

ثلاثة ارباعِ
الحِزب
٣٨
سَجدَة

سُورَةُ النَّملِ

**Figure 4**: Marginal notes indicating partitions and prostrations.

square pen gives a much more satisfying output, as shown on the right side of Fig. 3.

To mimic the printed versions of the Qur'an, we define a `\sura` command ("sura" is a chapter of the Qur'an) that produces:

مَكيّة سُورَةُ العاديات ١١ ءَايَة

and

مَكيّة سُورَةُ الإخلاص ٤ ءَايَات

when given the number of the sura (100 in the first case and 112 in the second). AlQalam provides default values for the remaining information to be displayed (number of verses and whether the sura was revealed in Makkah or Madinah) if they are not supplied by the user.

Fig. 4 shows verses with marginal notes indicating the partitioning and the location of a prostration, as is customary in printings of the Qur'an. The counter of the partition and the corresponding note are produced automatically when the user writes `*` in the file. The indicator and the note for the prostration are similarly produced by `^` in the input file.

## 6 Future work

Much more work is needed on the fonts to produce new typefaces and to enhance the current one. The production of multi-letter ligatures with a layered approach where the dots, vowels, and additional marks are stacked on the basic structure is still an open issue. It might even require changes to the way TEX and METAFONT (or other font generation tools) handle ligature tables.

The use of TEX macros for programming has its merits and problems as discussed earlier. We think that we have brought AlQalam quite close to the limits of such an approach. To handle line breaking and justification correctly, a much more fundamental change in TEX itself is needed. After describing the line breaking algorithm of TEX [11], Plass and Knuth propose a refinement where the badness function for the lines depends on the number of varying-width letters in the paragraph. Neither TEX nor its descendants have implemented this refinement. In the case of the Arabic script, it will not be just the varying-width letters but also the optional ligatures that may be formed or broken to change the length of the text on the line. We hope that one of the current projects to extend TEX ($\varepsilon$-TEX, $\Omega$, XeTEX, Oriental TEX, …) will include this change to the badness calculation. Much experimentation using several different languages will be needed to come up with the most suitable algorithm.

Another issue that requires more work is the contextual analysis to decide on the glyphs used. This analysis is not only within a word; it must sometimes look at two consecutive words. The following example shows why such an inter-word analysis is needed. In general in Arabic, a silent 'n' sound is pronounced normally before

ء ه ع ح غ خ

and goes through some form of vocal assimilation into the sound of the following letter otherwise. If the silent 'n' sound is at the end of a word in the form of a tanwin (for example), and the following word starts by a letter into which the 'n' assimilates, the tanwin will be changed from ـٌ ـً ـٍ to ـٌۢ ـًۢ ـٍۭ if the following letter is ب or to ـࣱ ـࣰ ـࣲ otherwise. The first letter of that following word gets a shaddah on it in the case of a full assimilation and does not get the shaddah for the incomplete assimilation.

For a program, these rules mean that we must do our analysis across any intervening spaces or command sequences, including counters for the verses and indicators of prostration or partitions that might come between consecutive words. The inter-word analysis is needed in many cases, not just for the 'n' sound, and it has some implications on the coloring rules as well.

As might be expected, our first attempts to achieve that endeavor using TEX macros proved to be quite laborious and are not yet fruitful. Currently, the user chooses the appropriate shape from the font manually. Once more we hope that the future TEX extensions can come to our help by providing easier means to program such an analysis.

## 7 Conclusions

In this article, we provided a summary of the traditional Arabic typesetting requirements as well as the first steps of a system to fulfill them. From these requirements, it appears that a layered approach is mandatory for high quality typography. We hope that future changes to TEX and METAFONT will enable us to achieve better ligature formation, line justification, and contextual analysis. To the best of our knowledge, there is no other software system available to serve the requirements of the different Qur'anic narrations.

## References

[1] Yannis Haralambous. Typesetting the holy Qur'an with TEX. In *Multi-lingual computing: Arabic and Roman Script: 3rd International conference*, Durham, UK, December 1992.

[2] Yannis Haralambous and Gábor Bella. Omega becomes a sign processor. In *EuroTEX 2005: Proceedings of the 15$^{th}$ Annual Meeting of the European TEX Users, Pont-à-Mousson, France*, pages 8–19, March 2005.

[3] Yannis Haralambous and John Plaice. Multilingual typesetting with Ω, a case study: Arabic. In *Proceedings of the International Symposium on Multilingual Information Processing, Tsukuba*, pages 63–80, March 1997.

[4] Donald E. Knuth and Pierre A. MacKay. Mixing right-to-left texts with left-to-right texts. *TUGboat*, 8(1):14–25, 1987.

[5] Klaus Lagally. ArabTEX: A system for typesetting Arabic. In *Multi-lingual computing: Arabic and Roman Script: 3rd International conference*, page 9.4.1, Durham, UK, December 1992.

[6] Klaus Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. In Jiří Zlatuška, editor, *EuroTEX 92: Proceedings of the 7th European TEX Conference*, pages 153–172, Brno, Czechoslovakia, September 1992. Masarykova Universita.

[7] Pierre A. MacKay. The internationalization of TEX with special reference to Arabic. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 481–484, November 1990. IEEE catalog number 90CH2930-6.

[8] Thomas Milo. Arabic script and typography: A brief historical overview. In John D. Berry, editor, *Language Culture Type: International Type Design in the Age of Unicode*, pages 112–127. Graphis, November 2002.

[9] Thomas Milo. Authentic Arabic: A case study. right-to-left font structure, font design, and typography. *Manuscripta Orientalia*, 8(1):49–61, March 2002.

[10] Thomas Milo. ALI-BABA and the 4.0 Unicode characters. *TUGboat*, 24(3):502–511, 2003.

[11] Michael F. Plass and Donald E. Knuth. Breaking paragraphs into lines. In Donald E. Knuth, editor, *Digital Typography*, pages 67–155. CSLI Publications, Stanford, California.

[12] Mohamed Zakariya. أنماطالحرفالعربي. *Al-Computer, Communications and Electronics Magazine* الكمبيوتروالاتصالاتوالإلكترونيات, 22(8):48–53, October 2005.

# Infrastructure for high-quality Arabic typesetting

Yannis Haralambous
Département Informatique, ENST Bretagne
CS 83 818, 29 238 BREST Cedex 3
France
yannis dot haralambous (at) enst dash bretagne dot fr

## 1 Introduction

This paper presents what we consider to be the ideal[1] (or at least a first step towards the ideal) infrastructure for typesetting in the Arabic script. This infrastructure is based on four tools which have partly been presented elsewhere:

1. the concept of *texteme*;
2. *OpenType* (or *AAT*) fonts. In fact in this paper we will talk about "super-OpenType" which consists in using static OpenType substitutions and positionings in dynamic typesetting;
3. $\Omega_2$ *modules*: transformations applied to the horizontal node list before entering the main loop;
4. an extended version of TEX's line-breaking graph for dynamic typesetting.

*Textemes* have been presented in [12] and [13]. They are atomic units of text extending the notion of character. A texteme is a collection of key-value pairs, some of which are mandatory (but may have an empty value) and others optional or freely definable by the user. Mandatory keys are "character" (a Unicode position), "font" (a font identifier) and "glyph" (a glyph identifier in the given font). The latter two keys are, in some sense, the common part between textemes and TEX's "character nodes" (quoted because in fact a "character node" contains only glyph-related information). Hence, a first application of textemes is to add character data to "character nodes". In fact there are other types of information which we also add to textemes:

- hyphenation: instead of using discretionary nodes, we add hyphenation-related information into textemes;
- color;
- horizontal and vertical offset of glyph (leaving the abstract box "width × (depth + height)" unchanged);
- metadata;
- etc.

We will see how — due to the internal structure of the Arabic writing system — the concept of texteme happens to be particularly useful for text in the Arabic script.

There is no need to present *OpenType* and *AAT* fonts; the reader can consult [20] or the corresponding Web pages at Microsoft and Apple.

$\Omega_2$ *modules* have been presented in [14]. The well-known technique of $\Omega_1$ Translation Processes ($\Omega$TPs) is applied at a later stage of text processing inside $\Omega_2$: just before the *end_graf* procedure, which is called when the complete list of nodes of a paragraph has been stored in memory, before we enter into the line breaking engine which will examine this list of nodes and insert active nodes at potential line breaks.

$\Omega_2$ will output the horizontal list of nodes, in XML. Since we are using textemes, this horizontal list contains both traditional types of nodes as well as texteme nodes. External processes will then transform these XML data and the result is again read by $\Omega_2$ and replaces the original horizontal list.

Besides the nodes of the horizontal list, $\Omega_2$ also includes in the XML data global information such as font name mapping, the current language, etc.

In the next section we will describe the fourth tool, namely the extended TEX graph.

## 2 Dynamic typesetting and an extended graph

When TEX processes a horizontal list, inserts active nodes for potential breakpoints, calculates badnesses for each arc of the graph and finally finds the shortest path (where we consider badness as a "distance"), glyphs do not change:

- if glyphs are given by "character nodes" then they are static;
- if they are given by discretionary nodes, then we have two possibilities: when there is no line break we have a single node list (possibly containing one or more glyphs) and when there is a break we have two entirely different lists, the "pre-break" and the "post-break". The choice

[1] The author has written several papers on the typesetting of Arabic: [6, 7, 8, 9, 10, 15, 16], and has developed Arabic systems using three different methods: a preprocessor (1990), intelligent ligatures (*ArabiTEX*, using TEX--XET, 1992) and $\Omega_1$ Translation Processes ($\Omega_1$ distribution and *Al-Āmal*, 1994). The system described in this paper will be the fourth (and hopefully the last) Arabic system developed by the author.

depends entirely on the fact whether we break or not;

- if they are given by ligature nodes, once again we have two possibilities: when the ligature is not broken then we have a single static glyph. When the ligature is broken we return to "characters" (or at least to something which is a bit closer to the concept of character, even though it is not exactly a character) and apply the main loop again to the two parts (before and after the break), which sometimes results in new ligatures. But once again each node list obtained that way is unique.

In all three cases glyphs either do not change or their change depends only on a line break close to them.

*Dynamic typesetting* is a method of typesetting where glyphs can change during the process of line breaking, for reasons which may depend on macrotypographic properties such as justification of the line or of the entire paragraph, or more global phenomena like glyphs on subsequent lines touching each other or to avoid rivers, etc.

Dynamic typography was applied by Gutenberg in his Bibles. He was systematically applying ligatures to optimize justification on the line level. It is very useful for writing systems using words but not allowing hyphenation, like Hebrew (where some letters have large versions without semantic overload, these letters have been used mostly at the end of lines, when printers realized that they are facing justification problems) or Arabic.

To perform dynamic typesetting with $\Omega_2$ we are extending the graph of badnesses so that we can have many arcs between two given nodes, each one corresponding to a given width ("width" in the sense of glue, that is a triple — ideal width, maximal stretch, maximal shrink) and to its badness.

Using an extended graph means applying the same principle of optimized typesetting on the paragraph level to paragraphs where glyphs (or glyph groups) have alternative forms (in the case of groups we call them "ligatures"). Performing the calculation of shortest path on such a graph means that the solution will be the best possible paragraph, chosen among all possible combinations of alternate forms of glyphs.

Alas, such a calculation can explode combinatorially. Imagine a paragraph of ten lines, each containing 60 glyphs, that is 600 glyphs in total. Imagine each glyph having two variant forms, that is a total of three choices for each glyph. No ligatures. That would already make $3^{600} \approx 1.87 \cdot 10^{286}$ possible combinations of glyphs, enough for running $\Omega_2$ until the next big bang and beyond, for only ten lines of text ... a perspective which would delight Douglas Adams if he was still with us.

This is why dynamic typesetting requires a *strategy*. Even if choices of glyph variants are mutually independent, one has to define rules to limit the number of glyph combinations.

For example, a realistic strategy could be the following:

1. classify glyphs into $N$ width classes (the higher $N$ the finer the result will be, but the more calculations we will have to do);
2. whenever we have to choose between glyphs in the same class, make a single choice, in a random manner;
3. if we have many choices for which the sum of the widths of classes is relatively constant, choose a single one, randomly.

In other words, we restrict the number of choices to those that give us different widths on the word level. Whenever we have different glyph combinations producing the same global width, we use a random generator to choose a single combination. This strategy is useful when there is no semantic overload. One can imagine refined versions where the combination chosen is not entirely random but is based on more-or-less strict criteria (for example: use ligatures preferably at the end of words, or do not use specific variants in the same word, etc.).

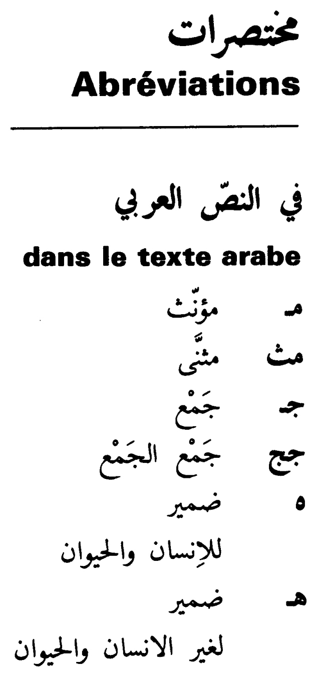
The strategy we have described is based solely on width criteria. But when many different glyph versions are designed the chances that some of them are in conflict (for example, may touch each other) heavily increases. Due again to combinatorial reasons, we can't expect the font designer to anticipate all possible conflicts between glyphs. We need a tool which will test each combination (for example, test whether glyphs are touching) and eventually add an additional penalty to the corresponding arc of the graph.

This tool can work on an interline level so that the calculation of global badness is more complex than just summing up the badnesses of individual arcs. The extended TEX graph used by $\Omega_2$ will use binary arithmetic on flags to add additional badnesses to given path choices.

Up to now, very few fonts exist with extremely many variants (one of which is, for example, *Zapfino* which has ten different 'd' letters) and in the case of calligraphic fonts the document's author (who becomes a "calligrapher") will probably be more interested in (manually) choosing a beautiful combination of glyphs than in having the absolutely best justification by leaving the choice of glyphs to the machine.

The case of Arabic is different: ligatures are much more common (especially in traditional writing styles) and calligraphers have a long tradition of using them in the frame of a justification-oriented strategy (see [3]). This is why the extended graph of TEX will prove especially useful for the Arabic script.

In the following we describe the infrastructure necessary for each step of Arabic text processing.



مختصرات
**Abréviations**

في النصّ العربي
**dans le texte arabe**

مـ مؤنّث
مث مثنًّى
جـ جَمْع
جج جَمْع الجَمْع
ه ضمير
للإِنسان والحيوان
هـ ضمير
لغير الانسان والحيوان

**Figure 1**: Abbreviations taken from a French-Arabic dictionary [1]. Notice on line 5 letter *meem* in initial form مـ instead of isolated form م which should normally be used since it is not followed by any other letter. The same happens with letter *heh* on line 11.

## 3 Infrastructure for Arabic text processing

### 3.1 Preliminaries: Dynamicity of Arabic script

Arabic text justification can be obtained by (in order of priority):

- using blank spaces of variable width (as in other scripts);
- enabling or disabling ligatures;
- choosing between alternative forms of glyphs;
- inserting "keshideh" connections between letters (in systems like [3] and [4] which can produce curvilinear connections; when the "keshideh" is simply a straight line segment, its esthetic value is very doubtful).

Article [3] mentions some additional justification methods (curvilinear baseline, typesetting the last words as interlinear annotations, etc.), which, in our humble opinion, are less suited for the visual paradigm of printed text and fall into the category of manuscript constructions.

### 3.2 Preliminaries: Texteme properties

The Arabic script functions in such a way that a lot of information is unwritten and has to be known in advance by the reader:

- Some letters are systematically connected and hence can take up to four different forms according to their immediate context (in Urdu script we have up to nine contextual forms). The form of a glyph can be calculated by contextual analysis, but in some cases this calculation must be overridden: for example, as we see in fig. 1, an initial letter *meem* مـ is used as an abbreviation for *mou'annith* مؤنّث (= female). This contradicts contextual rules: that *meem* normally should be isolated. Unicode provides a solution for this case: to use a ZERO-WIDTH JOINER character just after the *meem*. We consider Unicode's solution to be particularly clumsy: a "character" is (according, once again, to Unicode) the "abstract representation of a smallest component of written language" and this can hardly be said for the ZERO-WIDTH JOINER as used here. Furthermore, when doing copy-and-paste operations one should be sure to copy that invisible character, otherwise the *meem* will change form. Instead, $\Omega_2$ inserts Arabic contextual form inside the texteme, as a texteme property.

  In operating systems there is a dedicated library (Uniscribe for Windows, Pango for Linux, ATSUI for Apple) which performs contextual analysis and then transmits the result to an OpenType font. This is why OpenType provides a property for each contextual form (`init` for initial, `medi` for medial, `fina` for final, `isol` for isolated). In our case, the texteme properties play the rôle of OpenType property activators.
- Short vowels, although not always written, can be very useful for NLP (Natural Language Processing) applications (indexing, automatic translation, summarizing, etc.). People like Ahmed Lakhdar Ghazal in proposing a simplification of the Arabic script ([10], [18]) consider nonetheless that vowels should always be explicitly written in Arabic, to avoid ambiguities.

Tools like Sakhr's *Diacritizer* [19] or other tools described in [17] can provide the missing vowels. One can imagine an $\Omega_2$ module based on one of these technologies for adding textemes for the missing vowels (all vowels are represented by Unicode characters) with a "hidden" property activated.

Some people may consider full vowelization as archaic. It is true that the average Arabic reader does not need vowels to understand a text, except for rare cases (foreign words, etc.), in which it remains customary to use vowels. By using "visually hidden" textemes we can provide linguistically rich text without changing the visual image of the text and hence people's reading practices.

- Most words of Semitic languages are based on three-letter stems called *roots*. To analyze a word morphologically it is mostly sufficient to find its root and to consider the vowels which are added to the three letters and eventual prefixes and postfixes. In particular, being in possession of these data is the ideal condition for proper indexing of Arabic and the first step for pertinent automatic translation.

  Removing prefixes from Arabic words is necessary even for alphabetical sorting: it would be silly to sort *liradzul* under letter 'l' since it means in fact "for (*li-*) a man (*radzul*)" and the lemma should be *radzul*.

  Once again the solution is provided by textemes: using an NLP tool one can attach properties such as "first/second/third letter of semitic root", "prefix", "postfix", etc., to textemes. The sorter/indexer can then use them to operate properly.

- In some cases we are not sure about some of the information described above: ancient Arabic texts have no dots on letters (so that, for example, letters *beh*, *teh*, *theh*, *noon* and (in initial and medial form) *yeh* are written in exactly the same way), and even fewer short vowels or other diacritics. Reading such a text requires significant human interpretation (as in all ancient languages, but even more because of this particular aspect of Arabic script).

  Let us suppose that a scholar considers that a given letter of his manuscript has a 70% chance of being intended by its author as a *beh*, a 29.999% chance of being a *teh* and a chance in a million to be a *theh*. How can we insert this information into the text itself? Once again, we can use texteme properties. The same method can be applied for missing vowels. And one could develop various visual strategies for representing such textemes (color, hypertext links with pop up windows, etc.).

- Let us leave the semantic area aside and delve again into purely visual issues. Arabic letters can be connected by curvilinear segments called *keshideh*. But the amount of keshideh authorized between two letters is specified by rules (see [5], [9] and [16]). A special texteme property can be used to store the amount of keshideh allowed (for example, a floating number between `0` and `1` or a glue field) after a given texteme.

  Let us note the fact that keshideh has sometimes a semantic overload: it can be used to emphasize or to show metric properties of verses in poetry. In that case the glue field of the keshideh property will have a non-zero ideal value with shrink and stretch values.

  This property is important for post-processing: not only do we need a curvilinear stroke but the glyphs surrounding the keshideh in some cases get modified so that they smoothly fit together.

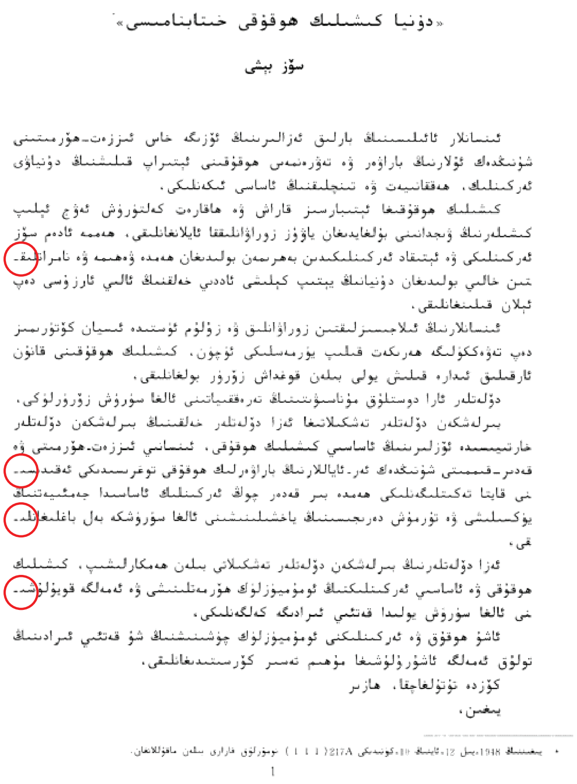
  Last but not least there is another aspect of keshideh: in some cases they may carry short vowels or diacritics. Indeed there is a Unicode character for keshideh: ARABIC TATWEEL. This Unicode character can be used as the base character for short vowels or diacritics (which are all combining characters). In that case we will still use curvilinear keshideh but, again, with a non-zero ideal width so that there is room enough to place the diacritic.

We have presented several cases where injecting extra information into Arabic textemes can prove useful for $\Omega_2$ processing or for other tasks (sorting, indexing, etc.). The other advantage of textemes is that this information will remain in the textual data and will be available to any subsequent operation.

In any case during step 1 of the process we need to calculate Arabic contextual forms. This is described next.

### 3.3 Step 1: Hyphenation

It has been said over and over again that Arabic is not hyphenated. This is true when we refer to Arabic language, but false when we refer to Arabic script. Indeed, there is one language written in Arabic script, namely Uighur, which uses hyphenation just like any European language. Uighur may use the Arabic script but is not a Semitic language and



«دۇنيا كىشىلىك ھوقۇقى خىتابنامىسى»*

سۆز بېشى

ئىنسانلار ئائىلىسىنىڭ بارلىق ئەزالىرىنىڭ ئۆزىگە خاس ئىززەت-ھۆرمىتىنى
شۇنىڭدەك ئۇلارنىڭ باراۋەر ۋە تەۋرەنمەس ھوقۇقىنى ئېتىراپ قىلىشنىڭ دۇنياۋى
ئەركىنلىك، ھەققانىيەت ۋە تىنچلىقنىڭ ئاساسى ئىكەنلىكى،
كىشىلىك ھوقۇقىغا ئېتىبارسىز قاراش ۋە ھاقارەت كەلتۈرۈش ئەۋج ئېلىپ
كىشىلەرنىڭ ۋىجدانىنى بۇلغايدىغان ياۋۇز زوراۋانلىققا ئايلانغانلىقى، ھەممە ئادەم سۆز
ئەركىنلىكى ۋە ئېتىقاد ئەركىنلىكىدىن بەھرىمەن بولىدىغان ھەمدە ۋەھىمە ۋە نامراتلىق-
تىن خالىي بولىدىغان دۇنيانىڭ يېتىپ كېلىشى ئاددىي خەلقنىڭ ئالىي ئارزۇسى دەپ
ئېلان قىلىنغانلىقى،
ئىنسانلارنىڭ ئىلاجسىزلىقتىن زوراۋانلىق ۋە زۇلۇم ئۈستىدە ئىسيان كۆتۈرمىسۇن
دەپ تەۋەككۇلىگە ھەرىكەت قىلىپ يۈرمەسلىكى ئۈچۈن، كىشىلىك ھوقۇقىنى قانۇن
ئارقىلىق ئىدارە قىلىش يولى بىلەن قوغداش زۆرۈر بولغانلىقى،
دۆلەتلەر ئارا دوستلۇق مۇناسىۋىتىنىڭ تەرەققىياتىنى ئالغا سۈرۈش زۆرۈرلۈكى،
بىرلەشكەن دۆلەتلەر تەشكىلاتىغا ئەزا دۆلەتلەر خەلقىنىڭ بىرلەشكەن دۆلەتلەر
خارتىيىسىدە ئۆزلىرىنىڭ ئاساسىي كىشىلىك ھوقۇقى، ئىنسانىي ئىززەت-ھۆرمىتى ۋە
قەدىر-قىممىتى شۇنىڭدەك ئەر-ئاياللارنىڭ باراۋەرلىك ھوقۇقى توغرىسىدىكى ئەقىدىسى-
نى قايتا تەكىتلىگەنلىكى ھەمدە بىر قەدەر چوڭ ئەركىنلىك ئاساسىدا جەمئىيەتنىڭ
يۈكسىلىشى ۋە تۇرمۇش دەرىجىسىنىڭ ياخشىلىنىشىنى ئالغا سۈرۈشكە بەل باغلىغانلى-
قى،
ئەزا دۆلەتلەرنىڭ بىرلەشكەن دۆلەتلەر تەشكىلاتى بىلەن ھەمكارلىشىپ، كىشىلىك
ھوقۇقى ۋە ئاساسىي ئەركىنلىكنىڭ ئومۇميۈزلۈك ھۆرمەتلىنىشى ۋە ئەمەلگە قوبۇلۇشى-
نى ئالغا سۈرۈش يولىدا قەتئىي ئىرادىگە كەلگەنلىكى،
ئاشۇ ھوقۇق ۋە ئەركىنلىكنى ئومۇميۈزلۈك چۈشىنىشنىڭ شۇ قەتئىي ئىرادىنىڭ
تولۇق ئەمەلگە ئاشۇرۇلۇشىغا مۇھىم تەسىر كۆرسىتىدىغانلىقى،
كۆزدە تۇتۇلغاچقا، ھازىر
يىغىن،

* يىغىننىڭ 1948-يىل 12-ئاينىڭ 10-كۈنىدىكى 217A (Ⅲ) نومۇرلۇق قارارى بىلەن ماقۇللانغان.

1

**Figure 2**: A text in Uighur (Universal Declaration of Human Rights).

hence does not use implicit short vowels: all vowels are explicitly written and one can easily identify syllables and hyphenate words between them.[2]

Uighur is indeed hyphenated but if we add soft hyphen characters we risk obstruction of contextual analysis, which is the next step. It is easier to add potential breakpoints as texteme properties, as is done for other languages.

### 3.4 Step 2: Contextual analysis

This is the step where the forms of Arabic letters are calculated based on the context. In the (rather rare) case where the user may want to manually specify a contextual form he/she just needs to set the corresponding texteme feature and to lock it, so that the contextual analysis module will not be able to change it.

Contextual analysis should be done on the module level rather than on the ΩTP level because ΩTPs transform only the contents of a buffer, so that any command inserted into an Arabic word will end the buffer and will obstruct contextual analysis. For example, if we want to colorize a part of an Arabic word we would normally use a `\textcolor` command, but this would enter in conflict with contextual analysis and the colorized letter would be typeset always in isolated form.

Performing contextual analysis on the level of nodes of the horizontal list allows us to obtain a much more reliable result.

### 3.5 Step 2′: Hamza rules

One of the major contributions of ArabTEX (by Klaus Lagally) to Arabic text processing was the fact that it considers Arabic script from a grammatical point of view, while Unicode is bound, by its tenth general principle, to follow legacy encodings like ASMO, which in turn have been based on the character set of Arabic typewriters.

There is one case where the difference between ArabTEX and Unicode is flagrant: the use of *hamza*. This letter represents the glottal stop and can be represented in four possible ways: as an isolated letter (ARABIC LETTER HAMZA), or carried by *alef* (ARABIC LETTER ALEF WITH HAMZA), by *yeh* (ARABIC LETTER YEH WITH HAMZA), or by *waw* (ARABIC LETTER WAW WITH HAMZA). Four Unicode characters for what is in fact a single letter.

The rules of visual representation of *hamza* are quite strict:

1. at word begin: *hamza* is carried by *alef*;
2. inside a word:
   (a) if the *hamza* is preceded or followed by a vowel /i/ (short or long), it is carried by *yeh*,
   (b) otherwise, if it is preceded or followed by a vowel /u/ (short or long), it is carried by *waw*,
   (c) if it is preceded by *yeh* and followed by short vowel /a/, it is carried by *yeh* (with some isolated exceptions where it is carried by *alef*),
   (d) if rules (a), (b), (c) do not apply and it is preceded or followed by a short vowel /a/ it is carried by *alef*,
   (e) if rules (a)–(d) do not apply and it is preceded by a long vowel /a/ it is written without carrier;
3. at word end: if it is preceded by a short vowel /i/, /a/ or /u/ then it is carried by the corresponding long vowel; if it is preceded by a long vowel or a consonant, then it is written without carrier.

[2] It seems that one of the reasons why hyphenation is not desirable for Ivrit (although used by Israeli newspapers) is the fact that the absence of short vowels may lead to bad hyphenations, a problem which could be solved by vowelization.

ArabTEX indeed takes an abstract representation of *hamza* as input and calculates the visual representation according to the rules (and exceptions) above. A "hamza module" for $\Omega_2$ could serve either to facilitate input of Arabic text (but in that case one should develop the corresponding GUI) or as a "spelling checker" for the specific grammatical issue.

### 3.6 Step 3: Bidi algorithm

The bidi algorithm is part of the Unicode specification. It deals with the visual representation of mixed RL (right-to-left) and LR (left-to-right) text. For example, if we consider capitals to be Arabic, when typesetting the sentence "My friend said شكراً!" must the exclamation mark be placed to the right or to the left of شكراً? In other words: is this exclamation mark part of the Arabic sentence "شكراً" or part of the English sentence "My friend said [...]!"? In the first case ("Arabic exclamation mark") the exclamation mark is placed "after" شكراً, and "after" in Arabic means "to the left of it". In the other case it is placed to the right of شكراً.

The problem here is that the exclamation mark has no explicit directionality: it may be equally well considered as being RL or LR. The bidi algorithm gives a canonical default solution to this problem and more generally to the way of rendering a paragraph containing LR, RL as well as neutral (with respect to directionality) characters.

One may ask: "why does Unicode care about rendering issues?" The reason is that one does not always want the canonical solution. To change the order in which blocks of the paragraph are displayed, one can use special Unicode characters RLE, LRE, RLO, LRO, PDF, RLM, LRM. This may sound complicated but in the everyday life of an Arabic language keyboard user, whenever a paragraph does not look like he/she expected it, he/she only needs to insert a character or two among these to obtain the correct rendering. The bidi algorithm is applied on-the-fly by WYSIWYG systems.

What happens when such a text is processed by $\Omega_2$? The latter just needs to perform the same calculations as to obtain the same results as the WYSIWYG system. To do this, one needs to consider the paragraph as a whole. And this is only possible on the level of the horizontal list. This is why a separate bidi module must be applied, and this is step 3 of the process.

### 3.7 Step 4: OpenType and super-OpenType features

In an Arabic rendering process OpenType tables can fulfill five functions:

1. Supply the glyph corresponding to the pair (character, contextual form), the form being provided as an OpenType feature [GSUB table, lookup of type 1 "single substitution"];
2. Supply grammatical (*lam-alif*) and esthetic ligatures [GSUB table, lookup of type 4 "ligature"];
3. Supply alternative forms for glyphs [GSUB table, lookup of type 3 "variant selection"];
4. Kerning between single or ligatured glyphs [GPOS table, lookup of type 2 "positioning of a pair of glyphs"];
5. Place short vowels and other diacritics on isolated glyphs or on ligature components [GPOS table, lookups of types 4 "diacritical marks" and 5 "diacritical marks on ligatures"].

One could imagine some of these features being contextual, with or without backtrack and lookahead. One could also imagine an Arabic OpenType font using the lookup of type 3 "cursive attachment" of the GPOS table.

Clearly these lookups handle most of the complexity of Arabic script. In $\Omega_2$, the GSUB and GPOS tables of the font are read by corresponding modules, which will transform the horizontal list of textemes accordingly.

More precisely, on a first parse of the font we store the glyphs which start a context or match a lookup, as well as the maximum length of context (with backtrack and lookahead) for each glyph. Then, when going through the horizontal list of textemes, for each glyph of a texteme we test whether it is part of a context and then check the following glyphs up to the maximum length for that glyph.

According to OpenType rules we must stop at the first lookup which matches the longest string.

What we call "super-OpenType" is the fact that we take not only the longest string but also all substrings starting by the same glyph, and also that we process all possible lookup matches and not only the first in the list. By doing this we store all possible OpenType transformations so that they produce distinct arcs in the TEX paragraph builder graph. This behaviour is only valid when we are doing dynamic typesetting.

For example, one can imagine four glyphs $g_1g_2g_3g_4$ forming a ligature (like the Arabic word "muhammad"). In a standard OpenType process, the application would find this ligature and stop. In super-OpenType we also store all possible "subligatures" in our texteme: $g_1g_2g_3$, $g_1g_2$, $g_3g_4$, etc. as well as single unligatured glyphs $g_1$, $g_2$, $g_3$, $g_4$. Every choice of ligatures and/or single glyphs results in a different badness calculation for the given line, and hence is a different arc of our (extended) graph.

The reader may wonder how we deal with hyphenation (although there is only one Arabic script language which is hyphenated: Uighur). As hyphenation is performed before OpenType transformations, we already have “alternative horizontal lists” (also called “bifurcations”): with and without line break. We can consider this a split of the horizontal list into two parts: one goes unaltered, and the other contains the hyphen and a special texteme property representing a line break. The OpenType modules go through both parts and apply the necessary transformations, eventually involving the glyph of the hyphen and special end-of-line or begin-of-line features ([11]).

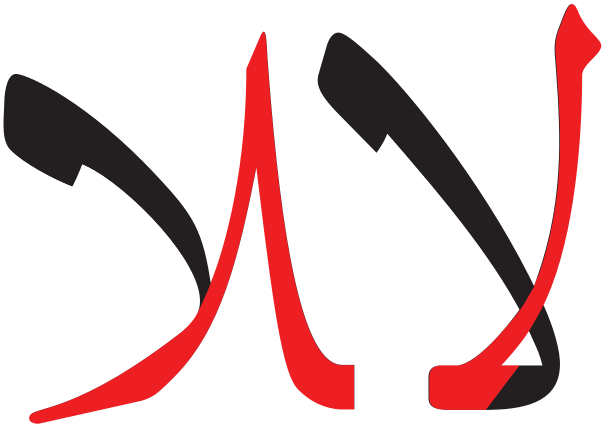


**Figure 3**: The *lam-alif* ligature as two glyphs.

Let us note *en passant* an important problem of Arabic fonts: ligatures *lam-alif* are always drawn as a single glyph. This is justified by the fact that the two letters are always connected, but can be quite problematic when we want to color one of the two letters. We suggest using the following approach: instead of implementing *lam-alif* as a single glyph obtained by a type 4 lookup, divide the ligature glyph in two parts and implement them as variants of the corresponding glyphs, obtained by type 1 lookups (“single substitutions”). In that way they can be colorized separately, but otherwise the visual result is the same.

This problem occurs only for the *lam-alif* ligature since in all other cases we deal with esthetic (and hence non-mandatory) ligatures and one can simply break the ligature into single and separately colorizable glyphs.

### 3.8 Step 5: Fine-tuning

An OpenType font designer is, after all, just a human being, and cannot possibly anticipate *all* possible combinations of glyphs, ligatures, short vowels, diacritics. To face this problem, and knowing that automatic positioning of Arabic short vowels and diacritics has been studied extensively [2], one can adopt at least two approaches:

1. Develop a system which will either refine the font tables whenever a conflict appears in a word (for example two letters touching each other, or a diacritic touching a letter, or a diacritic set in such a way that it is not clear to which letter it belongs) or simply break the corresponding ligature and return to a non-ligatured state (where no conflicts occur);
2. Develop a system checking the default (OpenType) positioning and correcting it accordingly (by slightly moving some of the visual components). This solution is especially interesting when one does not have the rights to modify the font.

This step deals with the latter solution: correcting *a posteriori* the positioning of Arabic letters, short vowels and diacritics, obtained by OpenType transformations. This involves heavy, but well understood, calculations based on glyph outlines: obtain the glyph outlines, place them at given horizontal and vertical offsets, and check whether they touch or even come closer than a given $\varepsilon$ to each other (with a lot of special cases depending on optical effects).

Another solution would be to take pixel images of the various glyphs *emboldened* (so that we also catch glyphs getting close without touching), assemble them using the corresponding horizontal and vertical offsets, and find pixels belonging to more than one glyph.

As always in such cases, a tool as the one described would need serious optimizations to run without slowing down the whole typesetting process. One could imagine a cache mechanism for storing words that present such problems so that the system does not need to do redundant calculations.

There are several ways to proceed for solving this problem. What we wish to point out is the fact that in an infrastructure as the one described here, it is possible to take control of glyphs *after* they have been transformed by OpenType rules.[3]

### 3.9 Step 6: Extended graph and strategies

We arrive now at the paragraph builder. After having gone through the five steps described above,

[3] At this level we recommend operating only on the glyph level. Indeed it would be unwise to change character values of textemes, but, at the very end, this is up to the user to decide: he/she has full control over the textemes before they are transmitted to the paragraph builder.

we have a horizontal list with, occasionally, variant glyphs, some of which are logically connected (for example, when we have a line break). This means that we have several ways of wandering across the horizontal list, according to our choices of variant glyphs and/or line breaks. As in standard TEX we start defining active nodes and calculate the badness of (potential) lines brought to the same width. But in the extended graph we have many graph arcs sharing the same departure and arrival nodes: as many as there are combinations of variant glyphs and activated/de-activated ligatures.

The extended graph is larger than the ordinary TEX graph, but the process of calculating the shortest path is the same. Nevertheless we may have to include logical expressions based on flags corresponding to arcs so that a given path through the graph may accumulate an extra penalty because of interline phenomena (lines touching each other, rivers, etc.).

As already mentioned in the introduction, when a font provides a large number of glyph variants, the badness calculation will suffer from combinatorial explosion. To avoid this we need to implement a *strategy* before entering into the extended graph. The goal of the strategy is to significantly decrease the number of arcs between two given nodes, and nevertheless obtain the smallest possible badness.

This is possible since the badness of a line depends only on widths of glyphs and not on the glyphs themselves. Which means that if we consider classes of glyphs of the same width we obtain the same result and hence need to perform our calculations only on the class level. Once the optimal classes have been found, one can simply choose glyphs randomly in the same class.

But even when using classes of glyphs one can obtain the same badness by ordering them differently. For example, a word can be typeset by using a "wide" class followed by a "narrow" one, or the other way around: the global width will be the same, and so will be the badness. The next step of the strategy would be to choose patterns of classes or at least eliminate arcs based on the same glyph classes but in different orders.

Let us not forget that besides the combinatorics of "rigid" widths provided by variant glyphs, we also have glue, obtained by blank spaces as well as by keshideh. The precision of classes of glyph widths must be set in inverse relation to the amount of glue we can use. The more keshideh and interword glue we will use the less precision we need since differences in width between the glyph class and the glyph actually used will be absorbed by glue.

### 3.10 Step 7: Post-processing

Multiple master tables have been defined for OpenType and then dismissed, as the Multiple Master format is now officially obsolete. AAT variation tables have been used only in a single experimental font (*Skia*). Graphite does provide continuous variation of glyphs. In one word: there is nowadays no glyph variation in font formats.

How do we deal then with keshideh and other glyph variations which have to be continuous?

A possible solution would be to use a postprocessor: a *dvips* module which will convert glue (flagged as "keshideh glue") into beautiful curvilinear strokes and replace the glyphs surrounding the keshideh by appropriate variants.

One could imagine a continuous stroke generator for generating keshideh but also a large set of predesigned keshideh and surrounding glyphs. The latter solution has the advantage of using hints, that can be useful at low resolutions, and should accelerate processing.

### 3.11 Final result

At the end we obtain textemes containing all the information accumulated through these seven steps, as well as the initial information: Unicode characters, contextual forms, semitic roots, full vowelization, etc.

### References

[1] *Mounged de poche.* Dar el-Machreq, 1991.

[2] Gábor Bella. An automatic mark positioning system for Arabic and Hebrew scripts. Master's thesis, ENST Bretagne, October 2003.

[3] Mohamed Jamal Eddine Benatia, Mohamed Elyaakoubi, and Azzeddine Lazrek. Arabic text justification. *TUGboat*, 27(2):137–146, 2006.

[4] Daniel M. Berry. Stretching letter and slanted-baseline formatting for Arabic, Hebrew and Persian with ditroff/ffortid and dynamic PostScript fonts. *Software Practice and Experience*, 29(15):1417–1457, 1999.

[5] Carl Faulmann. *Das Buch der Schrift enthaltend die Schriftzeichen und Alphabete aller Zeiten und aller Völker des Erdkreises.* Druck und Verlag der kaiserlich-königlichen Hof- und Staatsdruckerei, Wien, 1880.

[6] Yannis Haralambous. Arabic, Persian and Ottoman TEX for Mac and PC. *TUGboat*, 11(4):520–524, 1990.

[7] Yannis Haralambous. Towards the revival of traditional Arabic typography. In *Proceedings of the 7th European TEX Conference, Prague*, pages 293–305, 1992.

[8] Yannis Haralambous. Typesetting the Holy Qur'ān with TEX. In *Proceedings of the 3rd International Conference and Exhibition on Multilingual Computing, Durham 1992*, 1992.

[9] Yannis Haralambous. The traditional Arabic typecase extended to the Unicode set of glyphs. *Electronic Publishing—Origination, Dissemination, and Design*, 8(2/3):111–123, 1995.

[10] Yannis Haralambous. Simplification of the Arabic script: Two different approaches and their implementations. In *Electronic Publishing, Artistic Imaging and Digital Typography*, volume 1375 of *Springer Lecture Notes in Computer Science*, pages 138–156. Springer, 1998.

[11] Yannis Haralambous. New hyphenation techniques in $\Omega_2$. *TUGboat*, 27(1):98–103, 2006.

[12] Yannis Haralambous and Gábor Bella. Injecting information into atomic units of text. In *Proceedings of the ACM Symposium on Document Engineering, Bristol*, 2005.

[13] Yannis Haralambous and Gábor Bella. Omega becomes a sign processor. *EuroTEX 2005*, pages 99–110, 2005.

[14] Yannis Haralambous and Gábor Bella. Openbelly surgery in $\Omega_2$. *TUGboat*, 27(1):91–97, 2006.

[15] Yannis Haralambous and John Plaice. First applications of Ω: Adobe Poetica, Arabic, Greek, Khmer. *TUGboat*, 15(3):344–352, 1994.

[16] Yannis Haralambous and John Plaice. Multilingual typesetting with Ω, a case study: Arabic. In *Proceedings of the International Symposium on Multilingual Information Processsing, Tsukuba 1997*, pages 137–154. ETL Japan, 1997.

[17] Ruhi Sarikaya Imed Zitouni, Jerey S. Sorensen. Maximum entropy based restoration of Arabic diacritics. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL, Sydney*, pages 577–584, 2006.

[18] Ahmed Lakhdar-Ghazal. *Pour apprendre et maîtriser la langue arabe*. Éditions La Porte, 1991.

[19] Sakhr. Diacritizer. `http://www.sakhr.com/Sakhr_e/Technology/Diacritization.htm`.

[20] Yannis Haralambous (translated into English by Scott Horne). *Fonts & Encodings*. O'Reilly & Associates, 2007.

# Babel speaks Hindi

Zdeněk Wagner
Vinohradská 114
13000 Prague 3
Czech Republic
zdenek dot wagner (at) gmail dot com
http://icebearsoft.euweb.cz

**Abstract**

Babel provides a unified interface for creation of multilingual documents. Unfortunately no Indic languages are currently supported, so typesetting in Indic languages is based on specialised packages. The most advanced of these is Velthuis Devanāgarī for TeX, because it already provides Hindi values for language-dependent strings as well as a macro for a European-style date. A language definition file for plugging Hindi into Babel has therefore been recently developed.

The second part of the paper explains differences between Unicode and Velthuis transliteration. This is important for understanding the tool that can convert Hindi and Sanskrit documents from MS Word and OpenOffice.org into TeX via an XSLT 2.0 processor and a Perl script, as well as a method of making the PDF files searchable.

Finally the paper discusses some possibilities of further development: the advantages offered by XeTeX and the forthcoming integration of Lua into pdfTeX.

## 1 Introduction

Packages for typesetting in various Indic languages in both plain TeX and LaTeX have been available from CTAN for a long time. The authors of these packages have made substantial efforts to support the Indic scripts, which present difficulties that cannot be solved by TeX itself. For example, although generation of conjuncts and placing dependent vowels (matras) to subscripts and superscripts could be handled as ligatures in TFM files, the form of a conjunct depends also on language. While conjuncts क्त (kta) and न्न (nna) are used in Sanskrit and traditional Hindi, they are nowadays often replaced with the half forms क्‍त and न्‍न, respectively. Such matters can be solved only by using a preprocessor bound to a (La)TeX macro package.

The existing packages are used by indologists from all over the world as well as by people from India. It is therefore unfortunate that the packages support only the script and lack features to support the language. An exception is the Velthuis Devanāgarī for TeX package [1]. Since version 2.13, it contains definitions for language-dependent strings as well as a European-style date, and macros for switching between them and an English version. It therefore provides a kind of a mini-Babel. The further natural development step was to prepare a language definition file which will integrate Hindi into the Babel system.

## 2 Birth of the Language Definition File

The aim of our work was to enable transparent use of Hindi in multilingual documents by means of the standard Babel invocation:

```
\usepackage[hindi]{babel}
```

Preparing a language definition file for Babel is not a difficult task. It involves defining language-dependent macros such as `\chaptername`, and the date macro `\today`. These definitions were already present in the Devanāgarī package and they even properly handled switching between the Bombay, Calcutta and Nepali variants of the Devanāgarī fonts. However, placing them into the language definition file is not enough. Rendering these words in the Devanāgarī font requires a lot of special macros. Moreover, as mentioned previously, the Hindi text cannot be fed directly to TeX — preprocessing is mandatory.

It therefore seemed useless to copy the macros from the Devanāgarī package to the language definition file and follow the same development in two different places. It was therefore decided just to load `devanagari.sty` and redefine its options as language attributes. As a matter of fact, the language definition file itself without the fonts and the preprocessor would not work at all. Thus the requirement to have the Devanāgarī package installed does not pose a real limitation.

The `devanagari.sty` package already contains captions and date macros for English. These definitions must not be removed because they are already documented and their removal might damage legacy documents. On the other hand these definitions collide with the Babel core.

The package was therefore modified so that the macros are declared by means of `\providecommand` and the definition delayed via `\AtBeginDocument`. The macros are thus guaranteed to exist and the Babel definitions have higher preference no matter in which order the files are loaded. The language definition file checks the package version and will complain if an old version of `devanagari.sty` is installed.

As already mentioned, it is not sufficient to activate the Hindi language either as the main language or by any of the Babel language switching environments. The text must be enclosed within a `{\dn ...}` group, or the preprocessor will not find it.

## 3 Unicode vs. Velthuis transliteration

Devanāgarī originates in an ancient Brāhmī script, and is an abugida. Each consonant (vyanjana) also represents an inherent following vowel — *a* in the case of Devanāgarī. Other vowels are added as diacritics in the vicinity of the consonant. Groups of consonants often form conjuncts, with only a minority of them written using a virama sign. Initial form of vowels have a different shape than vowel diacritics (matras).

Unicode is based on characters. The inherent *a* is not encoded. Thus U+0915 represents the क (ka) syllable (akshara). The syllable कि (ki) is represented by the two Unicode characters U+0915 and U+093F. Reversing the order of glyphs in displayed output is left to the rendering engine [2].

Independent vowels (initial forms) have distinct codes, e. g. the code of इ (i) is U+0907. The three characters U+0915 U+094D U+0924 denote a Sanskrit conjunct क्त (kta). However, such a glyph may not be present in modern Hindi fonts. The rendering engine will then display क्‍त. In order to force the rendering engine to display the latter form even though the Sanskrit ligature is present in the font, zero-width-joiner must be inserted. The Unicode encoding will then be U+0915 U+904D U+200D U+0924.

In contrast, the transliteration scheme developed by Frans Velthuis tries to be as close to scholarly practice as possible. Devanāgarī is traditionally transliterated into the Roman alphabet where long vowel, retroflex consonants, etc., are marked with diacritics [10]. The practice varies slightly between different textbooks and dictionaries. The Velthuis transliteration is a 7-bit encoding so the text can be input on a standard US keyboard. Transliteration is based on pronunciation, although in Hindi a short *a* in the middle of words, as well as at the end of words, is very often not pronounced. The inherent *a* in the middle of the word must always be written but the final short *a* can be omitted. Thus करना must be input as *karanaa* while घर can be written just *ghar*.

Important aspects of the differences between these two approaches will be shown in the following subsections.

### 3.1 Conversion into the Velthuis transliteration

Book production is often a collaborative work of author(s), editor(s), and a compositor. The authors rarely supply the texts in TeX; they commonly use other text editors, mostly MS Word. The first task is therefore conversion of the supplied manuscript into TeX. Almost all markup must be removed and replaced in order to achieve a particular graphical design.

We have found it is advantageous to open the manuscript in OpenOffice.org, and then save in its native format, which is XML. Although various tools are freely available, they still retain too much of the author's markup. Use of TeXML will also require much work which can hardly be reused in other books. Since the OpenOffice.org file format is XML, conversion can be performed by XSLT. A simple stylesheet can remove nearly all markup, keeping just boldface, italics and footnotes.

Another difficulty is that some Unicode characters are not directly available in TeX. Fortunately, Saxon 8.x [5], which implements XSLT 2.0, offers character maps, which allow replacement of such characters with TeX control sequences. However, this is not enough for the texts in the Devanāgarī script. If we just convert Unicode characters to corresponding Roman letters, all inherent *a*'s would be lost.

Although the result of the XSLT transformation can be fed into TeX, some postprocessing is recommended. Sometimes bold text is entered such that each character is emboldened separately. Merging these into a single `\textbf` command is more easily done later in a Perl script than directly in the stylesheet. Also, the whole OpenOffice.org document is output on a single line. We need to edit the resulting file in order to wrap it to a reasonable width. Without any programming, this can be done by the Perl `Text::Wrap` module.

In addition to such formatting issues, we also split the main job of conversion between XSLT and Perl. First of all, long vowels can be encoded in the Velthuis transliteration either by doubling or by uppercasing. The stylesheet always uses uppercase for long vowels in order to prevent ambiguities. For instance, कई will be converted into *kaI* because *kaii* will be rendered as कैइ, which is wrong. Empty braces would also solve the problem, but using uppercase is easier. The independent vowels are transformed directly into the corresponding letter(s) in the Velthuis transliteration. The dependent vowels (matras) are preceded with an equal sign. The consonants are followed by equal signs and the virama is transformed into an underscore.

Afterwards the Perl script takes its turn. In the input, each paragraph appears on a single line to be processed. The first task is to form conjuncts. This will work for Sanskrit words as well as modern Hindi, where some ligatures are not used, e. g. in अड्‌डा. The virama will be added by the preprocessor. Conjuncts are created by

```
while (s/(\{\\dn [^}]*)=_/$1/) {}
```

In the next step matras are handled. Double equal signs are simply removed and lone equal signs denote missing inherent *a*'s to be added. This is achieved by the following lines.

```
while
 (s/(\{\\dn [^}]*)==([aAiIuU.eo])/$1$2/) {}
while (s/(\{\\dn [^}]*)=/$1a/) {}
```

Next, we remove the final inherent *a* unless we are converting a Sanskrit document.

```
while (!$opt_sanskrit &&
 s/(\{\\dn [^}]*)a([^.a-zA-Z])/$1$2/) {}
```

Finally the line is wrapped.

```
eval {
  print wrap('', '', $_);  1;
} or do {
  warn "Warning: $@";  print;
};
```

### 3.2 Searchable PDF files

PDF files play an important role as online documents. Although the chapter and section names may be placed into outlines and related parts may be found by hyperlinks, it is also desirable to be able to search for words and sentences. Here, Indic scripts present a big problem. PDF, similar to PostScript, deals with glyphs, but the field in the search dialogue accepts Unicode characters. Files generated by the Devanāgarī package are therefore unsearchable. XƎTEX [13] can use OpenType fonts, but it turns out that a PDF file created by XƎTEX is not searchable either, although the situation is not simple. XƎTEX can use several engines for PDF creation and the results differ. OpenOffice.org is slightly more successful because all simple words which do not contain conjuncts are searchable.

The key problem stems from the distinction between glyphs and characters. Mappings between glyphs and characters can be inserted into so-called 'ToUnicode' maps. This feature is already implemented in the `cmap.sty` package. An experimental ToUnicode map was therefore developed for the Velthuis Devanāgarī package. Since pdfTEX inserts these maps according to the font encoding, each Indic script will require its own encoding name for LATEX. Currently Devanāgarī, Bengali, and Gurmukhi use encoding U, and all of them rely on preprocessors with analogous functionality. In the present experimental project, encoding X0900 was used in order to refer to the corresponding Unicode block.

The `dvng` fonts contain single characters, ligatures and pieces which are glued together by TEX macros. Single characters and ligatures are directly mapped to Unicode characters. Half forms of the consonants are mapped to the corresponding characters followed by a virama. Vattu is added to a full consonant, therefore it is mapped to the र (ra) consonant preceded by virama. This makes words with conjuncts searchable.

Unfortunately it is not possible to solve all problems. Since PDF works with glyphs, i-matras always precede the consonants. When searchng such words, it is necessary to type them into the search field as they appear, i. e. to write the i-matra in front of the consonant. Acrobat Reader is also confused by placing vowel diacritics below or above consonants as well as by vattus. An extra word boundary is created. Thus when searching for कुल्लू, two words कु ल्लू must be typed into the search field. The word ड्राइवर must be entered in a way impossible in the Velthuis transliteration, namely as two words ड्र ाइवर. Unicode allows starting a word with a dependent vowel which is, of course, incorrect. This misfeature enables making such words searchable. Extra word boundaries are not formed if the akshara is drawn as a glyph in the `dvng` font. The words रुकना and मात्रा can be searched for as such. More detailed information is present in the documentation of the experimental ToUnicode map [12].

The ToUnicode map is also used when copying and pasting the text from a PDF file to another application. It is no surprise that we get into the same problems. The pasted text will contain extra spaces after vattu and both short and long u-matras. Words with i-matras will look visually correct but their Unicode representation will be wrong. If a word such as दिल्ली is copied and pasted from PDF and then sent to a sorting algorithm, it will appear in a nonsensical place, because the program will see a word starting with i-matra which is not allowed by Hindi orthography.

## 4 Future development

The greatest disadvantage of contemporary Velthuis Devanāgarī for TeX is the necessity of using the preprocessor. It does not seem so uncomfortable if a single language document is being prepared. However, if a trilingual document in Hindi, Bengali and Panjabi is being typeset, the source file has to be run through three preprocessors in series. The `devnag` preprocessor can handle a few Bengali or Panjabi words within a Hindi paragraph using angle brackets but not all preprocessors are that advanced. The necessity of using the preprocessor brings about difficulties with index creation which have not been solved yet. Replacing the preprocessor with another mechanism is thus an important step forward.

The first idea was to reimplement the preprocessor in encTeX [6]. It hooks into TeX's mouth and converts input characters to arbitrary tokens according to the conversion table. Conversion can be switched on and off by changing the value of `\mubytein`. The conversion table can also modify the file output of `\write` according to value of `\mubyteout`. However, since the conversion acts upon characters in the TeX's mouth, it is not possible to distinguish between characters within words and characters within control sequences. Reimplementation of the preprocessor in encTeX would thus be very difficult and the resulting code inefficient.

More promising is integration of the Lua scripting language [3] into pdfTeX. The preprocessor can be reimplemented in Lua and moreover new features can be added. It will be possible to read texts both in the Velthuis encoding and in Unicode. It will also be possible to process documents already run through the preprocessor so that compatibility will not be lost. When and if Lua is integrated into XƎTeX, it will be possible to choose between `dvng` and OpenType fonts. Having hooks to TeX's mouth as well as output in Lua, it will be easier to implement indexing software in Indic languages.

## 5 Requirements for multilingual environment

Multilingual support is required not only in TeX, but also in the whole operating system. First, it is necessary to display the Unicode characters correctly. Groups of consonants with viramas must be properly combined into conjuncts and matras moved to the correct place. It is achieved in Linux by one of the following libraries: ICU [4] and Pango [8]. However, these libraries are not yet used by all programs. Even Firefox does not use Pango by default, it must first be activated by setting `MOZ_PANGO_ENABLE=1`. MS Internet Explorer displays Hindi texts correctly.

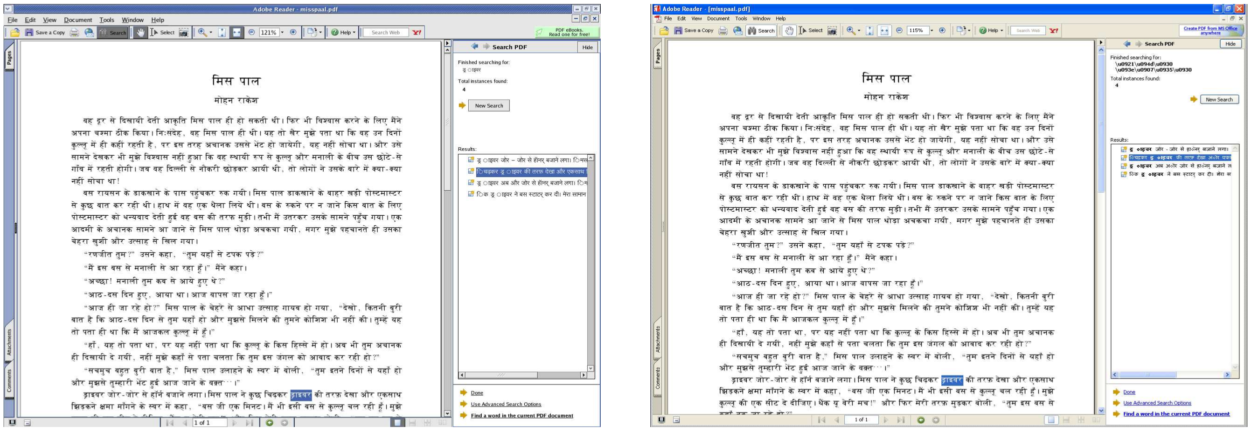
The text must also be prepared using Unicode. There remain problems with editors under Linux. OpenOffice.org, gEdit, and `<oXygen/>` (XML editor) [7] work well; yudit is also said to work, but I have not tried it. Support for Indic scripts in other editors is still missing.

Input in Indic scripts must, of course, be accepted in all applications and displayed correctly. This is a problem with Adobe Acrobat Reader under Windows. I did not manage to enter anything into the search dialogue directly from the Hindi keyboard in a usable way (it displayed something but did not search). It is possible to copy and paste the text from another application but it is displayed in Unicode sequences. A comparison of Linux and Windows versions is shown in Figure 1. The Linux version is more comfortable for users but both search equally well.

The search facility and copying texts from PDF require modification in the CMap encoding. Currently 1:1 and 1:many mappings are available. In order to be able to reverse the order of glyphs and handle two part vowels in Dravidian languages, a many:many mapping is needed. For better understanding, a few Devanāgarī and Malayālam sylables are shown in Table 1.

Table 1: Selected Devanāgarī and Malayālam syllables.

| **Meaning** | **Devanāgarī**<br>देवनागरी | **Malayālam**<br>മലയാളം |
|---|---|---|
| ma | म | മ |
| maa | मा | മാ |
| mi | मि | മി |
| mii | मी | മീ |
| me | मे | മെ |
| mo | मो | മൊ |



Adobe Reader - misspaal.pdf

# मिस पाल

## मोहन राकेश

वह दूर से दिखायी देती आकृति मिस पाल ही हो सकती थी। फिर भी विश्वास करने के लिए मैंने अपना चश्मा ठीक किया। निःसंदेह, वह मिस पाल ही थी। यह तो खैर मुझे पता था कि वह उन दिनों कुल्लू में ही कहीं रहती है, पर इस तरह अचानक उससे भेंट हो जायेगी, यह नहीं सोचा था। और उसे सामने देखकर भी मुझे विश्वास नहीं हुआ कि वह स्थायी रूप से कुल्लू और मनाली के बीच उस छोटे-से गाँव में रहती होगी। जब वह दिल्ली से नौकरी छोड़कर आयी थी, तो लोगों ने उसके बारे में क्या-क्या नहीं सोचा था!

बस रायसन के डाकखाने के पास पहुंचकर रुक गयी। मिस पाल डाकखाने के बाहर खड़ी पोस्टमास्टर से कुछ बात कर रही थी। हाथ में वह एक थैला लिये थी। बस के रुकने पर न जाने किस बात के लिए पोस्टमास्टर को धन्यवाद देती हुई वह बस की तरफ मुड़ी। तभी मैं उतरकर उसके सामने पहुँच गया। एक आदमी के अचानक सामने आ जाने से मिस पाल थोड़ा अचकचा गयी, मगर मुझे पहचानते ही उसका चेहरा खुशी और उत्साह से खिल गया।

“रणजीत तुम?” उसने कहा, “तुम यहाँ से टपक पड़े?”

“मैं इस बस से मनाली से आ रहा हूँ।” मैंने कहा।

“अच्छा! मनाली तुम कब से आये हुए थे?”

“आठ-दस दिन हुए, आया था। आज वापस जा रहा हूँ।”

“आज ही जा रहे हो?” मिस पाल के चेहरे से आधा उत्साह गायब हो गया, “देखो, कितनी बुरी बात है कि आठ-दस दिन से तुम यहाँ हो और मुझसे मिलने की तुमने कोशिश भी नहीं की। तुम्हें यह तो पता ही था कि मैं आजकल कुल्लू में हूँ।”

“हाँ, यह तो पता था, पर यह नहीं पता था कि कुल्लू के किस हिस्से में हो। अब भी तुम अचानक ही दिखायी दे गयीं, नहीं मुझे कहाँ से पता चलता कि तुम इस जंगल को आबाद कर रही हो?”

“सचमुच बहुत बुरी बात है,” मिस पाल उलाहने के स्वर में बोली, “तुम इतने दिनों से यहाँ हो और मुझसे तुम्हारी भेंट हुई आज जाने के वक़्त···।”

ड्राइवर जोर-जोर से हॉर्न बजाने लगा। मिस पाल ने कुछ चिढ़कर ड्राइवर की तरफ़ देखा और एकसाथ झिड़कने क्षमा माँगने के स्वर में कहा, “बस जी एक मिनट। मैं भी इसी बस से कुल्लू चल रही हूँ। मुझे कुल्लू की एक सीट दे दीजिए। थैंक यू वेरी मच!” और फिर मेरी तरफ़ मुड़कर बोली, “तुम इस बस से

Search PDF

Finished searching for:
ड्र ○ाइवर

Total instances found:
4

Results:
ड्र ○ाइवर जोर – जोर से हॉनर् बजाने लगा। ि○मस
ि○चढ़कर ड्र ○ाइवर की तरफ़ देखा और एकसाथ
ड्र ○ाइवर अब और जोर से हॉनर् बजाने लगा। ि○म
ि○क ड्र ○ाइवर ने बस स्टाटर् कर दी। मेरा सामान

Search PDF

Finished searching for:
\u0921\u094d\u0930
\u093e\u0907\u0935\u0930

Total instances found:
4

Results:
ड्र ○ाइवर जोर - जोर से हा○नर् बजाने लगा।
ि○चढ़कर ड्र ○ाइवर की तरफ़ देखा अ○ोर एक
ड्र ○ाइवर अब अ○ोर जोर से हा○नर् बजाने ल
ि○क ड्र ○ाइवर ने बस स्टाटर् कर दी। मेरा स

File
Edit
View
Document
Tools
Window
Help
Save a Copy
Search
Select
121%
115%
Help
Search Web
PDF eBooks. Read one for free!
Create PDF from MS Office anywhere
Adobe Reader - [misspaal.pdf]
Hide
New Search
Pages
Attachments
Comments
1 of 1
Done
Use Advanced Search Options
Find a word in the current PDF document

**Figure 1**: Comparison of search for ड्राइवर (input as ड्र ाइवर) in Adobe Acrobat Reader 7 in Linux (left) and Windows XP (right).

## 6 Conclusion

The paper describes how the Babel module for Hindi was made. It further presents thoughts concerning the use of Indic texts as online documents. Tools for conversion from MS Word and OpenOffice.org into TEX and for making PDF files with Devanāgarī texts searchable are presented. These tools are available from the author's web page [11].

## 7 Acknowledgement

The author would like to acknowledge the work of other developers of the Velthuis Devanāgarī for TEX, as the package is the base for this project, as well as Karel Píška for converting the Indic fonts to the Type 1 format [9]. Special thanks belong to Anshuman Pandey for translating the language-dependent strings into Hindi, and John Smith and Arnošt Štědrý for providing test files created by XƎTEX. The author would also like to acknowledge Alexandr Babič for running the test under Ubuntu and Petr Tomášek for explanation of topics related to font rendering in X. Finally, the author would like to acknowledge financial support of his attendance of the TUG 2006 conference by CSTUG and TEX Users Group.

## References

[1] Devanāgarī for TEX.
http://devnag.sarovar.org/.

[2] Joan Aliprand et al. *The Unicode Standard*, chapter South Asian Scripts. The Unicode Consortium, 2003. http://www.unicode.org/faq/indic.html#5.

[3] Hans Hagen. LuaTEX: Howling to the moon. *TUGboat*, 26(2):152–157, 2005. http://www.tug.org/TUGboat/Contents/contents26-2.html.

[4] International components for Unicode.
http://icu.sourceforge.net/.

[5] Michael Kay. Saxon, the XSLT and XQuery processor. http://saxon.sourceforge.net/.

[6] Petr Olšák. encTEX.
http://www.olsak.net/enctex.html.

[7] <oXygen/> XML editor.
http://www.oxygenxml.com/.

[8] Pango. http://www.pango.org/.

[9] Karel Píška. Indic Type 1 fonts for TEX.
CTAN:fonts/ps-type1/indic.

[10] Transliteration pages. http://homepage.ntlworld.com/stone-catend/translit.htm.

[11] Zdeněk Wagner. My free software.
http://icebearsoft.euweb.cz/sw.php.

[12] Zdeněk Wagner. Searchable PDF with Devanāgarī texts. http://icebearsoft.euweb.cz/dvngpdf/.

[13] The XƎTEX typesetting system.
http://scripts.sil.org/xetex.

# LaTeX as a tool for the typographic reproduction of ancient texts

Apostolos Syropoulos
Greek TeX Friends Group
366, 28th October Str.
GR-671 00 Xanthi, Greece

## Abstract

Modern digital typography makes it possible to reliably reproduce ancient texts. In the TeX world, one usually employs a macro package and one or more PostScript fonts in order to accomplish his/her task. Here we briefly describe some tools that have not been described in the literature and then we give our own response to the important question regarding the suitability of the LaTeX tools that are available for the reproduction of ancient texts. More generally, we are also concerned about the suitability of our typesetting enginees to handle demanding typographic problems. Naturally, an answer cannot be definitive, nevertheless, we conclude that the tools and typesetting engines have provisions to handle all possible forms of ancient documents.

## 1 Introduction

The reproduction of ancient texts is an important issue, mainly because by digitally reproducing ancient documents we can preserve our cultural heritage. Indeed, digital typography is about culture and art as much it is about science and technology. And it is not surprising that many TeX experts have devoted much time and energy to develop a number of tools for the typesetting of ancient scripts. For instance, Peter Wilson and the present author have created a number of such tools, which are partially described in [1, 2, 3]. Obviously, these tools cannot cover all possible cases, but are good enough for most tasks. However, the really important question is whether tools such as these can be used to digitally reproduce any ancient document. Naturally, an answer to this question cannot be definitive, nevertheless, as it will be demonstrated later on, it seems that these tools can be used to reproduce *any* ancient document.

The purpose of this article is not only to show that modern typesetting tools can be used to reproduce many ancient documents, but also to describe a number of tools that have not been described in the literature so far. For this reason, we will describe the tools developed to typeset the symbols of the disk of Phaistos, the symbols of the Linear A script, the symbols of the Epi-Olmec script, and the staves used in medieval Iceland. In addition, we will briefly discuss our project to provide support for the Cretan hieroglyphics.

In the next section we will review the typesetting tools for various scripts that have not been described in the literature. Then we will discuss whether these tools solve the problem they have been designed to tackle. We conclude with a short discussion regarding our future projects.

## 2 From ancient Greece through ancient Meso-America to medieval Iceland

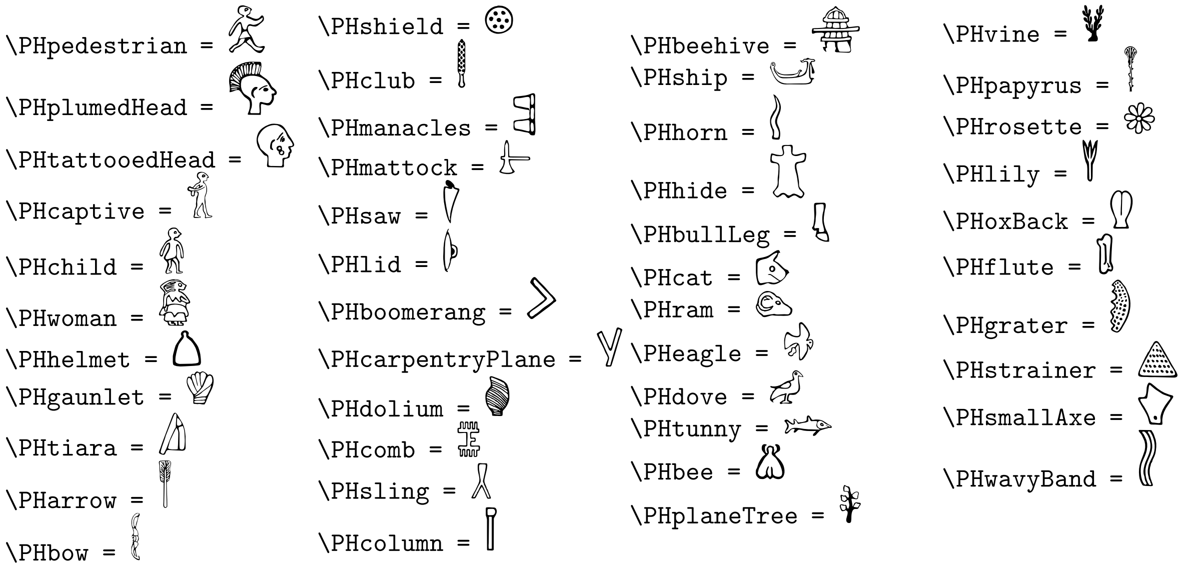
The great island of Crete was the site of the Minoan civilisation. On this island Cretans developed their various writing systems, including Linear A and Linear B. Before these scripts gained widespread use and acceptance, a hieroglyphic script was in use. To the best of our knowledge, this script has not been deciphered yet. Also, on the famous Disk of Phaistos is inscribed a text in an undeciphered script. Wilson has created a font and a LaTeX package to typeset Linear B documents, whilst the present author developed a font and a LaTeX package that can be used to typeset Linear A tablets.

### 2.1 The Disk of Phaistos

The present author and Stratos Doumanis have created a font and an accompanying LaTeX package, named phaistos, that can be used to digitally reproduce both sides of the Disk of Phaistos. The package provides access to the glyphs of the Phaistos font, via commands of the form `\PHxxxx`. The `xxxx` part of the name follows the "*Phaistos Con-Script Unicode Standard*" (`http://www.evertype.com/standards/csur/phaistos.html`. The list of glyph access commands is shown in Table 1.

### 2.2 Linear A

The package LinearA provides access to the fonts LinearA and LinearACmplxSigns, which contain both the simple and compound symbols of the Linear A



\PHpedestrian =
\PHplumedHead =
\PHtattooedHead =
\PHcaptive =
\PHchild =
\PHwoman =
\PHhelmet =
\PHgaunlet =
\PHtiara =
\PHarrow =
\PHbow =

\PHshield =
\PHclub =
\PHmanacles =
\PHmattock =
\PHsaw =
\PHlid =
\PHboomerang =
\PHcarpentryPlane =
\PHdolium =
\PHcomb =
\PHsling =
\PHcolumn =

\PHbeehive =
\PHship =
\PHhorn =
\PHhide =
\PHbullLeg =
\PHcat =
\PHram =
\PHeagle =
\PHdove =
\PHtunny =
\PHbee =
\PHplaneTree =

\PHvine =
\PHpapyrus =
\PHrosette =
\PHlily =
\PHoxBack =
\PHflute =
\PHgrater =
\PHstrainer =
\PHsmallAxe =
\PHwavyBand =

Table 1: Glyph access commands provided by the phaistos package.

script. The term compound symbol denotes a symbol that consists of two or more simple symbols. For example, the compound symbol is composed of the symbols and . The two fonts contain 389 glyphs and since there are no "official" names for the various glyphs, the glyph access commands have the following general form: `\LinearALLLL`, where `LLLL` is a roman numeral. For example, this Linear A text:

was typeset with the following commands:

```
\LinearAXXIX{\LinearAXLIII}{\LinearAXXXIV}%
\LinearALXVIII{\LinearALX}{\LinearAIV}%
{\LinearAXXII}%
```

Each command is defined using the `\xspace` command, and thus we need to surround the various commands with curly braces to avoid the production of extra white space. If a user does not want to refer to individual glyphs and worry about spacing, he/she can employ the following redefinition after the command that includes the LinearA package:

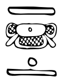
```
\let\xspace\relax
```

In Table 2 the reader can see the glyph access commands provided for the simple symbols of the Linear A script. Of course it would be nice to have this as a package option, and it is planned for inclusion in a future release.

### 2.3 Epi-Olmec

The epiolmec package provides the necessary commands to typeset Epi-Olmec texts and numerals. The various glyph access commands are shown in Table 3. The Epi-Olmec people of Meso-America used the vigesimal numbering system (i.e., a base 20 numbering system). The package provides two commands to generate Epi-Olmec numerals, which are identical to the numerals employed by the Mayan people. Specifically, the command `\vigesimal` typesets a number on a horizontal line, while the command `\StackedVigesimal` typesets the same numeral on a vertical line, which is the historical way that the Epi-Olmec people wrote numbers.

For example, the command `\vigesimal{2006}` produces the symbols , while the command `\StackedVigesimal{2006}` produces:

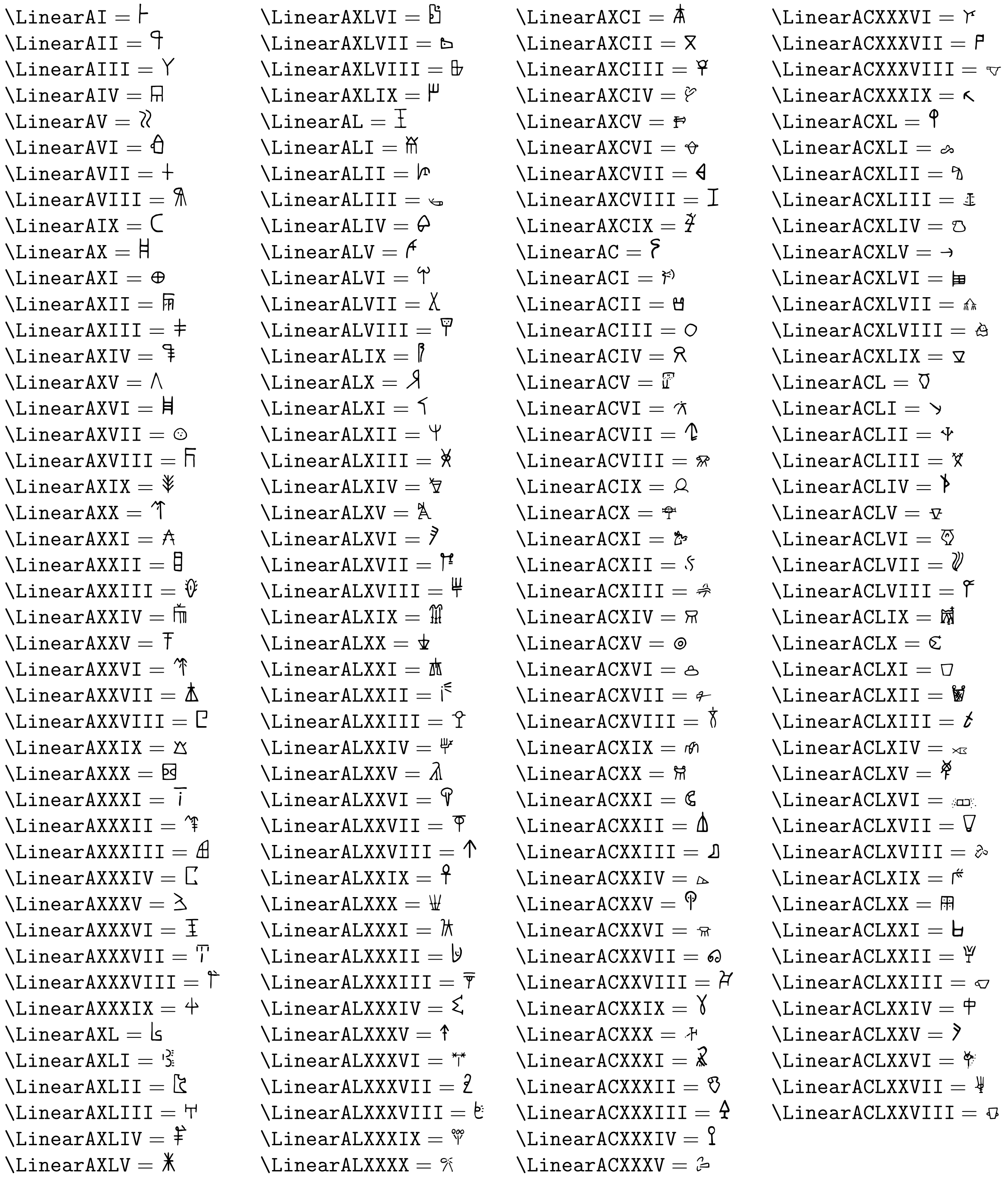


### 2.4 Icelandic

The package icelandic provides access to the font icelandic. This font contains the Icelandic runes as well as most of the Icelandic sorcery and witchcraft staves. The letters can be accessed by simply typing the lowercase Latin equivalent (see Table 4).

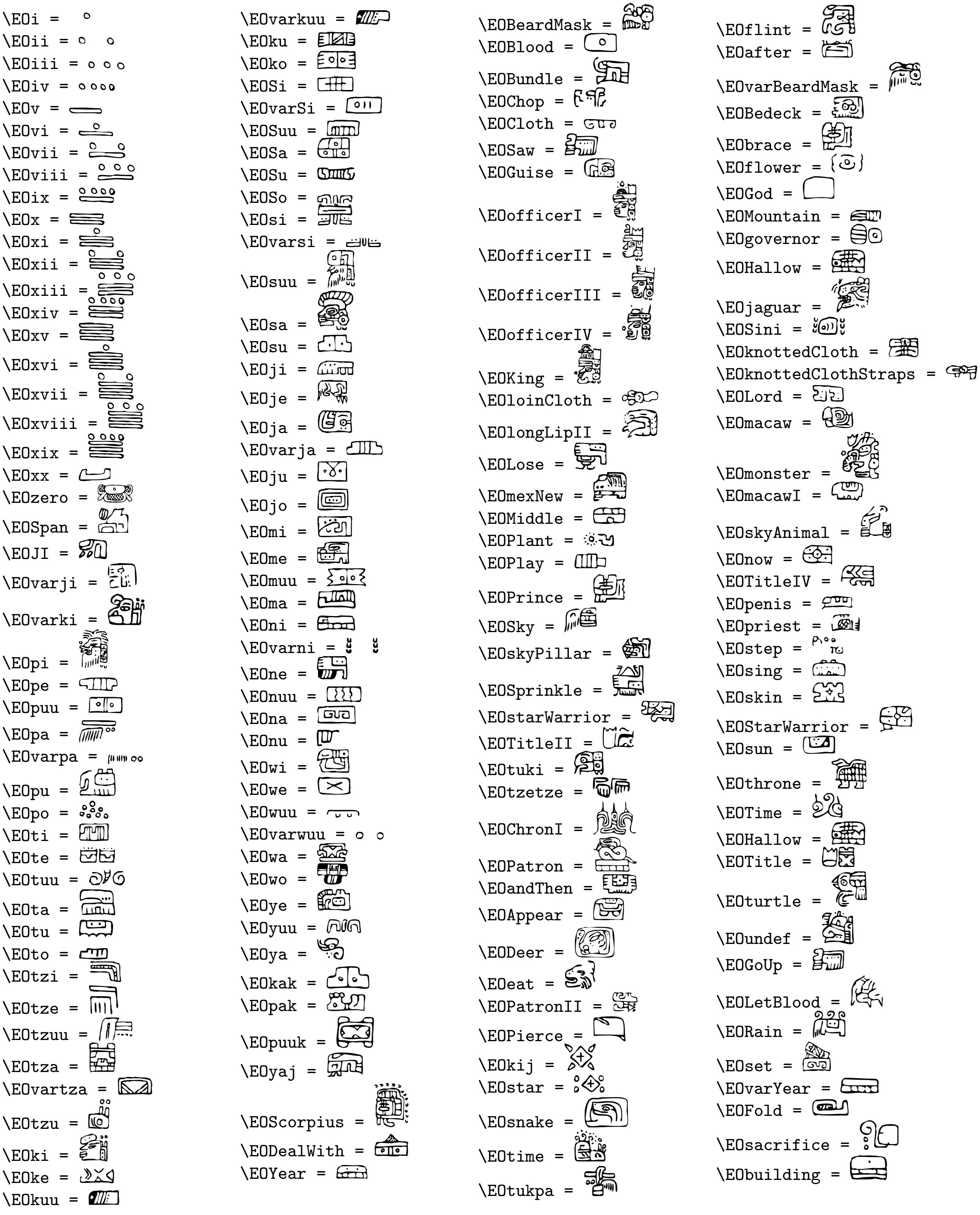
For example, the following text: was typeset with the input '`island`':

To typeset the staves, one uses commands similar to those provided by the LinearA package (see Table 5).



```
\LinearAI =
\LinearAII =
\LinearAIII =
\LinearAIV =
\LinearAV =
\LinearAVI =
\LinearAVII =
\LinearAVIII =
\LinearAIX =
\LinearAX =
\LinearAXI =
\LinearAXII =
\LinearAXIII =
\LinearAXIV =
\LinearAXV =
\LinearAXVI =
\LinearAXVII =
\LinearAXVIII =
\LinearAXIX =
\LinearAXX =
\LinearAXXI =
\LinearAXXII =
\LinearAXXIII =
\LinearAXXIV =
\LinearAXXV =
\LinearAXXVI =
\LinearAXXVII =
\LinearAXXVIII =
\LinearAXXIX =
\LinearAXXX =
\LinearAXXXI =
\LinearAXXXII =
\LinearAXXXIII =
\LinearAXXXIV =
\LinearAXXXV =
\LinearAXXXVI =
\LinearAXXXVII =
\LinearAXXXVIII =
\LinearAXXXIX =
\LinearAXL =
\LinearAXLI =
\LinearAXLII =
\LinearAXLIII =
\LinearAXLIV =
\LinearAXLV =
\LinearAXLVI =
\LinearAXLVII =
\LinearAXLVIII =
\LinearAXLIX =
\LinearAL =
\LinearALI =
\LinearALII =
\LinearALIII =
\LinearALIV =
\LinearALV =
\LinearALVI =
\LinearALVII =
\LinearALVIII =
\LinearALIX =
\LinearALX =
\LinearALXI =
\LinearALXII =
\LinearALXIII =
\LinearALXIV =
\LinearALXV =
\LinearALXVI =
\LinearALXVII =
\LinearALXVIII =
\LinearALXIX =
\LinearALXX =
\LinearALXXI =
\LinearALXXII =
\LinearALXXIII =
\LinearALXXIV =
\LinearALXXV =
\LinearALXXVI =
\LinearALXXVII =
\LinearALXXVIII =
\LinearALXXIX =
\LinearALXXX =
\LinearALXXXI =
\LinearALXXXII =
\LinearALXXXIII =
\LinearALXXXIV =
\LinearALXXXV =
\LinearALXXXVI =
\LinearALXXXVII =
\LinearALXXXVIII =
\LinearALXXXIX =
\LinearALXXXX =
\LinearAXCI =
\LinearAXCII =
\LinearAXCIII =
\LinearAXCIV =
\LinearAXCV =
\LinearAXCVI =
\LinearAXCVII =
\LinearAXCVIII =
\LinearAXCIX =
\LinearAC =
\LinearACI =
\LinearACII =
\LinearACIII =
\LinearACIV =
\LinearACV =
\LinearACVI =
\LinearACVII =
\LinearACVIII =
\LinearACIX =
\LinearACX =
\LinearACXI =
\LinearACXII =
\LinearACXIII =
\LinearACXIV =
\LinearACXV =
\LinearACXVI =
\LinearACXVII =
\LinearACXVIII =
\LinearACXIX =
\LinearACXX =
\LinearACXXI =
\LinearACXXII =
\LinearACXXIII =
\LinearACXXIV =
\LinearACXXV =
\LinearACXXVI =
\LinearACXXVII =
\LinearACXXVIII =
\LinearACXXIX =
\LinearACXXX =
\LinearACXXXI =
\LinearACXXXII =
\LinearACXXXIII =
\LinearACXXXIV =
\LinearACXXXV =
\LinearACXXXVI =
\LinearACXXXVII =
\LinearACXXXVIII =
\LinearACXXXIX =
\LinearACXL =
\LinearACXLI =
\LinearACXLII =
\LinearACXLIII =
\LinearACXLIV =
\LinearACXLV =
\LinearACXLVI =
\LinearACXLVII =
\LinearACXLVIII =
\LinearACXLIX =
\LinearACL =
\LinearACLI =
\LinearACLII =
\LinearACLIII =
\LinearACLIV =
\LinearACLV =
\LinearACLVI =
\LinearACLVII =
\LinearACLVIII =
\LinearACLIX =
\LinearACLX =
\LinearACLXI =
\LinearACLXII =
\LinearACLXIII =
\LinearACLXIV =
\LinearACLXV =
\LinearACLXVI =
\LinearACLXVII =
\LinearACLXVIII =
\LinearACLXIX =
\LinearACLXX =
\LinearACLXXI =
\LinearACLXXII =
\LinearACLXXIII =
\LinearACLXXIV =
\LinearACLXXV =
\LinearACLXXVI =
\LinearACLXXVII =
\LinearACLXXVIII =
```

Table 2: Glyph access commands for the simple symbols provided by the linearA package.



```
\EOi =
\EOii =
\EOiii =
\EOiv =
\EOv =
\EOvi =
\EOvii =
\EOviii =
\EOix =
\EOx =
\EOxi =
\EOxii =
\EOxiii =
\EOxiv =
\EOxv =
\EOxvi =
\EOxvii =
\EOxviii =
\EOxix =
\EOxx =
\EOzero =
\EOSpan =
\EOJI =
\EOvarji =
\EOvarki =
\EOpi =
\EOpe =
\EOpuu =
\EOpa =
\EOvarpa =
\EOpu =
\EOpo =
\EOti =
\EOte =
\EOtuu =
\EOta =
\EOtu =
\EOto =
\EOtzi =
\EOtze =
\EOtzuu =
\EOtza =
\EOvartza =
\EOtzu =
\EOki =
\EOke =
\EOkuu =
```

```
\EOvarkuu =
\EOku =
\EOko =
\EOSi =
\EOvarSi =
\EOSuu =
\EOSa =
\EOSu =
\EOSo =
\EOsi =
\EOvarsi =
\EOsuu =
\EOsa =
\EOsu =
\EOji =
\EOje =
\EOja =
\EOvarja =
\EOju =
\EOjo =
\EOmi =
\EOme =
\EOmuu =
\EOma =
\EOni =
\EOvarni =
\EOne =
\EOnuu =
\EOna =
\EOnu =
\EOwi =
\EOwe =
\EOwuu =
\EOvarwuu =
\EOwa =
\EOwo =
\EOye =
\EOyuu =
\EOya =
\EOkak =
\EOpak =
\EOpuuk =
\EOyaj =
\EOScorpius =
\EODealWith =
\EOYear =
```

```
\EOBeardMask =
\EOBlood =
\EOBundle =
\EOChop =
\EOCloth =
\EOSaw =
\EOGuise =
\EOofficerI =
\EOofficerII =
\EOofficerIII =
\EOofficerIV =
\EOKing =
\EOloinCloth =
\EOlongLipII =
\EOLose =
\EOmexNew =
\EOMiddle =
\EOPlant =
\EOPlay =
\EOPrince =
\EOSky =
\EOskyPillar =
\EOSprinkle =
\EOstarWarrior =
\EOTitleII =
\EOtuki =
\EOtzetze =
\EOChronI =
\EOPatron =
\EOandThen =
\EOAppear =
\EODeer =
\EOeat =
\EOPatronII =
\EOPierce =
\EOkij =
\EOstar =
\EOsnake =
\EOtime =
\EOtukpa =
```

```
\EOflint =
\EOafter =
\EOvarBeardMask =
\EOBedeck =
\EObrace =
\EOflower =
\EOGod =
\EOMountain =
\EOgovernor =
\EOHallow =
\EOjaguar =
\EOSini =
\EOknottedCloth =
\EOknottedClothStraps =
\EOLord =
\EOmacaw =
\EOmonster =
\EOmacawI =
\EOskyAnimal =
\EOnow =
\EOTitleIV =
\EOpenis =
\EOpriest =
\EOstep =
\EOsing =
\EOskin =
\EOStarWarrior =
\EOsun =
\EOthrone =
\EOTime =
\EOHallow =
\EOTitle =
\EOturtle =
\EOundef =
\EOGoUp =
\EOLetBlood =
\EORain =
\EOset =
\EOvarYear =
\EOFold =
\EOsacrifice =
\EObuilding =
```

Table 3: Glyph access commands provided by the epiolmec package.

| Unicode name | Access character |
|---|---|
| RUNIC LETTER ANSUZ A | a |
| RUNIC LETTER BERKANAN BEORC RJARKAN B | b |
| RUNIC LETTER IWAZ EOH | c |
| RUNIC LETTER D | d |
| RUNIC LETTER E | e |
| RUNIC LETTER FEHU FEOH FE F | f |
| RUNIC LETTER GEBO GYFU G | g |
| RUNIC LETTER HAGLAZ H | h |
| RUNIC LETTER ISAZ IS ISS I | i |
| RUNIC LETTER THURISAZ THURS THORN | j |
| RUNIC LETTER KAUNA | k |
| RUNIC LETTER LAUKAZ LAGU LOGR L | l |
| RUNIC LETTER MANNAZ MAN M | m |
| RUNIC LETTER NAUDIZ NYD NAUD N | n |
| RUNIC LETTER OTHALAN ETHEL O | o |
| RUNIC LETTER PERTHO PEORTH P | p |
| RUNIC LETTER INGWAZ | q |
| RUNIC LETTER RAIDO RAD REID R | r |
| RUNIC LETTER SIGEL LONG-BRANCH-SOL S | s |
| RUNIC LETTER TIWAZ TIR TYR T | t |
| RUNIC LETTER URUZ UR U | u |

Table 4: Runic letters supported by the staves package.

## 3 Are our tools good enough?

The suitability of any tool, in general, is a tantalizing question that every designer should consider when creating his/her tools.

In our case, we need to know whether our tools can be used to reproduce any ancient document they were intended to be able to reproduce. The various scripts presented so far are not especially complicated, and one can safely say that the tools can reliably reproduce documents writen in these scripts.

However, there are much more esoteric writing systems, such as the Mayan writing system, that seem quite challenging. The Mayan writing system is a two-dimensional system, in the sense that there is a main sign surrounded by other signs. The main sign is larger than the other signs, which are affixes. There are four kinds of affixes: prefixes, which are placed to the left of the main sign, superfixes, which are placed above the main sign, subfixes, which are placed below the main sign, and postfixes, which are placed at the right of the main sign. Affixes can also be fused within the main glyph and are called infixes. So this is a vastly more complicated writing system, and it would be quite challenging to develop a tool for mechanically typesetting Mayan documents from sources in some Latinized transliterated form. It would seem this is a typographic job particularly well-suited for advanced typesetting engines such as Ω, ℵ, and/or XƎTEX.[1]

On the other hand, to merely literally reproduce a Mayan script, one could use the picture enviroment or define some other environment/command to build these complicated compounds. Of course, this is an unintelligent solution. An appealing solution would be quite difficult to implement.

So where are we? The answer is that we need advanced typesetting engines to be able to typeset complicated writing systems easily. So far, it seems that most cases can be handled by existing tools, but it is necessary to make sure that these tools are reliable and give always the expected results. This is quite feasible, but demands a clear design and one that will not be based on previous programming approaches and design principles..

## 4 Epilog

We have presented work that we have done over the last few years in the field of digital typography. Specifically, we presented the tools we developed to typeset ancient documents. This work and other similar projects prompted us to ponder about the suitability of our typesetting engines as typographic tools capable to handle any ancient script. We concluded that an unintelligent design is possible for any imaginable writing system, but the recreation of an ancient document from "raw" data is something that demands very sophisticated typesetting engines. Thus, we need to examine all writing systems and to adapt our tools so as to be able to handle all possible cases. This will prompt researchers and developers to work toward the creation of new tools and/or the improvement of existing tools.

### Acknowledgements

I would like to thank the *TUGboat* reviewers for their comments and suggestions.

### References

[1] Apostolos Syropoulos. Replicating Archaic Documents: A Typographic Challenge. *TUGboat*, 24(3):319–322, 2003.

[2] Apostolos Syropoulos, Antonis Tsolomitis, and Nick Sofroniou. *Digital Typography Using LaTeX*. Springer Professional Computing. Springer-Verlag, New York, 2003.

[3] Peter Wilson. The alphabet tree. *TUGboat*, 26(3):199–214, 2005.

[1] Ω is the last letter of the Greek alphabet, while ℵ is the first letter of the Hebrew alphabet, which makes me think that Ω was supposed to be the ultimate typesetting engine, but ℵ showed that probably there is still a long way to the end... A belief that was verified with the emergence of XƎTEX!

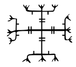
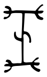
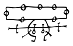
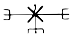
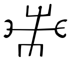
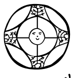
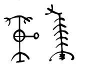
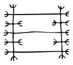
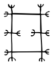
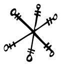
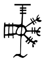
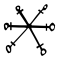
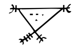
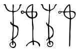
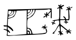
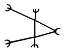
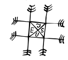
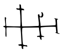
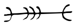
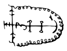
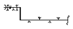
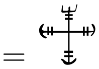
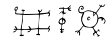
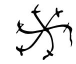
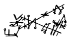
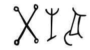
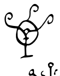
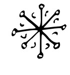
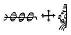
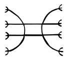
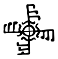
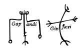
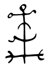
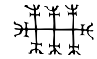
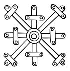
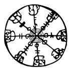
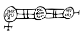
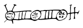
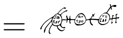
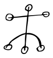
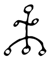
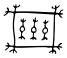
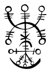
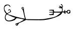
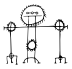
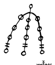
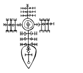
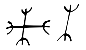
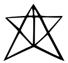
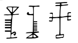
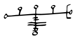
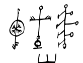
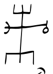
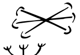
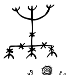
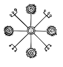
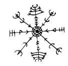
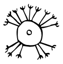
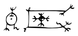
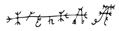
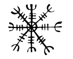
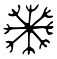
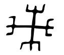
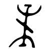
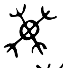
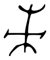
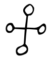
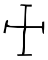
| | |
|---|---|
| \staveI = |  |
| \staveII = |  |
| \staveIII = |  |
| \staveIV = |  |
| \staveV = |  |
| \staveVI = |  |
| \staveVII = |  |
| \staveVIII = |  |
| \staveIX = |  |
| \staveX = |  |
| \staveXI = |  |
| \staveXII = |  |
| \staveXIII = |  |
| \staveXIV = |  |
| \staveXV = |  |
| \staveXVI = |  |
| \staveXVII = |  |
| \staveXVIII = |  |
| \staveXIX = |  |
| \staveXX = |  |
| \staveXXI = |  |
| \staveXXII = |  |
| \staveXXIII = |  |
| \staveXXIV = |  |
| \staveXXV = |  |
| \staveXXVI = |  |
| \staveXXVII = |  |
| \staveXXVIII = |  |
| \staveXXIX = |  |
| \staveXXX = |  |
| \staveXXXI = |  |
| \staveXXXII = |  |
| \staveXXXIII = |  |
| \staveXXXIV = |  |
| \staveXXXV = |  |
| \staveXXXVI = |  |
| \staveXXXVII = |  |
| \staveXXXVIII = |  |
| \staveXXXIX = |  |
| \staveXL = |  |
| \staveXLI = |  |
| \staveXLII = |  |
| \staveXLIII = |  |
| \staveXLIV = |  |
| \staveXLV = |  |
| \staveXLVI = |  |
| \staveXLVII = |  |
| \staveXLVIII = |  |
| \staveXLIX = |  |
| \staveL = |  |
| \staveLI = |  |
| \staveLII = |  |
| \staveLIII = |  |
| \staveLIV = |  |
| \staveLV = |  |
| \staveLVI = |  |
| \staveLVII = |  |
| \staveLVIII = |  |
| \staveLIX = |  |
| \staveLX = |  |
| \staveLXI = |  |
| \staveLXII = |  |
| \staveLXIII = |  |
| \staveLXIV = |  |
| \staveLXV = |  |
| \staveLXVI = |  |
| \staveLXVII = |  |
| \staveLXVIII = |  |
| a = | ᚨ |
| b = | ᛒ |
| c = | ᛣ |
| d = | ᛞ |
| e = | ᛖ |
| f = | ᚠ |
| g = | ᚷ |
| h = | ᚺ |
| i = | ᛁ |
| j = | ᚦ |
| k = | ᚲ |
| l = | ᛚ |
| m = | ᛗ |
| n = | ᚾ |
| o = | ᛟ |
| p = | ᛕ |
| q = | ◇ |
| r = | ᚱ |
| s = | ᛊ |
| t = | ᛏ |
| u = | ᚢ |

Table 5: Glyph access commands provided by the staves package.

# Generating TEX from mathematical content with respect to notational settings

Elena Smirnova
Ontario Research Centre for Computer Algebra
The University of Western Ontario
London, ON, N6A 5B7, Canada
elena (at) orcca dot on dot ca
http://www.orcca.on.ca/MathML/elena.html

Stephen M. Watt
Department of Computer Science
The University of Western Ontario
London, ON, N6A 5B7, Canada
watt (at) csd dot uwo dot ca
http://www.csd.uwo.ca/~watt.html

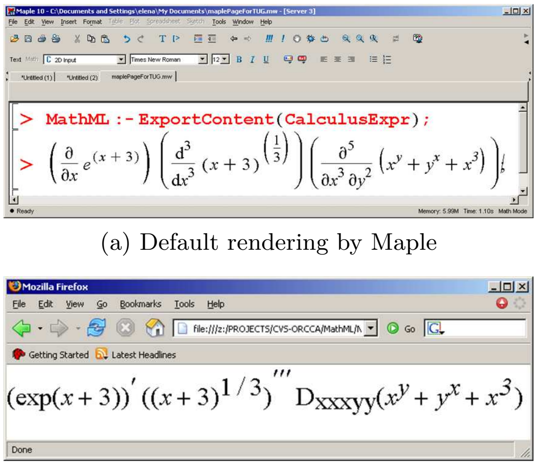
### Abstract

We describe how to obtain client-preferred notations in TEX generated from the output of mathematical software environments. Our approach is based on the fact that most packages can produce MathML or other XML-based formats for mathematical content. Generating TEX from these allows notational choices to be applied during the translation process. The particular choices of notation can be made either at the time TEX is generated or later, by the use of TEX macros. We show how this approach may be applied to the generation of TEX from both *presentationally*- and *conceptually*-oriented mathematical content and how MathML may be used in the process. Our implementation conserves the implicit high-level semantics of macro use in both TEX and MathML. Since the expressions generated by mathematical software may be quite lengthy, we also discuss issues that arise in line-breaking.

## 1 Introduction

Most mathematical software systems allow one to export expressions in TEX format. For many mathematical ideas there are several choices of notation, and typically the TEX formulae generated by software systems use the notational conventions selected by the system designers. These choices might be quite different from what would be selected by the client of the package, if a choice were offered.

We are interested in the problem of generating TEX that respects the notational conventions that are preferred by the client. To see the difference between a default and a customized rendering of an expression, compare the formulae of Figure 1a and Figure 1b. In most cases it is not possible for a user to obtain TEX output that uses their preferred notation. If the user of a computer algebra system or other mathematical software package wishes to publish the results of a computation, he or she must either accept the presentation offered by the system, or rewrite the TEX content. In this paper we present an alternative approach, based on adding notation preferences to mathematical TEX converters.


Maple 10 - C:\Documents and Settings\elena\My Documents\maplePageForTUG.mw - [Server 3]
File Edit View Insert Format Table Plot Spreadsheet Sketch Tools Window Help
Text Math 2D Input Times New Roman 12
*Untitled (1) *Untitled (2) maplePageForTUG.mw
Ready
Memory: 5.99M Time: 1.10s Math Mode

```
> MathML :- ExportContent(CalculusExpr);
```

$$> \left(\frac{\partial}{\partial x} e^{(x+3)}\right)\left(\frac{d^3}{dx^3}(x+3)^{\left(\frac{1}{3}\right)}\right)\left(\frac{\partial^5}{\partial x^3\,\partial y^2}\left(x^y+y^x+x^3\right)\right)$$

(a) Default rendering by Maple

Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///z:/PROJECTS/CVS-ORCCA/MathML/M
Go
Getting Started
Latest Headlines
Done

$$(\exp(x+3))' \left((x+3)^{1/3}\right)''' \,\mathrm{D}_{\mathrm{xxxyy}}(x^y+y^x+x^3)$$

(b) Customized rendering via Mozilla

**Figure 1**: Two renderings of the same expression.

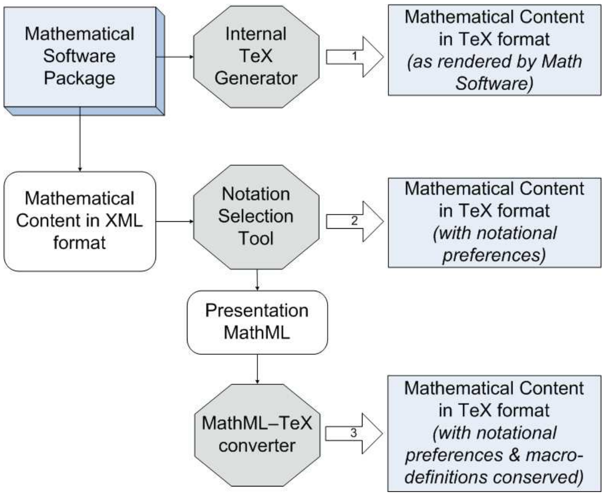

Mathematical Software Package
Internal TeX Generator
1
Mathematical Content in TeX format (as rendered by Math Software)
Mathematical Content in XML format
Notation Selection Tool
2
Mathematical Content in TeX format (with notational preferences)
Presentation MathML
MathML–TeX converter
3
Mathematical Content in TeX format (with notational preferences & macro-definitions conserved)

**Figure 2**: Three ways to generate TEX from a mathematical software system

There are several ways in which TEX for mathematical expressions may be generated from a software package such as Maple [1], Mathematica [2], Axiom [3], Aldor [4], *etc.* One approach is to make a direct translation to TEX from the internal expression representation of the software system. Exporting it to TEX directly from the system will then produce package-specific presentation of this content (Figure 2, arrow 1). Another approach may be taken if the mathematical object has been created in a web-oriented mathematical environment, such as with the MONET web-services [5, 6]. In this situation the expression will most likely be encoded using some XML-based standard, such as OpenMath [7] or Content MathML [8]. These formats are supported by most computer algebra systems as well as many web browsers and other packages. The objective of this paper is to demonstrate a flexible technique to generate of TEX presentation from these XML-based, semantically-oriented formats.

We explore two manners of producing TEX from XML formats: The first is to do so directly from mathematical content, taking into account notational settings (Figure 2, arrow 2). The second approach is to use a two-stage conversion (Figure 2, arrow 3). In this case, MathML — combined with elements using some extended set of tags — is first generated from the mathematical content. This extended MathML is then translated into TEX with corresponding TEX macros. In this case, high-level mathematical constructs may be mapped directly to TEX macros. This ensures that the semantics of the original expression are conserved in the output TEX content.

This paper is organized as follows: Section 2 describes how presentation of mathematical content can be customized using a Notation Selection Tool. Section 3 describes a MathML to TEX translator that is used in multi-stage conversion. Section 4 provides some details of how line breaking is achieved in the generated TEX. Section 5 presents our conclusions and outlines some possible directions for future work in this area.

## 2 Generating presentation from content using notational preferences

Anyone with more than a passing familiarity with the subject understands that there is no universal notation for mathematics. There are mathematical concepts for which there are several different notations, and there are notations for which there are several different mathematical concepts. Figure 3 shows how the choice of different notation can affect the appearance of mathematical content. This choice of notation is certainly the major determining factor in how an expression will appear.

### 2.1 A notation selection tool

In earlier work we described a Notation Selection Tool [9, 10] designed to control conversion of mathematical expressions in XML format. The original purpose of this tool was to provide a graphical user interface (Figure 4) for notation selection. The tool generates an XSLT stylesheet used to transform Content MathML to Presentation MathML using the desired notational conventions. The stylesheet and the generated Presentation MathML are determined by the user's choice of notation settings.

### 2.2 Extending the tool for direct conversion to TEX

In the present work we use a key feature of the Notation Selection Tool: Its extensible design allows one to add new translation directions without modifying the software implementation. In particular, we can add direct conversion of OpenMath and Content MathML to TEX.

We described in [9] how the Notation Selection Tool is initialized by a configuration file. This file is the only component in the design that provides information about the mathematical concepts that the converter can handle. It also stores the transformation rules to be applied for the selected notations.

Because all knowledge of the MathML conversion is contained in this configuration file, it may also be used to specify conversions involving other XML formats for input and other XML or text formats for output. Updating the configuration file

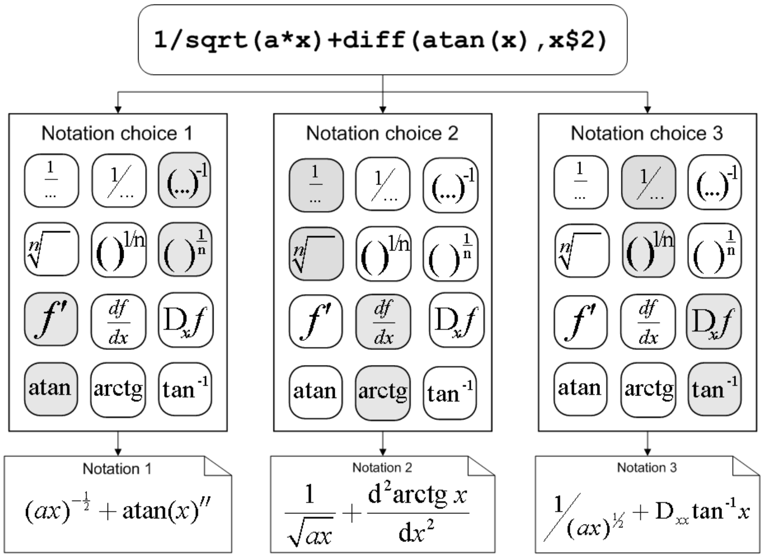

1/sqrt(a*x)+diff(atan(x),x$2)
Notation choice 1
Notation choice 2
Notation choice 3
atan
arctg
tan-1
Notation 1
$(ax)^{-\frac{1}{2}} + \mathrm{atan}(x)''$
Notation 2
$\frac{1}{\sqrt{ax}} + \frac{\mathrm{d}^2\mathrm{arctg}\,x}{\mathrm{d}x^2}$
Notation 3
$1/(ax)^{1/2} + \mathrm{D}_{xx}\tan^{-1}x$

**Figure 3**: Generating different notations for the same mathematical content

with new conversion rules, such as with OpenMath as source and TEX as target, allows the Notation Selection Tool to perform new conversions according to desired rules:

```
<catalog>
  <name> CALCULUS </name>
  <itemlist>
    <item>
      <keyword> PARTIAL DERIVATIVE </keyword>
      <content>
          OpenMath encoding for mathematical
          concept PARTIAL DERIVATIVE
      </content>
      <choicelist>
        <choice>
          <!-- The first notation choice for  -->
          <image src = "pd_1.gif"/>
          <keyvalue> 1 </keyvalue>
          <presentation>
             <converter input = "OpenMath" output="LaTeX">
                XSLT template for OpenMath
                to LaTeX for this notation
             </converter>
             ...
          </presentation>
        </choice>
        ...
      </choicelist>
    </item>
    ...
  </itemlist>
</catalog>
```

If the user selects the notation $\mathrm{D}_x$ for partial differentiation and $f'$ for ordinary differentiation, then, instead of obtaining the default output, the Maple expression shown in Figure 1a will be converted to

```
$${{\left(\mathop{exp}{\left({x+3}\right)}\right)}^\prime}\,
  {{({{\left({x+3}\right)}^{1/3}})}^{\prime\prime\prime}}\,
  {{\mathrm D}_{xxxyy}\left({{x^y}+{y^x}+{x^3}}\right)}$$,
```

which renders as

$$(exp\,(x+3))'\,\left((x+3)^{1/3}\right)'''\,\mathrm{D}_{xxxyy}\left(x^y+y^x+x^3\right).$$

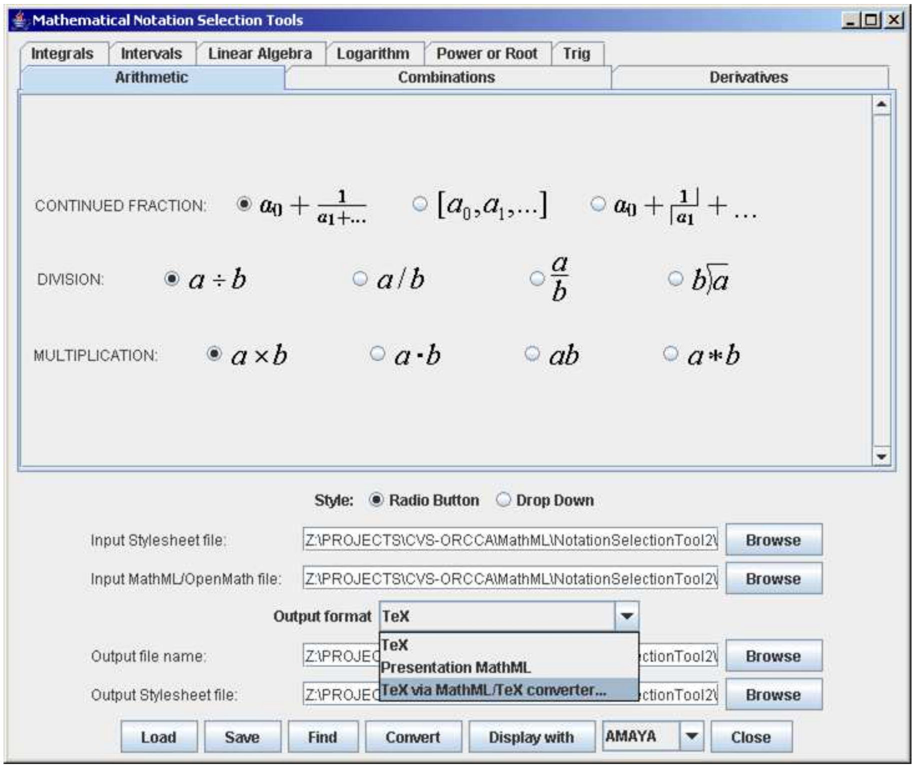
The configuration file may define more than one output format, for example both LaTeX and MathML. In this case, the conversion rules are selected according to the target format specified by the user (see Figure 4). This approach is a special case of that described in [11] for the conversion of mathematical documents into multiple forms.

### 2.3 The notation selection tool as a front-end in multi-stage conversion

We have discussed extension of the Notation Selection Tool by modifying the configuration file to provide new conversions *directly* to TEX. We can, instead, do the translation in a *series of stages* to increase flexibility.

We can use the Notation Selection Tool as an intermediate translation stage generating MathML. This MathML can then be further processed to produce TEX. This multi-stage mode of generating TEX from Content MathML or OpenMath can be specified by a menu selection in the Notation Selection Tool. In this mode, the Notation Selection Tool first generates MathML and then passes it to our configurable MathML to TEX converter [12].

This multi-stage process allows the Notation Selection Tool to generate extended markup (*i.e.* MathML and other XML) that may then later be transformed. To do this, the desired extended markup is placed in the output rules of the configuration file. This extended markup can use new tags to capture the semantics of new mathematical concepts or complex combinations of OpenMath or MathML constructs. For example, one might use a new XML



CONTINUED FRACTION: ● $a_0 + \frac{1}{a_1 + \dots}$ ○ $[a_0, a_1, \dots]$ ○ $a_0 + \frac{1|}{|a_1} + \dots$

DIVISION: ● $a \div b$ ○ $a/b$ ○ $\frac{a}{b}$ ○ $b\overline{)a}$

MULTIPLICATION: ● $a \times b$ ○ $a \cdot b$ ○ $ab$ ○ $a * b$

Style: ● Radio Button ○ Drop Down

Input Stylesheet file: Z:\PROJECTS\CVS-ORCCA\MathML\NotationSelectionTool2\

Input MathML/OpenMath file: Z:\PROJECTS\CVS-ORCCA\MathML\NotationSelectionTool2\

Output format: TeX (TeX / Presentation MathML / TeX via MathML/TeX converter...)

Output file name: Z:\PROJEC…ctionTool2\

Output Stylesheet file: Z:\PROJEC…ctionTool2\

Display with: AMAYA

**Figure 4**: The Notation Selection Tool, stand-alone application interface

element to capture the semantics of binomial coefficients or continued fractions, which do not appear in standard MathML. In such a case, a new transformation rule can be added directly to the configuration file of the Notation Selection Tool:

```
<converter input = "Content MathML" output="LaTeX">
  <xsl:template match = "apply/mmlx:choose[position()=1]
                        [count(child::*)=2]">
   <mmlx:binomial>
        <xsl:for-each select = 'mmlx:choose/child::*'>
           <xsl:copy-of select='.'/>
        </xsl:for-each>
   </mmlx:binomial>
  </xsl:template>
</converter>
```

The output of the XSLT transformation in the above example will contain Presentation MathML extended with the new tag `mmlx:binomial`. The prefix "`mmlx`" is defined at the beginning of the generated XML, and indicates that this element belongs to a different namespace than the standard MathML elements. The resulting element can be either expanded immediately after the conversion to obtain a combination of appropriate presentation markup and semantic annotation (possibly using a `<csymbol>` element). Alternatively, it may be carried on to the next step of the translation to TEX, as described in the next section.

## 3 MathML to TEX conversion

In earlier work we have explored the question of conversion between TEX and MathML, using a set of bidirectional transformation rules [13, 14, 15]. Here we summarize the aspects of the converter that are used in notation selection for TEX via MathML.

### 3.1 Modes of conversion

In [15] we presented a MathML to TEX translator that converts expressions in MathML representation to equivalent TEX expressions. The translator supports conversion at three different levels of content granularity: (1) entire files, (2) individual expressions and (3) separate objects. The last option allows the user to manipulate stand-alone MathML and TEX-objects obtained from sources other than the usual MathML or TEX documents.

To perform the file-level conversion, the translator processes an entire MathML file and produces a complete TEX document. The converter can also handle other XML files containing MathML and replace the MathML elements with their TEX equivalents, embedded as `CDATA` sections, as shown on Figure 5. This option is useful when our converter is used as a pre-processor for HTML to LATEX translators such as `html2latex` [16]. Thus, from a sequence of two translations we can produce TEX documents from HTML web pages containing mathematics.

### 3.2 Implementation

Even though the most natural choice for implementing a converter for XML-based languages such as MathML is via XSLT stylesheets, we have taken another approach. The main argument against using XSLT for this converter was our desire to provide a symmetric path for the inverse conversion from TEX to MathML. We chose to organize both of the translators based on *bidirectional* mappings between MathML and TEX constructs. Mapping rules describe the correspondence between TEX and MathML patterns in the following format:

```
<pat:template>
  <!-- TeX command with parameters -->
  <pat:tex op="\frac" params="\patVAR!{num}\patVAR!{den}"/>

  <!-- Corresponding MathML tree -->
  <pat:mml op="mfrac">
    <mfrac>
      <pat:variable name="num"/>
      <pat:variable name="den"/>
    </mfrac>
  </pat:mml>
</pat:template>
```

Templates, such as the above, are organized into mapping files. The same file can be used both for converting from TEX to MathML and *vice versa.* The converter tools are implemented in Java and read one or more mapping files as they are initialized. The converter may be run as a stand-alone application (Figure 6) or as a web service [12]. The same configuration file is used to control a transformer in the reverse direction [17], from TEX to MathML. While it is the core Java program that is carrying out the actual conversion, its *behavior* is defined by the selection of mapping files.

This approach provides the converter with the desired flexibility: none of the conversion rules are hard-coded and any of them may be updated by editing the corresponding templates in the mapping files. The consistency between the two directions of conversion is preserved, since any of the mapping files can be used by either of the converters.

New mappings can be added in a similar way. Whenever a new pattern, such as a TEX macro or XML template, is introduced, a new template can be added to a mapping file. This immediately enables the conversion using a new transformation rule.

### 3.3 Conserving high-level semantics in translation

In [13] and in Section 2.3 we have described how new mathematical constructs can be described with TEX macros or XSLT template definitions.

Macros are usually used as abbreviations for lengthy expressions, expressions that are particularly notable, or that appear more than once. These expressions typically have some meaning that makes them natural choices for expression by macros. When converting a mathematical document between formats, we wish to conserve whatever implicit semantics is captured by the macro markup. Expanding macros and then converting loses this information.

In [13] and [18] we showed that rather than expanding all macros to low-level formatting instructions, in many cases it is possible to map high-level markup in one setting to corresponding markup in another, thus conserving implied semantics. We may arrange that each TEX style or class file have a counterpart XSLT stylesheet for use with Presentation MathML, and each TEX macro have a corresponding XSLT template definition.

We now give a complete example: Suppose we are working with documents that involve the Lambert "W" function of two arguments $k$ and $z$, denoted $W_k(z)$. It would be possible to denote this explicitly as `W_k(x)` everywhere in a TEX document, and as corresponding presentation markup in a MathML document. We prefer, however, to define a TEX macro, such as

```
\newcommand{\LambertW}[2]{W_{#1}\left( {#2} \right)}
```

and a corresponding XSLT template

```
<xsl:template match="mmlx:LambertW">
  <mrow>
     <msub>
       <mo> W </mo>
       <xsl:apply-templates
            select = ’mmlx:LambertW/child::*[1]"/>
     </msub>
     <mfenced>
       <xsl:apply-templates
            select = ’mmlx:LambertW/child::*[2]"/>
     </mfenced>
   </mrow>
</xsl:template>
```

We also add a direct mapping rule for `\LambertW` and `<mmlx:LambertW>` to one of the translator mapping files:

```
<pat:template>
  <!-- TeX command -->
  <pat:tex op="\LambertW"
          params="\patVAR!{k} \patVAR!{z}"/>
  <!-- MathML element -->
  <pat:mml op="mmxl:LambertW">
    <mmxl:LambertW>
      <pat:variable name="k"/>
      <pat:variable name="z"/>
    </mmxl:LambertW>
  </pat:mml>
</pat:template>
```

Using this mapping, a MathML expression

```
<mmxl:LambertW>
  <mn> 1 </mn>
  <mi> z </mi>
</mmxl:LambertW>
```

```
<?xml version='1.0' encoding='utf-8'?>
<html>
 <h1> XHTML + MathML </h1>
 <ol>
   <li>
    <math>       ⇐ Expression 1
      <msup>
        <mi>x</mi>
        <mn>2</mn>
      </msup>
      <mo>+</mo>
      <mn>1</mn>
    </math>
   </li>
   <li>
    <math>       ⇐ Expression 2
     <msubsup>
       <mi> A </mi>
       <mi> i </mi>
       <mi> j </mi>
     </msubsup>
    </math>
   </li>
 </ol>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<html>
 <h1> HTML + MathML </h1>
 <ol>
  <li>
   <LaTeX xmlns='orcca.on.ca'> ⇐ Expression 1
     <![CDATA[$${x^2}+1$$]]>
   </LaTeX>
  </li>
  <li>
   <LaTeX xmlns='orcca.on.ca'> ⇐ Expression 2
    <![CDATA[$${A_i^j}$$]]>
   </LaTeX>
  </li>
 </ol>
</html>
```

**Figure 5**: Example of conversion from XHTML with MathML to TeX

will be translated to TeX as `\LambertW{1}{z}` instead of `W_1\left(z\right)`.

This approach, converting from MathML to TeX driven by high-level rules, allows a concept-level translation of user-defined macros. This preserves mathematical semantics implied by the markup of the original expression and, most importantly for the present paper, allows user-preferred notations to be given by alternative definitions of the target TeX macros.

## 4 Automated line breaking

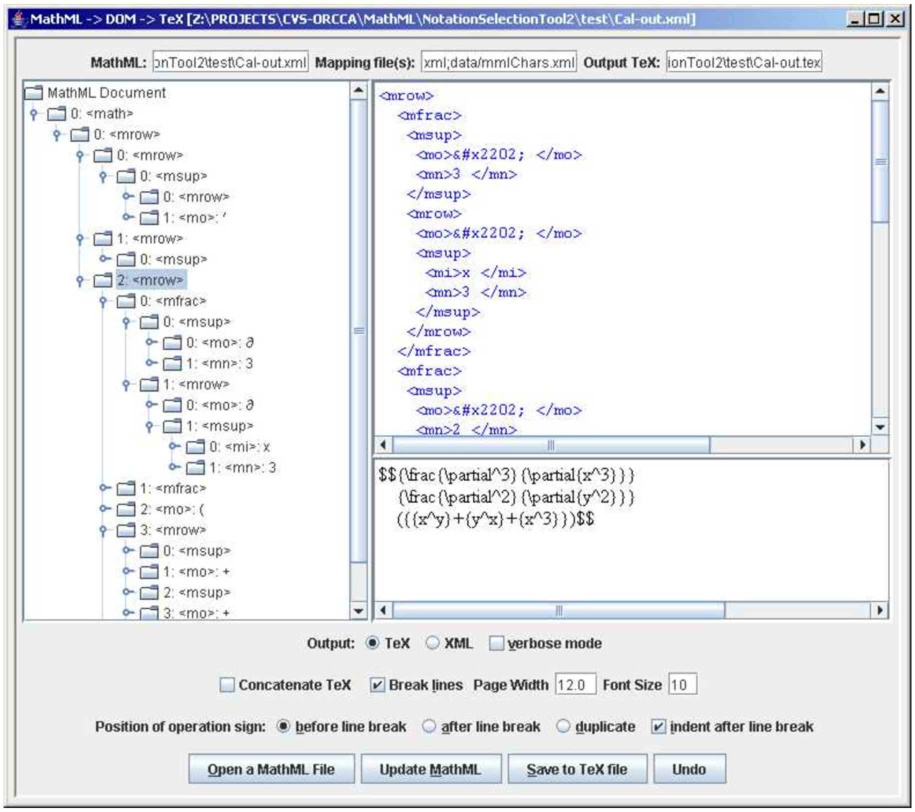
A secondary benefit of using a non-XSLT approach for conversion from MathML to TeX is that it makes automated line breaking of long expressions easier.

### 4.1 Motivation

In general, automated line-breaking of mathematical expressions is a complex problem and has been studied by designers of computer algebra systems for some three decades [19]. MathML browsers, such as Amaya, MathPlayer and those of the Netscape family, either have their own mechanism for line breaking in mathematical formulae, or provide a scrolling region for long expressions. TeX, however, does not natively support either of these options. Therefore long TeX formulae generated from MathML may not fit in the text area of the document. Very often composing a mathematical paper with long formulae will give results as shown in Figure 7.

Manual line breaking in TeX formulae is viable only when the size of the expression is relatively small. However, when MathML content is generated as output from a mathematical software package, or when numerous documents are to be converted, this approach is not sufficient. One solution would be to use the `breqn` package [20] to display generated TeX formulae. There are a number of reasons, however, why we have elected to provide line-breaking as part of the TeX generation:

- Generated TeX formulae can be very large, so a number of line breaking (and page breaking) issues arise that do not arise with hand-written equations. In particular, `breqn` fails to separate factors of long products with implied multiplication. We can handle these situations better than a human-oriented package such as `breqn`.
- Generated TeX formulae can be idiosyncratic, so providing our own line breaking of equations gives finer control.
- As part of the TeX generation we have already performed much of the analysis that is required for line breaking.
- As a purely practical consideration, not all TeX environments support the `breqn` package.



MathML -> DOM -> TeX [Z:\PROJECTS\CVS-ORCCA\MathML\NotationSelectionTool2\test\Cal-out.xml]

MathML: onTool2\test\Cal-out.xml Mapping file(s): xml;data/mmlChars.xml Output TeX: ionTool2\test\Cal-out.tex

```
MathML Document
  0: <math>
    0: <mrow>
      0: <mrow>
        0: <msup>
          0: <mrow>
          1: <mo>: ′
      1: <mrow>
        0: <msup>
      2: <mrow>
        0: <mfrac>
          0: <msup>
            0: <mo>: ∂
            1: <mn>: 3
          1: <mrow>
            0: <mo>: ∂
            1: <msup>
              0: <mi>: x
              1: <mn>: 3
        1: <mfrac>
        2: <mo>: (
        3: <mrow>
          0: <msup>
          1: <mo>: +
          2: <msup>
          3: <mo>: +
```

```
<mrow>
  <mfrac>
   <msup>
    <mo>&#x2202; </mo>
    <mn>3 </mn>
   </msup>
   <mrow>
    <mo>&#x2202; </mo>
    <msup>
     <mi>x </mi>
     <mn>3 </mn>
    </msup>
   </mrow>
  </mfrac>
  <mfrac>
   <msup>
    <mo>&#x2202; </mo>
    <mn>2 </mn>
```

```
$${\frac{\partial^3}{\partial{x^3}}}
  {\frac{\partial^2}{\partial{y^2}}}
  (({x^y}+{y^x}+{x^3}))$$
```

Output: ● TeX ○ XML □ verbose mode

□ Concatenate TeX ☑ Break lines Page Width 12.0 Font Size 10

Position of operation sign: ● before line break ○ after line break ○ duplicate ☑ indent after line break

**Figure 6**: The MathML to TEX converter, stand-alone application interface

### 4.2 Algorithm overview

The subject of line-breaking for mathematical formulae is complex, and a full description of our approach is beyond the scope of this paper. For the purpose of this article, we highlight only some of the important aspects.

In a manner similar to the *total-fit* approach to line breaking implemented for TEX paragraphs by Knuth and Plass [21], our algorithm searches for all possible breakpoints in a formula and tries to find the combination of line breaks that will produce the best global arrangement.

In addition to splitting linear text, the method must take into account the two-dimensional nature of mathematical content and also consider the implicit semantics of expressions.

This leads to a number of constraints. For example, script sub-expressions may not be separated from their base expressions. Another common case is that of juxtaposition: Immediate function arguments should not be separated from the name of the function, *e.g.* we must avoid

$$\begin{array}{l} p(x-y,z)\ \sin \\ (2x+1)\ , \end{array}$$

but implicit multiplication can be split between the factors

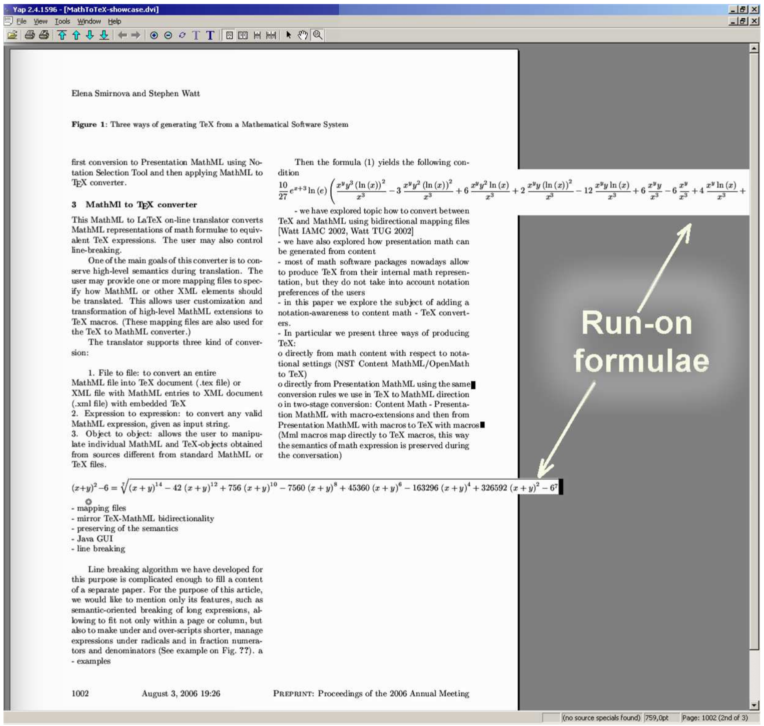
$$\begin{array}{l} p(x-y,z)\times \\ \sin(2x+1)\ . \end{array}$$

Juxtaposition is difficult to distinguish, since function application and implicit multiplication often have no explicit indication which operation is intended (even though MathML provides invisible operators for this purpose).

In general, every possible breakpoint is assigned a “penalty” value, so for example, signs and relations, such as $=$, $\Rightarrow$, *etc.*, as well as $+$ and $-$ are usually given priority over division and multiplication. Preference is given to breaking a formula closer to the root of the expression tree than within a branch. This means that a product of sums will preferentially break at the multiplications, unless the other penalties make this an overwhelmingly bad choice.

### 4.3 Customization

In the previous example we saw that when multiplication is expressed implicitly, line breaking may force the addition of an explicit multiplication sign. This may be represented by a central dot $\cdot$, times $\times$,



Elena Smirnova and Stephen Watt

**Figure 1**: Three ways of generating TeX from a Mathematical Software System

first conversion to Presentation MathML using Notation Selection Tool and then applying MathML to TEX converter.

**3 MathMl to TEX converter**

This MathML to LaTeX on-line translator converts MathML representations of math formulae to equivalent TeX expressions. The user may also control line-breaking.

One of the main goals of this converter is to conserve high-level semantics during translation. The user may provide one or more mapping files to specify how MathML or other XML elements should be translated. This allows user customization and transformation of high-level MathML extensions to TeX macros. (These mapping files are also used for the TeX to MathML converter.)

The translator supports three kind of conversion:

1. File to file: to convert an entire
MathML file into TeX document (.tex file) or
XML file with MathML entries to XML document (.xml file) with embedded TeX
2. Expression to expression: to convert any valid MathML expression, given as input string.
3. Object to object: allows the user to manipulate individual MathML and TeX-objects obtained from sources different from standard MathML or TeX files.

$$(x{+}y)^2{-}6 = \sqrt[7]{(x+y)^{14} - 42\,(x+y)^{12} + 756\,(x+y)^{10} - 7560\,(x+y)^{8} + 45360\,(x+y)^{6} - 163296\,(x+y)^{4} + 326592\,(x+y)^{2} - 6^7}$$

- mapping files
- mirror TeX-MathML bidirectionality
- preserving of the semantics
- Java GUI
- line breaking

Line breaking algorithm we have developed for this purpose is complicated enough to fill a content of a separate paper. For the purpose of this article, we would like to mention only its features, such as semantic-oriented breaking of long expressions, allowing to fit not only within a page or column, but also to make under and over-scripts shorter, manage expressions under radicals and in fraction numerators and denominators (See example on Fig. ??). a
- examples

Then the formula (1) yields the following condition

$$\frac{10}{27}e^{x+3}\ln(e)\left(\frac{x^y y^3(\ln(x))^2}{x^3} - 3\,\frac{x^y y^2(\ln(x))^2}{x^3} + 6\,\frac{x^y y^2\ln(x)}{x^3} + 2\,\frac{x^y y(\ln(x))^2}{x^3} - 12\,\frac{x^y y\ln(x)}{x^3} + 6\,\frac{x^y y}{x^3} - 6\,\frac{x^y}{x^3} + 4\,\frac{x^y\ln(x)}{x^3} + \right.$$

- we have explored topic how to convert between TeX and MathML using bidirectional mapping files [Watt IAMC 2002, Watt TUG 2002]
- we have also explored how presentation math can be generated from content
- most of math software packages nowadays allow to produce TeX from their internal math representation, but they do not take into account notation preferences of the users
- in this paper we explore the subject of adding a notation-awareness to content math - TeX converters.
- In particular we present three ways of producing TeX:
o directly from math content with respect to notational settings (NST Content MathML/OpenMath to TeX)
o directly from Presentation MathML using the same conversion rules we use in TeX to MathML direction
o in two-stage conversion: Content Math - Presentation MathML with macro-extensions and then from Presentation MathML with macros to TeX with macros (Mml macros map directly to TeX macros, this way the semantics of math expression is preserved during the conversation)

1002 August 3, 2006 19:26 PREPRINT: Proceedings of the 2006 Annual Meeting

**Figure 7**: TEX article without line breaking in long formulae.

asterisk ∗ or other operator ⊗, ⊙, .... Our package allows the user to specify which is preferred.

Likewise, the positioning of the operator at a line break can be set to the upper line, the lower line, or repeated on both lines (see Figure 8). In addition various indentation styles may be desired for the multiple lines. These options reflect different notational and cultural preferences, and can be customized by the user of the converter through the GUI interface or command line.

### 4.4 Line breaking in sub-expressions

As well as allowing a formula to fit within a given page or column width, the line breaking algorithm also handles situations where certain sub-expressions are too large for conventional line-breaking. These situations include long scripts (compare Figures 9 and 10), in fraction numerators and denominators (Figure 11), and expressions under radicals (Figure 12).

The sizes of the bounding boxes for the folded sub-expressions are normally calculated automatically, based on an optimal fit in the text area. If desired, they may instead be specified by the user. For example, for sub-scripts this is specified as a maximum ratio of the script width to the width of the base expression.

$$\begin{array}{l} x_1 - y_2 + z_3 + \\ \quad \alpha - \beta^2 + \gamma^3 + \\ \quad f(1) + g(2) \end{array}$$

(a) Before break

$$\begin{array}{l} x_1 - y_2 + z_3 \\ +\alpha - \beta^2 + \gamma^3 \\ +f(1) + g(2) \end{array}$$

(b) After break

$$\begin{array}{l} x_1 - y_2 + z_3 + \\ +\alpha - \beta^2 + \gamma^3 + \\ +f(1) + g(2) \end{array}$$

(c) Before and after break

**Figure 8**: Operator placement at line breaks

For large expressions, line breaking is a true two-dimensional problem involving box composition. In this setting, the height and width of individual terms depend on the choices made in displaying its sub-expressions. Examples of such nested line breaking are shown in Figures 10, 11 and 12.

As a practical implementation detail, we use the `array` environment to organize multi-line output in mathematical expressions. This is shown in

$$\sum_{(i,j)\in\{(\alpha,\beta)\ \mid\ ax^{\alpha}+bx^{\beta}=m,\ m\in R^{+},\ \alpha\neq\beta,\ \gcd(a,b)=1\}} f(\ldots$$

**Figure 9**: Expression without line breaking does not fit in a text column

$$\sum_{(i,j)\in\left\{(\alpha,\beta)\,\middle|\,\begin{array}{l} ax^{\alpha}+bx^{\beta}=m,\\ m\in R^{+},\ \alpha\neq\beta,\\ \gcd(a,b)=1\end{array}\right\}} f(i)g^{-1}(j)\phi(i+j,i-j)$$

**Figure 10**: Adding line breaking in subscript allows the whole expression to fit in a column

$$\begin{array}{l}\frac{10}{27}\,y^2\,\mathrm{e}^{(x+3)}\,\sqrt[3]{(x+3)}\,\mathrm{W}_0(1.5+2.5\,\imath)\times\\ \dfrac{\left(\begin{array}{l} x^{(y-3)}\ln{(x)}^2\,y^3+6\,x^{(y-3)}\ln{(x)}\,y^2+\\ 6\,x^{(y-3)}\,y-3\,x^{(y-3)}\ln{(x)}^2\,y^2-\\ 12\,x^{(y-3)}\ln{(x)}\,y-6\,x^{(y-3)}+\\ 2\,x^{(y-3)}\ln{(x)}^2\,y+4\,x^{(y-3)}\ln{(x)}+\\ y^{(x-2)}\,x^2\ln{(y)}^3-y^{(x-2)}\,x\ln{(y)}^3+\\ 6\,y^{(x-2)}\,x\ln{(y)}^2+6\,y^{(x-2)}\ln{(y)}-\\ 3\,y^{(x-2)}\ln{(y)}^2-14\,y^{(x-2)}\,x\ln{(y)}\end{array}\right)}{{(x+3)}^3}\end{array}$$

**Figure 11**: Line breaking in long fractions

$$(x+y)^2-6=\sqrt[7]{\begin{array}{l}{(x+y)}^{14}-42\,{(x+y)}^{12}+\\ 756\,{(x+y)}^{10}-7560\,{(x+y)}^{8}+\\ 45360\,{(x+y)}^{6}-163296\times\\ {(x+y)}^{4}+326592\,{(x+y)}^{2}-6^7\end{array}}$$

**Figure 12**: Line breaking in long sub-radical expressions

the code fragment of Figure 13. Note the command `\displaystyle` preceding every new line. This ensures rendering of fractions and scripts in display mode. To see the effect, compare the appearance of the formula with fractions of Figure 14a and Figure 14b.

## 4.5 Open questions

```
$$
\begin{array}{l}
 \displaystyle A\,\frac{x+y}{3}+{14\,t}^{3}-\\
 \displaystyle B\,\frac{a-b}{4}+{12\,t}^{4}+\\
 \displaystyle C\,\frac{m+n}{5}+{10\,t}^{5}
\end{array}
$$
```

**Figure 13**: Encoding of multi-line constructions

$$\begin{array}{l} A\,\frac{x+y}{(a+b)^3}+14m^{12}-\\ B\,\frac{a-b}{(x-y)^3}+12n^{14}\end{array}$$

(a) In-line style

$$\begin{array}{l} \displaystyle A\,\frac{x+y}{(a+b)^3}+14m^{12}-\\ \displaystyle B\,\frac{a-b}{(x-y)^3}+12n^{14}\end{array}$$

(b) Display style

**Figure 14**: Array elements (a) without and (b) with explicit display style.

One of the challenges in line breaking is to distinguish implicit multiplication and application of unspecified or user-defined functions. We may assume function application in the cases of explicit markup or known functions, such as `\mathop{tg} \alpha` or `\tan x`. In general, however, we need either good heuristics, non-trivial semantic analysis, or explicit user markup.

Another challenge, this time from the aesthetic point of view, is how to arrange mixed expressions, such as shown in Figure 12, where we decide to maintain a single baseline for the overall expression and to align sub-expressions in place, instead of moving them to separate lines and splitting them there.

The final issue we mention is the question of page breaking in the case of multi-page content. This situation frequently arises with output of computer algebra systems. We have implemented an approach that takes page size into account, but there remain a large number of questions with respect to handling of subexpressions and layout choices.

## 5 Conclusions and future work

We have explored an alternative approach to generating TEX expressions from mathematical content when the content is presented in a conceptually oriented format.

The main idea of our approach is to maintain the mathematical markup at a high level, either in TEX or MathML, allowing extended markup for new mathematical concepts. This allows higher-level transformations among the formats and allows late binding of user-specified notational choices.

We have shown how the rendering of mathematical content with TEX can be customized with our Notation Selection Tool. For this, we considered two methods of conversion to TEX: One as a direct translation using features of the Notation Selection Tool. The second method was to use MathML extended with new elements. We showed that the second approach offers a more fine-grained control over the conversion process. It allows implicit semantics of mathematical expressions to be mapped from MathML template definitions to TEX macros. Additionally, it allows a line breaking implementation suitable for large, generated mathematical expressions.

We continue to explore certain open problems in the conversion between MathML and TEX. These include the automatic generation of templates for mapping rules and XSLT templates for notation conversions. We also wish to further investigate enhanced expression breaking methods in the presence of selectable notations. In particular, we intend to explore generating line-breaking hints for a late stage line breaker (*i.e.* one that operates after notation specialization). Another point of interest for our group in the MathML to TEX conversion area is in automated generation of TEX style files from XML cascading style sheets [22].

## References

[1] *Maple User Manual*, Maplesoft, a division of Waterloo Maple Inc., 2005.

[2] *Mathematica*, Wolfram Research, Inc., 2004, `http://www.wolfram.com`.

[3] Richard D. Jenks and Robert Sutor, *AXIOM: the scientific computation system*, Springer-Verlag, New York, 1992.

[4] S.M. Watt, *Aldor*, pp. 265–270, in Handbook of Computer Algebra J. Grabmeier, E. Kaltofen, V. Weispfenning (editors) , Springer Verlag, Heidelberg, 2003.

[5] Mathematics on the Net, Symbolic Services, 2003, `http://www.orcca.on.ca/MONET/`.

[6] Mike Dewar, Elena Smirnova and Stephen M. Watt, *XML in Mathematical Web Services*, Proc. XML 2005 Conference — Syntax to Semantics, Nov 14–18, 2005, Atlanta GA, USA, `http://www.idealliance.org/proceedings/xml05/`.

[7] S. Buswell, O. Caprotti, D.P. Carlisle, M.C. Dewar, M. Gaetano, M. Kohlhase, et al., *The OpenMath Standard 2.0*, 2004, `http://www.openmath.org/cocoon/openmath/standard/om20/index.html`.

[8] R. Ausbrooks et al. *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*, World Wide Web Consortium Recommendation, 21 October 2003, `http://www.w3.org/TR/2003/REC-MathML2-20031021`.

[9] Elena Smirnova and Stephen M. Watt, *Notation Selection in Mathematical Computing Environments*, pp. 339–355, Proc. Transgressive Computing 2006: A conference in honor of Jean Della Dora (TC 2006), April 24–26 2006, Granada, Spain.

[10] The notation selection on-line tool, 2002, `http://www.orcca.on.ca/MathML/NotationSelectionTool/`.

[11] William Naylor and Stephen M. Watt, *Meta-Stylesheets for the Conversion of Mathematical Documents into Multiple Forms*, Annals of Mathematics and Artificial Intelligence, Vol. 38, pp. 3–25, 2003.

[12] MathML to TEX on-line converter, 2001, `http://www.orcca.on.ca/mathml/texmml/textomml.html`.

[13] Stephen M. Watt, *Exploiting Implicit Mathematical Semantics in Conversion between TEX and MathML*, Proc. Internet Accessible Mathematical Communication, (IAMC 2002), July 2002, Lille, France, `http://www.symbolicnet.org/conferences/iamc02`.

[14] S.M. Watt, *Conserving Implicit Mathematical Semantics in Conversion between TEX and MathML*, *TUGboat*, Vol. 23, No. 1, p. 108, 2002, `http://www.tug.org/TUGboat/Articles/tb23-1/watt.pdf`.

[15] E. Smirnova and S.M. Watt, *MathML to TEX Conversion: Conserving High-Level Semantics in Translation*, International Conference on MathML and Math on the Web (MathML 2002), June 28–30 2002, Chicago, USA, 2002, `http://www.mathmlconference.org/2002/presentations/smirnova/`.

[16] HTML to LATEX converter, `http://www.rpi.edu/~sofkam/html2latex/1.0/common/doc/html2latex.html`,

[17] TEX to MathML on-line converter, `http://www.orcca.on.ca/mathml/texmml/mmltotex.html`, 2001.

[18] Igor Rodionov and Stephen M. Watt, *Content Faithful Stylesheets for MathML*, Ontario Research Centre for Computer Algebra, University of Western Ontario, Research Report TR-00-14, 2000.

[19] John Keith Foderaro, *Typesetting MACSYMA equations*, Proc. 2nd. MACSYMA User's Conference, June 1979.

[20] Michael Downes, *Breaking equations*, *TUGboat*, Volume 18, No. 3, Proceedings of the 1997 Annual Meeting, 1997, `http://www.tug.org/TUGboat/Articles/tb18-3/tb56down.pdf`.

[21] D.E. Knuth and M.F. Plass, *Breaking paragraphs into lines*, Software — Practice and Experience, 1981.

[22] *Cascading Style Sheets*, `www.w3.org/Style/CSS/`.

# DVI2SVG: Using LaTeX layout on the Web

Adrian Frischauf and Paul Libbrecht
German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3
66123 Saarbrücken
Germany
adrianf (at) activemath dot org, paul (at) activemath dot org
http://www.activemath.org/~adrianf

### Abstract

The problem of presenting mathematical formulas on the Web is non-trivial. Current systems offer only partial answers to such requirements as the guaranteed layout on the client side or the availability of font glyphs. We describe DVI2SVG, a system to convert TeX's output into Scalable Vector Graphics. This approach responds to the requirements above and several others. We also present how it has been put to use in ActiveMath, a learning environment on the Web which presents mathematical documents personalized to each learner.

## 1 Different approaches to supporting mathematics on the Web

Classically, learning content is presented on the Web using the HTML format. This format, however, is unable to provide rich graphical constructs that are needed to render normal mathematical expressions. HTML's layout capabilities are limited, unable to fully render such constructs as the square-root or a fraction with proper baseline alignment. Mathematical formulas also often use characters which may be unavailable on some operating systems while HTML offers no method to ensure that a given font glyph will be available.

Using images for formulas solves these two issues but introduces several other problems. The resulting formula has no way to align properly inside a line of text, is not scalable, and does not adapt to changing text size. Moreover the separate parts of the formula can not be addressed as different objects by interactive scripts running on the client.

Presentation MathML [3] has the potential to realize a full featured mathematical presentation but current browser support has several drawbacks. As with HTML, the font glyphs have to be available to display the special characters correctly.

Another possibility for the delivery of mathematical content over the Web is to use PDF documents. This has no problem with fonts or layout but the PDF document does not allow much interactivity. As an "E-Paper" it is an offline resource and not an online presentation format.

Our DVI2SVG implementation has thus tried to answer the following requirements:

- provide layout quality as high as that of LaTeX, for text, formulas, and mixtures of both;
- deliver the content with guaranteed availability of font glyphs when presented to the client;
- present the content on a platform which can be dynamically scripted.

Moreover, we wished to integrate such a solution in the ActiveMath learning environment which combines and caches individual paragraphs before being personalized and delivered to the clients.

The specification of clients for the Scalable Vector Graphics provides an answer to all these requirements and was chosen for this reason.

## 2 The Scalable Vector Graphics format

The SVG file format [4] is an XML [1] language for describing two-dimensional vector graphics. It was issued as a recommendation by the W3C in 2003. This format allows for easy editing by hand, as well as easy generation, because of the many libraries available for manipulating XML. SVG allows font glyphs to be embedded within the document presented, and supports and specifies document object model access through a scripting API. Because of its graphical nature, the SVG format is able to display a complete layout faithfully even though it is unable to compute the layout itself. We thus put to use the well-known quality of LaTeX layout to create a document rendered using the modern SVG specification.

## 3 DVI2SVG

DVI2SVG is a converter for DVI files, the format output by TeX and LaTeX. It is written in Java and processes streams of DVI tokens. It parses the DVI input file and generates events, each event representing a command of the input file and holding the current state of the page (position, font, etc.). The Writer interprets the commands and produces the vector graphics XML.

The DVI format, in contrast to SVG, is a document format with multiple pages. For each page of the input, DVI2SVG produces a separate SVG file. Each of these pages contains a header with the font definitions for this specific page. Since the fonts tend to be large, only the font glyphs used in the page are actually included. Especially with fonts used for the formulas, partial glyph embedding saves a large amount of space, since typically only a few characters of each math font are used in a page.

DVI2SVG makes use of the classical TEX fonts as translated to SVG by Michel Goossens [5], whose script converts entire fonts into their SVG glyph equivalents.

The TEX character encoding is used within this conversion. This poses a problem since the TEX fonts contain character codes which are invalid in an XML document. The '`&#06;`' (the Sigma character $\Sigma$) is an example of such. The solution used is to map the TEX character codes to the Unicode private area above `0xE000`. No special characters which break the document occur. The resulting SVG source document is not human readable and is also unusable for Web-robots. This issue is only temporary, as implementations such as the Hermes translator[1] show that it is possible to extract good Unicode text from DVI output.

As a command-line tool, DVI2SVG can be used to process static DVI files and publish scientific documents in SVG on the Web.

### 3.1 Support for additional LATEX packages

In addition to the basic DVI commands, DVI2SVG also supports additional features. LATEX packages such as `color`, `hyperref` or `graphicx` use special commands to enrich a document. They are also translated to SVG.

A TEX command such as `\special{abc}` is copied as ASCII text into the DVI file as a special event. DVI2SVG defines a custom language which it is able to interpret. Since SVG is an XML format, the language of the specials is closely related to that. Such a special command in the DVI file looks like:

```
svg: rect @x=0 @y=0 @width=10 @height=10 /rect
```

This is transformed into an XML fragment:

```
<rect x="0" y="0" width="10" height="10"/>
```

It is thus possible for an author with little knowledge of SVG to enrich the document with the graphics and interactivity that the SVG format supports.

Using this protocol, drivers for the `color` and `graphicx` packages have been designed for DVI2SVG. By compiling LATEX documents to DVI using these drivers, pictures, text in colors, rotated texts, etc., can be embedded using the same macros as those used, for example, for PDF documents. The supported image file formats include SVG, GIF, JPEG, and PNG.

The creation of links with the `hyperref` package is also supported. Since DVI2SVG produces one file per input page, in-document links will be translated into links to the SVG file for the corresponding page.

## 4 Integration with ActiveMath

ActiveMath is a Web-based intelligent learning environment [8]. It presents mathematical documents transformed from OMDoc [6], an XML language for semantically representing mathematical documents. The mathematical formulas in OMDoc are encoded in OpenMath [2].

The presentation engine of ActiveMath [10] has been designed to support the dynamic generation of content presentation from OMDoc fragments such as definitions or examples. It generates documents in different output formats such as HTML, XHTML + MathML, SVG, and PDF.

It first extracts the OMDoc fragments, injects related objects, and applies an XSLT transformation: this is done exactly once per fragment and per language, and is cached. Once queries from learners arrive, this cached result is interpreted to inject personalization parameters:

- special mathematical notations are chosen depending on the context and user;
- fragments are presented in an order that may be particular to the learner (the learner can edit his own books, and have them created by a course generator [9]);
- exercise links have to be generated with user information;
- traces of the learner-model are output within the presentation, which helps the learner's motivation and tracking of his progress.

The resulting SVG fragments are assembled using stream combination and the high-performance

[1] Hermes is a translator from LATEX to XHTML+MathML, see `http://hermes.roua.org/`.

template engine Velocity.[2] This architecture realizes an effective simultaneous delivery to classrooms of learners.

The resulting presentation is enriched with interactivity: for example, at the time of fragment extraction, each mathematical symbol used semantically in the OMDoc source is annotated with its title. The XSLT transformation outputs the necessary `\special` commands so that scripting code on the client brings up a tooltip-like layer above the presented symbol. Potentially, other interactive features made possible by the semantic nature of the source could be provided, for example, the formula sub-term highlighting, context menus, and drag-and-drop presented in [7].

### 4.1 Example conversion

We present the processing steps into SVG and see where caching is possible and where personalization happens. In the first step, the sources are fetched from the database. Such a source might look like:

```
<definition for="SVG">
  <metadata>
    <Title>
    Definition of SVG
    </Title>
  </metadata>
  <CMP xml:lang="en">
    SVG stands for Scalable Vector Graphics
    and is a Web graphics format.
    <OMOBJ>
      <OMA>
        <OMS name="divide"/>
        <OMI value="1"/>
        <OMA>
          <OMS name="sqrt"/>
          <OMI value="2"/>
        </OMA>
      </OMA>
    </OMOBJ>
  </CMP>
</definition>
```

Then, using XSLT, this source is transformed into the following LaTeX fragments:

```
...
\begin{fragment}
\section*{Definition of SVG}
SVG stands for Scalable Vector Graphics
and is a Web graphics format.
$\frac{1}{\sqrt{2}}$
\end{fragment}
...
```

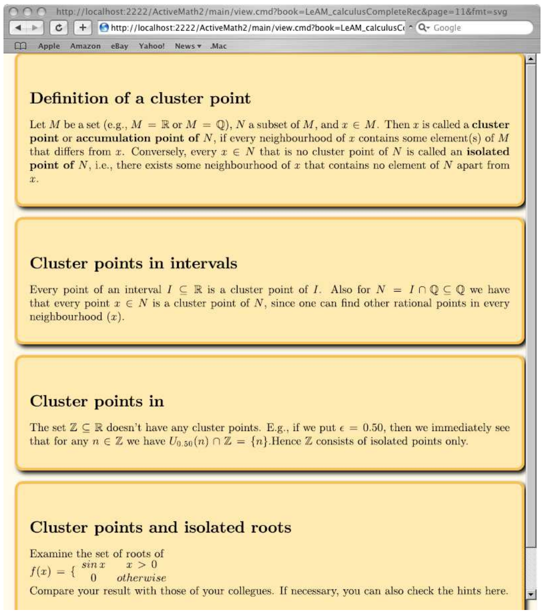
Such a fragment is translated to SVG as follows. For readability, the private Unicode-range text parts were replaced by their ASCII representation. In this SVG document one can see the fine control over the horizontal positioning of each glyph (in the precise `x` values), one of the major ingredients of LaTeX's layout's quality.

```
<rect x="0" y="0" width="468" height="690"
  stroke="none" fill="none" />
<g>
  <text font-family="CMBX" font-size="16.97">
    <tspan y="99.30" x="61.69 76.32 85.02
     95.63 106.24 111.54 118.96 124.26 133.81
     150.77 160.31 172.51 183.11 197"
     >Definition of SVG</tspan>
  </text>
  <text font-family="CMR" font-size="11.78">
    <tspan y="125.20" x="61.69 68.1 76.43
     89.32 93.88 98.36 104.13 110.54 116.95
     125.34 128.87  134.64 142.97 149.38
     154.50 160.27 163.48 169.25  175.65
     178.86 187.83 195.52 200.65 205.77
     210.26 216.03 224.36 233.41 237.90
     243.66 250.07 256.48 259.69 264.81
     273.21 278.98 285.39 295.64 298.85
     307.24 316.86 327.75 332.87 343.13
     348.90 353.38 359.15 365.56 371.97
     375.17 380.30 388.70 392.22
     397.99 402.48 412.09 417.86 422.34"
     >SVG stands for Scalable
      Vector Graphics and is a Web
      graphics format.</tspan>
  </text>
  <rect x="431.85" y="122.062"
    width="11.13" height="0.39"
    fill="Black" stroke="Black"
    stroke-width="0.1" />
  <text font-family="CMR" font-size="7.85">
    <tspan y="120.56" x="435.33">1</tspan>
  </text>
  <rect x="438.81" y="123.20" width="4.17"
     height="0.35" stroke-width="0.1"
     fill="Black" stroke="Black" />
  <text font-family="CMSY" font-size="7.85">
    <tspan x="431.9" y="123.6"
      >&#0xE017;</tspan>
  </text>
  <text font-family="CMR" font-size="7.85">
    <tspan y="130.08" x="438.81">2</tspan>
  </text>
</g>
```

These fragments are embedded into a document and are then processed into DVI using LaTeX. The conversion to SVG files using DVI2SVG is invoked. Similar to caching of HTML fragments (for serving HTML to clients), the SVG fragments are now cached.

In the last step, the SVG fragments are assembled to build a page using Velocity. The Velocity

[2] See `http://jakarta.apache.org/velocity/`


http://localhost:2222/ActiveMath2/main/view.cmd?book=LeAM_calculusCompleteRec&page=11&fmt=svg
http://localhost:2222/ActiveMath2/main/view.cmd?book=LeAM_calculusC
Google
Apple
Amazon
eBay
Yahoo!
News
.Mac

**Definition of a cluster point**

Let $M$ be a set (e.g., $M = \mathbb{R}$ or $M = \mathbb{Q}$), $N$ a subset of $M$, and $x \in M$. Then $x$ is called a **cluster point** or **accumulation point of** $N$, if every neighbourhood of $x$ contains some element(s) of $M$ that differs from $x$. Conversely, every $x \in N$ that is no cluster point of $N$ is called an **isolated point of** $N$, i.e., there exists some neighbourhood of $x$ that contains no element of $N$ apart from $x$.

**Cluster points in intervals**

Every point of an interval $I \subseteq \mathbb{R}$ is a cluster point of $I$. Also for $N = I \cap \mathbb{Q} \subseteq \mathbb{Q}$ we have that every point $x \in N$ is a cluster point of $N$, since one can find other rational points in every neighbourhood $(x)$.

**Cluster points in**

The set $\mathbb{Z} \subseteq \mathbb{R}$ doesn't have any cluster points. E.g., if we put $\epsilon = 0.50$, then we immediately see that for any $n \in \mathbb{Z}$ we have $U_{0.50}(n) \cap \mathbb{Z} = \{n\}$.Hence $\mathbb{Z}$ consists of isolated points only.

**Cluster points and isolated roots**

Examine the set of roots of

$$f(x) = \{ \begin{matrix} sin\, x & x > 0 \\ 0 & otherwise \end{matrix}$$

Compare your result with those of your collegues. If necessary, you can also check the hints here.

**Figure 1**: An ActiveMath page using SVG.

page provides an SVG skeleton, includes the fonts, and embeds the other SVG fragments. This step not only collects all the items for the page but also determines the final appearance of the page and the personalized appearance of the items. In the situation shown in Figure 1, for example, yellow shaded rounded rectangles are put around each item.

## 5 Conclusion

We have presented the DVI2SVG processor and how it has been integrated into ActiveMath.

Compared to other approaches of putting mathematics on the Web, DVI2SVG appears to offer interesting promises:

- Compared to PDF-based solutions, DVI2SVG offers richer interactivity, being based on standard scripting features.
- Compared to other solutions to convert TEX-based files to HTML or MathML, DVI2SVG ensures available fonts and the highest quality of layout provided by the classical (LA)TEX algorithms.
- compared to approaches which make use of the Flash player,[3] DVI2SVG is more open due to using its XML format; moreover, fragments are easier to combine. A feature of the Flash player format that we have not yet found in SVG, however, is the ability to embed an SVG document within another SVG document and preserve all interactivity in the embedded document; this could have avoided the repeated delivery of the SVG fonts for TEX.

SVG appears to be a real opportunity for Web-based presentation of TEX documents for the future. For now, some drawbacks remain: mainly that SVG players which have to be installed before one can use any SVG abilities; SVG support is emerging in developer versions of the Mozilla and Safari browsers,[4] but this support is incomplete; in particular, it is lacking the ability to render embedded fonts, a fundamental ingredient of the DVI2SVG approach and the only way to ensure that all font glyphs will be available on the client.

At present, the Java-based SVG viewer Batik[5] and the latest version of the Adobe SVG plugin,[6] which is available only for Windows, are the only players which work well with DVI2SVG.

We hope to see a renewal of the Adobe family of plugins following the merger of the two competing vector graphics leaders: Adobe and MacroMedia could bring the widely available Flash vector graphics plugins to fully support the SVG format.

## References

[1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C Recommendation PR-xml-971208, World Wide Web Consortium, December 1997. `http://www.w3.org/TR/PR-xml.html`.

[2] Stephen Buswell, Olga Caprotti, David Carlisle, Mike Dewar, Marc Gaëtano, and Michael Kohlhase. The OpenMath standard, version 2.0. Technical report, The OpenMath Society, June 2004. Available from `http://www.openmath.org/`.

[3] D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical markup language, version 2.0, 2001. `http://www.w3.org/TR/MathML2/`.

[4] Jon Ferraiolo, Jun Fujisawa, and Dean Jackson. Scalable vector graphics (SVG) 1.1 specification. Technical report, World Wide Web Consortium, 2003. Available from `http://www.w3.org/TR/SVG11/`.

[3] The Flash player presents animated vector graphics using a widespread plugin; see `http://www.macromedia.com`.

[4] For the current status, see `http://www.mozilla.org/projects/svg/status.html` and `http://webkit.org/projects/svg/status.xml`.

[5] The Batik SVG toolkit is available at `http://xml.apache.org/batik`; it is a Java library which supports downloaded fonts but with limited performance.

[6] The Adobe SVG plugin is available at `http://www.adobe.com/svg`.

[5] Michel Goossens and Vesa Sivunen. LaTeX, SVG, fonts. *TUGboat*, 22(4):269–281, 2001. Available from `http://tug.org/TUGboat/Articles/tb22-4/tb72goos.pdf`.

[6] M. Kohlhase. OMDoc: Towards an OpenMath representation of mathematical documents. Seki Report SR-00-02, Fachbereich Informatik, Universität des Saarlandes, 2000. See also `http://www.mathweb.org/omdoc`.

[7] Paul Libbrecht and Dominik Jednoralski. Drag and drop of formulae from a browser. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[8] E. Melis, G. Goguadze, M. Homik, P. Libbrecht, C. Ullrich, and S. Winterstein. Semantic-aware components and services of ActiveMath. *British Journal of Educational Technology*, 37(3):405–423, May 2006.

[9] C. Ullrich. Tutorial planning: Adapting course generation to today's needs. In M. Grandbastien, editor, *Young Researcher Track Proceedings of 12th International Conference on Artificial Intelligence in Education*, pages 155–160, Amsterdam, The Netherlands, 2005.

[10] C. Ullrich, P. Libbrecht, S. Winterstein, and M. Mühlenbrock. A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. In Kinshuk, C. Looi, E. Sutinen, D. Sampson, I. Aedo, L. Uden, and E. Kähkönen, editors, *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004), Joensuu, Finland*, pages 21–25, 2004.

# LaTeX 2$_\varepsilon$, pict2e and complex numbers

Claudio Beccari
Politecnico di Torino
Turin, Italy
claudio dot beccari (at) polito dot it

**Abstract**

In 2003 the endless list of LaTeX packages was enriched by the package pict2e, intended to substitute for the dummy one that has accompanied every LaTeX distribution since 1994. This package implements everything as stated by Lamport in the second edition of his LaTeX manual (for LaTeX 2$_\varepsilon$). But if you explore the inner workings of the new pict2e, you discover the new package has some unexpected potential applications, especially if complex number arithmetic operations are included in it.

## 1 Introduction

The original package pict2e which accompanied the first release of LaTeX 2$_\varepsilon$ in 1994 was just a dummy package that would simply type out an info message that the *real* package was not yet available. Nevertheless, the LaTeX manual by Leslie Lamport [2] already described the features of this expected package; its primary function was to relieve the strong limitations of the picture environment, mainly due to the fact that graphic objects were realized by means of special fonts which necessarily contained a limited number of “graphic” glyphs.

Anyone who has used the original picture environment in LaTeX may have looked forward to the new pict2e package, so as to be able to draw the usual graphics available with other drawing facilities, even those that are an integral part of commercial and/or open source text processors.

The new pict2e [1] relieves all the limitations of the old picture environment, in particular: the small set of possible inclinations of segments and vectors; the limited number of radii for drawing circles; the rigidity in drawing ovals, whose corners suffered from the limited number of quarter circle arcs; the shortest length of segments and vectors limited to 10pt except for horizontal and vertical ones; the line thickness limited to two values due to the very limited number of special picture fonts; only second order Bézier curves which were made up of small dots partially superimposed on one another.

The new pict2e resorts to the output driver facilities, in the sense that it is dvips or pdf(la)tex[1] that takes care of drawing straight and curved lines, filled and unfilled contours, arrow tips, and the like, with all the facilities offered by the powerful PostScript language, even in its simplified form as used in PDF documents.

Figure 1 shows an example of a set of lines with slopes of 10°, 20°, ..., 80°. The following picture code reflects the usual syntax, with the only exception that line slopes are three digit integers, instead of the relatively prime one digit integers limited to a magnitude of 6 as in the “old” picture environment. The coefficients of the line slopes are simply obtained by rounding to the closest integers the sines and cosines of the angular slopes multiplied by 1000.

```
\unitlength=1mm
\begin{picture}(70,70)
...
\put(0,0){\line(985,174){68.95}}
\put(0,0){\line(940,342){65.80}}
\put(0,0){\line(866,500){60.62}}
...
\put(0,0){\line(342,940){23.94}}
\put(0,0){\line(174,985){12.18}}
\end{picture}
```

Depending on the output driver, pict2e inserts the necessary \special commands with the appropriate syntax, so that when running pdflatex the output PDF file contains the drawings that are directly visible with the PDF viewer. When running latex, the DVI file generally[2] must be processed with dvips to get a PostScript file where the drawings are directly visible with the PostScript viewer, and/or the PostScript file may be processed with ps2pdf to get a self-contained PDF file.

[1] Some other drivers are partially or totally supported.

[2] Several DVI file previewers can interpret the PostScript \specials, but this is not universally true.

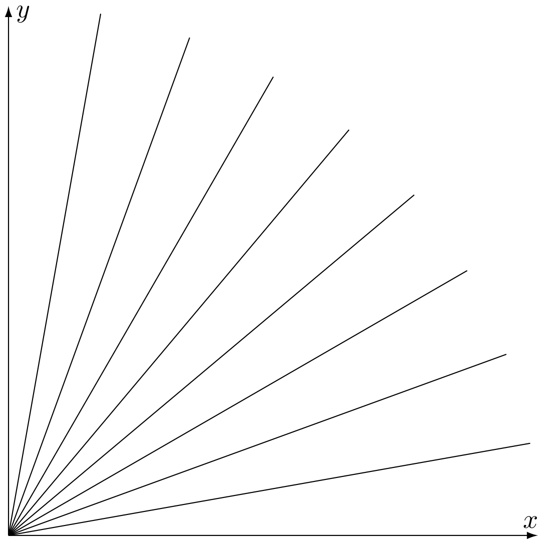


**Figure 1**: Line segments with angular slopes that are multiples of 10°, drawn with `pict2e`

When dealing with `pict2e`, I believe the `latex` + `dvips` + `ps2pdf` procedure is less interesting than the direct production of a PDF file by means of `pdflatex`, because in the former case the author may alternatively use the well-known and more powerful `PSTricks` [3].

I would like to encourage any LaTeX users who may be unaware of the availability of `pict2e` to download the package from CTAN, if necessary, and experiment with the new features. In particular, I would like to draw to the attention of Linux users that TeX distributions coming with some Linux systems are quite out-of-date with respect to CTAN. My own Linux-based distribution (2005/08/15), for example, contains only issue 14 of `latexnews.dvi` dated 2001/06/01, while my updated MiKTeX distribution contains issue 16 dated 2003/12/01. The new `pict2e` was announced in issue 15, also dated 2003/12/01. The current version (as of September 2006) of `pict2e` was updated 2004/08/06. The current version is available in current and future distributions of MiKTeX and TeX Live.

In the following sections I will give some examples of `pict2e` usage, and then describe some enhancements of the package, how to use some internal commands and how to build powerful new commands to draw arbitrary curves by means of third order Bézier curves. I will also need to describe some elementary properties of complex numbers and therefore how to implement complex number arithmetic by means of LaTeX and the underlying TeX macros and primitives.

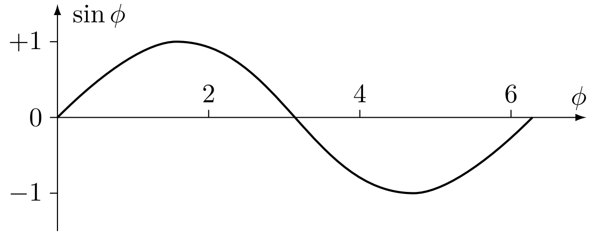


**Figure 2**: A sine wave

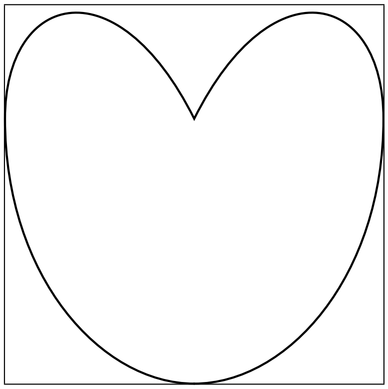


**Figure 3**: A curve containing a cusp

## 2 Examples

Our first example is an accurate sine wave, as in figure 2. This can be created as follows:

```
\Curve(0,0)<1,1>%          0 deg
      (1.570796,1)<1,0>%   90 deg
      (4.712389,-1)<1,0>% 270 deg
      (6.283185,0)<1,1>%  360 deg
```

where the parentheses contain the curve node coordinates and the angle brackets contain the direction coefficients of the curve tangents at each node.

A diagram with a cusp is shown in figure 3; the code used is the following:

```
\Curve(2.5,0)<1,0>(5,3.5)<0,1>%
      (2.5,3.5)<-.5,-1>[-.5,1]%
      (0,3.5)<0,-1>(2.5,0)<1,0>
```

Another example is given in figure 4 where the `\polyline` macro, described later, is used. The code for generating the heptagon and star vertices is the following:

```
\begin{picture}(5,5)(-2.5,-2.5)
\DividE 360pt by 7pt to\Seventh
\DirFromAngle\Seventh to\Dir
\CopyVect 0,2.5 to\Vone
\MultVect\Vone by\Dir to\Vtwo
\MultVect\Vtwo by\Dir to\Vthree
\MultVect\Vthree by\Dir to\Vfour
\MultVect\Vfour by\Dir to\Vfive
\MultVect\Vfive by\Dir to\Vsix
\MultVect\Vsix by\Dir to\Vseven
```

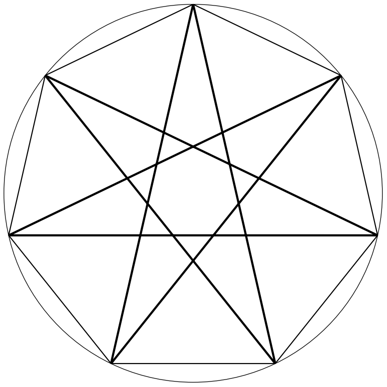


**Figure 4**: Heptagon and seven pointed star

```
\polyline(\Vone)(\Vtwo)(\Vthree)(\Vfour)%
         (\Vfive)(\Vsix)(\Vseven)(\Vone)
\thicklines
\polyline(\Vone)(\Vfour)(\Vseven)%
  (\Vthree)(\Vsix)(\Vtwo)(\Vfive)(\Vone)
\end{picture}
```

## 3 Extensions to the pict2e package

The pict2e package, according to the description in [2], retains the limitation that the slope parameters of the picture segments are represented with integer numbers. According to the authors, Rolf Niepraschk and Hubert Gäßlein, this limitation is due to the specific division routine used, as well as fulfilling the line and vector specifications specified by Lamport.

In a previous paper [4] I complained about the fact that even ε-TeX does not implement real floating number calculations and I invited developers to extend ε-TeX functionality in that direction.

Meanwhile, the LaTeX programmer must rely on "poor man" methods. The only TeX object that is representable with a fractional number in the input flow is the *scale factor* used for scaling lengths: when you type

```
\newlength{\dimA} \newlength{\dimB}
\setlength{\dimA}{33.25pt}
\setlength{\dimB}{1.44\dimA}
\showthe\dimB
```

you expect to see on the log file (and on the screen) that the dimension register \dimB contains the value of 47.88pt. Actually the log file will exhibit the value of 47.88008pt because of conversion, rounding and truncation errors during the whole process. Here is where the floating point arithmetic would be handy... in the future. But notice that 47.88 is the arithmetic product of the fractional measure in points of the register \dimA multiplied by the fractional number 1.44. Multiplication is then relatively an easy task provided we can convert back and forth fractional numbers and dimensions.

The trick is easy, and despite being classified as "dirty" in *The TeXbook* [5, page 375], it has been used by almost everyone needing to use this poor man approach to fractional number multiplication.

Division is trickier because it can produce overflows (like multiplication), the division by zero error, and it does not have any relation to scale factors, the only objects that TeX can use as multipliers.

Integer division is generally unusable and so the routine Gäßlein and Niepraschk used accepts integer dividend and divisor transformed into lengths, but yields a length whose measure in points is the required fractional quotient.

At the time pict2e became available I had been using a division routine for several years; it was part of a package of mine that was never published. The good point is that I had been using that package for years and that routine always worked reliably, although no controls were actually performed to avoid overflows or divisions by zero (they could be possibly be done before calling the routine). This routine actually divides two lengths and yields their ratio as a fractional signed decimal number. The code may be seen in figure 5.

TeX programming was used together with some plain TeX macros that are also available in the kernel of LaTeX. I chose to use the delimited argument facility of TeX, which is not available in LaTeX, because coding becomes more readable; the funny choice of the name with initial and final capitals has a long and insignificant history, but I did not want to change it here, for the sake of avoiding contradictions. For the same reason I did not translate \segno into, say, \Sign, but I suppose that its meaning is understandable by everybody. In practice \DividE implements a long division between the numerator and denominator lengths translated into scaled points (this is what TeX does when a counter is assigned a length value) stored into two numerical counters; at every iteration the remainder is multiplied by ten and the single digit new quotient \q is appended to the overall quotient \Q.

The only test I added to this last version of the routine is to assign a positive maximum TeX value to the quotient in case of division by zero, so that the best TeX approximation to infinity is used.

With this division routine at hand, we can extend the slope argument of segments to any reasonable fractional number; the \line of pict2e may be changed to the code in figure 6.

Some comments are in order because some of

```
\def\DividE#1by#2to#3{%
    \begingroup
    \dimendef\Numer=254\relax \dimendef\Denom=252\relax
    \countdef\Num 254\relax \countdef\Den 252\relax \countdef\I=250\relax
    \Numer #1\relax \Denom #2\relax
    \ifdim\Denom<\z@ \Denom -\Denom \Numer -\Numer\fi
    \def\segno{}\ifdim\Numer<\z@ \def\segno{-}\Numer -\Numer\fi
    \ifdim\Denom=\z@
      \ifdim\Numer>\z@\def\Q{16383.99999}\else\def\Q{-16383.99999}\fi
    \else
      \Num=\Numer \Den=\Denom \divide\Num\Den
      \edef\Q{\number\Num.}%
      \advance\Numer -\Q\Denom \I=6\relax
      \@whilenum \I>\z@ \do{\DividEDec\advance\I\m@ne}%
    \fi
    \xdef#3{\segno\Q}\endgroup
}%

\def\DividEDec{\Numer=10\Numer \Num=\Numer \divide\Num\Den
            \edef\q{\number\Num}\edef\Q{\Q\q}\advance\Numer -\q\Denom}%
```

**Figure 5**: Another division routine for fractional values

the code may seem redundant. The `\line` macro behaves as in the original `pict2e`, except that the "only" argument `#1` actually has the usual format of two fractional or integer numbers separated by a comma; this is the form I will give to the representation of complex numbers; the `\line` macro does not actually need this machinery, but since the necessary macros are already there, why not?

The `\DirOfVect` macro takes the two direction comma-separated coefficients passed in argument `#1`, interprets them as the horizontal and vertical components of a vector and determines the directing cosines, or, if you prefer, normalizes these two vector components to the length of the vector itself, so that they are both fractional numbers whose magnitude does not exceed unity. This is good for the following operations and division calculations. The rest of the macro is very similar to the original one. But it may be observed that the above normalization does not depend on the integer or fractional nature of the directional coefficients; it even neglects the fact that their magnitude may be larger than 1000, which is the last constraint remaining in the original `pict2e` `\line` macro.

Of course these coefficients should not be too large, even though the length of the vector computation implies some powers of two and a square root; computations are made in such a way as to extract from the root the largest of the two components, so that a number not exceeding unity gets squared and the radicand never exceeds 2. The `\ModOfVect` macro actually executes this square root and I have never observed any deficiency in its calculations.

This extension suggests another one; since the direction coefficients may be of any reasonable magnitude, why should we maintain the `picture` syntax for defining lines, where along with the direction coefficients it is necessary to specify the horizontal projection of the segment? Why not define the line with its absolute horizontal and vertical components? Everything would be much cleaner and the execution time would be much shorter. Thus, I defined an alternative line description, as follows:

```
\def\Line(#1,#2){%
\pIIe@moveto\z@\z@
\pIIe@lineto{#1\unitlength}%
            {#2\unitlength}%
\pIIe@strokeGraph}%
```

where the arguments passed to the macro represent the actual components of the segment, and no length is specified. With `\put` you put the segment origin as usual and the `\Line` macro does the rest. The "`moveto`", "`lineto`" and "`stroke`" keywords are those used in PostScript and in many descriptive graphic languages; these are some of the new keywords introduced by `pict2e` and they may induce a small revolution in considering graphics with LaTeX.

For example it is possible to define a macro for tracing a polygonal line joining an arbitrary number

```
\def\line(#1)#2{\begingroup
  \@linelen #2\unitlength
  \ifdim\@linelen<\z@\@badlinearg\else
    \expandafter\DirOfVect#1to\Dir@line
    \GetCoord(\Dir@line)\d@mX\d@mY
    \ifdim\d@mX\p@=\z@\else
       \ifdim\d@mX\p@<\z@ \@tdB=-\p@\else\@tdB=\p@\fi
       \DividE\@tdB by\d@mX\p@ to\sc@lelen \@linelen=\sc@lelen\@linelen
    \fi
    \pIIe@moveto\z@\z@
    \pIIe@lineto{\d@mX\@linelen}{\d@mY\@linelen}%
    \pIIe@strokeGraph
  \fi
\endgroup\ignorespaces}%

\def\GetCoord(#1)#2#3{%
  \expandafter\SplitNod@\expandafter(#1)#2#3\ignorespaces}

\def\SplitNod@(#1,#2)#3#4{\edef#3{#1}\edef#4{#2}}%

\def\DirOfVect#1to#2{\GetCoord(#1)\t@X\t@Y
  \ModOfVect#1to\@tempa \DividE\t@X\p@ by\@tempdimc to\t@X
  \DividE\t@Y\p@ by\@tempdimc to\t@Y
  \MakeVectorFrom\t@X\t@Y to#2\ignorespaces}%

\def\ModOfVect#1to#2{\GetCoord(#1)\t@X\t@Y
  \@tempdima=\t@X\p@ \ifdim\@tempdima<\z@ \@tempdima=-\@tempdima\fi
  \@tempdimb=\t@Y\p@ \ifdim\@tempdimb<\z@ \@tempdimb=-\@tempdimb\fi
  \ifdim\@tempdima>\@tempdimb
     \DividE\@tempdimb by\@tempdima to\@T
     \@tempdimc=\@tempdima
  \else
     \DividE\@tempdima by\@tempdimb to\@T
     \@tempdimc=\@tempdimb
  \fi \ifdim\@T\p@>\z@
     \@tempdima=\@T\p@ \@tempdima=\@T\@tempdima
     \advance\@tempdima\p@
     \@tempdimb=\p@
     \@tempcnta=5\relax
     \@whilenum\@tempcnta>\z@\do{\DividE\@tempdima by\@tempdimb to\@T
     \advance\@tempdimb \@T\p@ \@tempdimb=.5\@tempdimb
     \advance\@tempcnta\m@ne}%
     \@tempdimc=\@T\@tempdimc
  \fi
  \Numero#2\@tempdimc
\ignorespaces}%

\def\MakeVectorFrom#1#2to#3{\edef#3{#1,#2}\ignorespaces}%
```

**Figure 6**: Redefinition of the `\line` macro using the new division routine

of nodes as such, as in \polyline here:[3]

```
\def\polyline(#1){\beveljoin
  \GetCoord(#1)\d@mX\d@mY
  \pIIe@moveto{\d@mX\unitlength}%
              {\d@mY\unitlength}%
  \p@lyline}%

\def\p@@lyline(#1){%
  \GetCoord(#1)\d@mX\d@mY
  \pIIe@lineto{\d@mX\unitlength}%
            {\d@mY\unitlength}%
  \p@lyline}%

\let \lp@r( \let\rp@r)

\def\p@lyline{%
  \@ifnextchar\lp@r{\p@@lyline}%
      {\pIIe@strokeGraph\ignorespaces}%
}%
```

This simple macro \polyline would be rather difficult to realize without the moveto, lineto and stroke keywords.

A final small improvement consists in setting the shape of the line terminators; by default they are square caps but when tracing thick lines that meet at the same point it is better to set them round. See figure 7 for a comparison. The following code gives access to these settings:

```
\ifcase\pIIe@mode\relax
\or %PostScript
 \def\roundcap{\special{ps:: 1 setlinecap}}%
 \def\squarecap{\special{ps:: 0 setlinecap}}%
 \def\roundjoin{\special{ps:: 1 setlinejoin}}%
 \def\beveljoin{\special{ps:: 2 setlinejoin}}%
\or %pdf
 \def\roundcap{\pdfliteral{1 J}}%
 \def\squarecap{\pdfliteral{0 J}}%
 \def\roundjoin{\pdfliteral{1 j}}%
 \def\beveljoin{\pdfliteral{2 j}}%
\fi
```

I prefer to have the round cap version as the default setting, but this is a question of personal taste. Apparently these settings, set up by means of the special programming language of the destination file, are global ones so it is necessary to countermand them once the default has to be restored; it is not possible to rely on groups in the usual TeX way. I also have the impression that at each closing of a picture environment any setting is lost; I do not know the PostScript language well enough to understand if some internal pict2e command executes this reset, but after all it is no trouble to reset the preferred settings at the beginning of each picture.

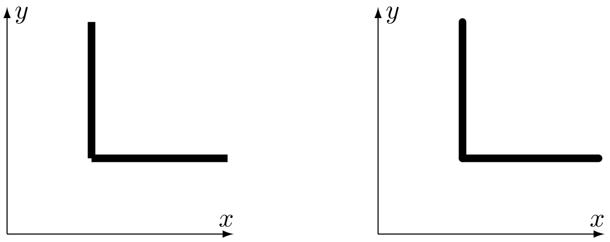


**Figure 7**: Square and round caps

It happens that the line terminator choice does not work with the original pict2e \line definition when the drawn segments are purely horizontal or vertical, while it does work with my redefinition, as can be seen in figure 7. After all the original pict2e definition of \line mimics the original "LaTeX 2.09" one, where it was important to avoid drawing lines by means of the special graphic fonts when horizontal and vertical lines could be more easily and efficiently drawn with the low level DVI commands TeX uses for vertical and horizontal rules. When lines are drawn with the device driver facilities it is no longer necessary to make horizontal and vertical lines special cases.

I should remark that several other graphic packages are available; among them the curves package by Ian Maclaine [6] certainly is the first one that might benefit from these new facilities introduced by pict2e. There is also the package bundle pgf by Till Tantau [7]; PGF stands for "portable graphic format" and its intention is to provide LaTeX with a portable set of macros providing nearly as much as PSTricks, even when running pdflatex. The latter program is at the base of the excellent presentation document class beamer and is certainly worth using because of its fine properties. In the event, I did not extend pgf because I found some difficulties in writing some macros, such as to relocate output to specified coordinates. Moreover I believe that pict2e, although much simpler then pgf, is part of LaTeX, not a major extension as pgf is.[4]

## 4 Complex numbers

As has partially been seen, drawing implies treating *directions*; in particular, it is necessary to manipulate vectors and their directions. METAFONT [8], the program for drawing fonts written by Knuth himself, treats all these objects with complex numbers. Knuth hardly ever cites complex numbers in

[3] Of course with delimited arguments it is not possible to use the LaTeX macro definition commands.

[4] With pict2e I had no difficulties rewriting the macros of a package of mine for drawing electronic circuits; I was not able to do the same with pgf; of course the one to blame is just myself.

*The METAFONTbook*, but all the inner and outer workings are done by means of *pairs* that are nothing else but complex numbers. The manipulation of directions and angles is always done in an alternating change from Cartesian to polar representation of complex numbers; here and there some of the operations available on pairs are explicit complex number operations.

Most people don't know or don't like complex numbers; perhaps this is due to the fact that they contain imaginary quantities, something far away from the everyday reality of numbers.

Mathematicians, on their side, usually do little to ease students' learning of complex numbers, and with their love for abstraction and generalization they sometimes miss conveying the message that these entities are nothing more than "scale-rotate" operators: they simply scale an object up and down and rotate it around a pivoting point. Almost everybody is familiar with these operators, from using one of the many interactive drawing programs, even relatively simple ones.

To see this, take a vector $\vec{v}$ drawn from the origin of a Cartesian plane defined with axes $x$ and $y$; if you project the above vector on the $x$ axis you get the horizontal component $\vec{v}_x$, while if you project it on the $y$ axis you get the vertical component $\vec{v}_y$. If you define two unit vectors, $\vec{u}_x$ parallel to the $x$ axis and pointing to increasing $x$ values, and similarly $\vec{u}_y$ for the $y$ axis, you can separate in every component the information of its magnitude from that of its direction and you can write

$$\vec{v} = \vec{v}_x + \vec{v}_y = v_x\vec{u}_x + v_y\vec{u}_y$$

Now let us emphasize the link the vector $\vec{v}$ has with the unit direction along the $x$ axis by writing

$$\vec{v} = [v_x + (\vec{u}_y/\vec{u}_x)v_y]\vec{u}_x$$

so that we may interpret the contents of the square brackets as the operator that acts on the unit $x$ vector, by scaling it according to the magnitude of $\vec{v}$, and by rotating it by a certain angle, the angle of $\vec{v}$ with respect to the $x$ axis. The contents of the square brackets have the same characteristics as those we anticipated for complex numbers.

The ratio $\vec{u}_y/\vec{u}_x$ is generally given the name of 'i' by the mathematicians and 'j' by most technologists.[5] Its application has the geometric meaning of changing the unit vector $\vec{u}_x$ to the unit vector $\vec{u}_y$, i.e. rotating the unit vector $\vec{u}_x$ 90° counterclockwise.

If we apply twice in a row the 90° rotation operator 'i' to the unit vector $\vec{u}_x$, producing a total rotation of 180°, we get the renowned expression

$$\mathrm{i}\,\vec{u}_y = \mathrm{i}(\mathrm{i}\,\vec{u}_x) = \mathrm{i}^2\,\vec{u}_x = -1\vec{u}_x$$

that is

$$\mathrm{i}^2 = -1 \qquad \text{or} \qquad \mathrm{i} = \sqrt{-1}$$

which induced XVI century mathematicians to call 'i' the *imaginary unit*.

If we further process the above results, we get

$$v_x + (\vec{u}_y/\vec{u}_x)v_y = v_x + \mathrm{i}\,v_y = |\vec{v}|\left(\frac{v_x}{|\vec{v}|} + \mathrm{i}\,\frac{v_y}{|\vec{v}|}\right)$$

where

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2}$$

We recognize that if the original vector $\vec{v}$ is inclined by an angle $\theta$ counterclockwise with respect to the $x$ axis, then the two above fractions represent the cosine and sine of such an angle

$$\begin{aligned}\frac{v_x}{|\vec{v}|} &= \cos\theta\\ \frac{v_y}{|\vec{v}|} &= \sin\theta\end{aligned}$$

The scaling factor of the operator acting on the unit $x$ vector is $|\vec{v}|$ and the direction of the $x$ unit vector is changed by the angle $\theta$ counterclockwise. The operator is actually a scale-rotate operator, i.e. a complex number.

If we apply two scale-rotate operators in a row to the unit $x$ vector, we make the following observations:

1. the two scaling factors behave as two multipliers and are commutative;
2. the two rotation angles add up and are commutative;
3. the total effect produced by the two operators is therefore equivalent to that of a single operator whose magnitude is the product of the two magnitudes and whose angle is the sum of the two angles.

In order to represent such effects with the operation of multiplication it is advisable to use magnitudes as regular factors, and to use angles as exponents of a suitable base; the mathematicians tell us that a scale-rotate operator of magnitude $a$ and of angle $\theta$ can be represented as

$$a(\cos\theta + \mathrm{i}\,\sin\theta) = a\,\mathrm{e}^{\mathrm{i}\,\theta}$$

which is called Euler's formula. There are many serious reasons for choosing 'e' as the base and for representing the exponent as an imaginary quantity, but we are not concerned here with them; we simply note that given two scale-rotate operators $a\exp(\mathrm{i}\,\theta)$ and $b\exp(\mathrm{i}\,\phi)$ their total effect is $(ab)\exp[\mathrm{i}(\theta+\phi)]$.

This observation together with Euler's formula lets us understand the meaning of division by a complex number, i.e. a scale-rotate operator; in fact, the

[5] Notice that this mathematical *operator* is not a variable and therefore according to international standards must be written with an upright font.

division is nothing but the inverse operator of a multiplication, and this can be expressed as

$$\left[a\,\mathrm{e}^{\mathrm{i}\,\theta}\right]^{-1}=\frac{1}{a}\,\mathrm{e}^{-\,\mathrm{i}\,\theta}$$

where we observe that the scaling factor is simply the reciprocal of the multiplicative one, while the rotation term is just in the opposite direction relative to the multiplicative one.

The scale-rotate interpretation of complex numbers also lets us understand very easily the meaning of addition and subtraction of such entities — which end up being the same as addition and subtraction of vectors. Moreover the vector notation becomes redundant, since the scale-rotate operators always act on the unit $x$ vector, which can thus be taken for granted and omitted from the complex number expressions. These expressions therefore maintain the meaning of vector operations *and* of complex number relationships.

For our present purposes, let us emphasize that $\exp(\mathrm{i}\,\theta)$ has unit magnitude, and therefore contains only the information on the direction. Furthermore, $\exp(-\,\mathrm{i}\,\theta)$ represents a rotation in the opposite direction; if we have the means of multiplying by such factors we can change the direction of any vector the way we like, either counterclockwise or clockwise.

For TeX arithmetic it is better to use simple multiplications without exponentials, but Euler's formula lets us change back ad forth from the exponential form to the Cartesian one; the exponential form gives us an easy interpretation of the rotating effects while the Cartesian form gives us an easy mechanism for executing the complex multiplication and therefore the required rotation.

## 5 Complex number TeX macros

I am not going to include here the code for every complex number operation [9]; let me just list the macro names of the operations I wanted to realize, with some explanations to clarify detail.

Notice that I decided to maintain most if not all fractional numbers in control sequences. I also use control sequences to pass complex values to the macros, so that in order to operate on the complex number parts a macro (`\GetCoord`) is needed to separate them, and another (`\MakeVectorFrom`) to reassemble them.

Most macros have delimited arguments, with the main command is followed by the sequence of arguments separated by keywords; rarely, arguments must be enclosed in the traditional curly braces, as normally necessary in LaTeX. For example in order to extract the magnitude (modulus) and the direction from a given vector the macro name is `\ModAndDirOfVect` but the various arguments are separated by the words `to` and `and` so that a typical call might be

```
\ModAndDirOfVect\VectorA to\ModA and\DirA
```

In this context the word "vector" is synonymous with complex number or scale-rotate operator; the word "direction" refers to a complex number with unit magnitude so that the scaling factor is unity.

In the following list of macros the parameters `#1`, `#2`,… are the arguments passed to the various macros. The macro names are assumed to be self explanatory.

```
\SinOf#1to#2
\CosOf#1to#2
\TanOf#1to#2
\MakeVectorFrom#1#2to#3
\CopyVect#1to#2
\ModOfVect#1to#2
\DirOfVect#1to#2
\ModAndDirOfVect#1to#2and#3
\GetCoord(#1)#2#3
\DistanceAndDirOfVect#1minus#2to#3and#4
\XpartOfVect#1to#2
\YpartOfVect#1to#2
\DirFromAngle#1to#2
\ScaleVect#1by#2to#3
\ConjVect#1to#2
\AddVect#1and#2to#3
\SubVect#1from#2to#3
\MultVect#1by#2to#3
\MultVect#1by*#2to#3
\DivVect#1by#2to#3
```

The list ends with the usual four arithmetic operations performed on any mathematical entity; the variant of the multiplication that contains an asterisk performs the multiplication of the first operand by the *complex conjugate* of the second operand; the complex conjugate of a complex number is just the scale-rotate operator where the rotation direction has been reversed.

The above list starts with the usual trigonometric functions. Actually, I stated earlier that arithmetic in TeX should be done with numbers and directions; angles, that are so expressive in Euler's formula, should be avoided. Nevertheless, at some point it's necessary to convert angles to their sines and cosines, but switching back and forth from the Euler representation to the Cartesian one implies the computation of both direct and inverse trigonometric functions. TeX can do both operations (with acceptable approximations) but it slows down quite a bit with frequent transformations. I implemented the computation of the direct trigonometric functions of angles *in degrees*, not in radians, by means of the continued fraction expansion of the half angle tangent and the parametric formulas:

$$\begin{aligned}\sin\theta &= \frac{2\tan x}{1+\tan^2 x}\\ \cos\theta &= \frac{1-\tan^2 x}{1+\tan^2 x}\\ \tan\theta &= \frac{2\tan x}{1-\tan^2 x}\end{aligned}$$

where

$$\tan x = \cfrac{1}{\cfrac{1}{x} - \cfrac{1}{\cfrac{3}{x} - \cfrac{1}{\cfrac{5}{x} - \cfrac{1}{\cfrac{7}{x} - \cdots}}}}$$

and $x = \theta/114.591559$ is the half angle in degrees converted to radians.

This iterative formula for the tangent is quite fast and its precision is remarkable if we consider the modest performance of TEX calculations with fractional numbers. I decided to stop the continued fraction with the term containing the coefficient '11'; probably it is a little too much for TEX capabilities but I prefer to perform one extra cycle than to miss the target.

Unfortunately I could not find similar fast algorithms for the inverse trigonometric functions; I decided to avoid using such inverse functions. METAFONT implements both algorithms, but METAFONT is not TEX: the former was designed to perform fractional number calculations (although represented in fixed radix notation) while the latter was designed for efficiently typesetting text, with calculations reduced to integer operations with some simple tricks to cope with the necessity of fractional "factors".

The macro `\DirFromAngle` is the only one that uses trigonometric functions; further on, just direction vectors are used.

## 6 Circular arcs

`pict2e` implements only the drawing commands specified by Lamport in [2]; it can draw full circles or quarter circles but it cannot draw arcs of any other specified angle amplitude. Or better: it cannot draw them because of the lack of user commands, but it has all the potentialities.

Suppose we want to draw an arc by specifying its center, its starting point and its angle amplitude. The center and the starting point may be absolute coordinates in the `picture` environment space or may be relative to the position specified with a `\put`.

With `pict2e` we can resort to third order Bézier curves as it is done in METAFONT; if the third order curve is not misused, it can approximate up to a half circle with remarkable precision. The problem is to find the arc end point and the correct control points of the Bézier curve.

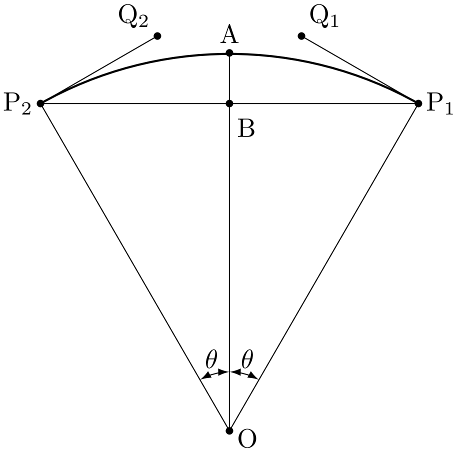


**Figure 8**: Circular arc elements

With reference to figure 8 it is a simple exercise, given the center O, the starting point $\mathrm{P}_1$, and the arc angle $2\theta$, to determine the coordinates of the end arc point $\mathrm{P}_2$ and the two control points $\mathrm{Q}_1$ and $\mathrm{Q}_2$.

For the end point $\mathrm{P}_2$ it suffices to take the vector $\overrightarrow{\mathrm{P}_1 - \mathrm{O}}$ and rotate it about the center point by the given angle $2\theta$; this operation is simplified by the complex number arithmetic described above.

A little trickier is the determination of the control points. It is evident that they lie on the segments perpendicular to the vectors $\overrightarrow{\mathrm{P}_1 - \mathrm{O}}$ and $\overrightarrow{\mathrm{P}_2 - \mathrm{O}}$, but how long are the vectors $\overrightarrow{\mathrm{Q}_1 - \mathrm{P}_1}$ and $\overrightarrow{\mathrm{Q}_2 - \mathrm{P}_2}$? To answer, it is necessary to know the cubic Bézier equation:

$$\mathrm{P} = \mathrm{P}_1(1-t)^3 + 3\mathrm{Q}_1 t(1-t)^2 + 3\mathrm{Q}_2 t^2(1-t) + \mathrm{P}_2 t^3$$

that can be found (using other symbols) in *The METAFONTbook*.

P is the generic point on the curve; the start and end points and the control points form the coefficients of the equation; $t$ is a parameter that runs from 0 to 1 while P moves from $\mathrm{P}_1$ to $\mathrm{P}_2$. The above equation in reality represents the pair of equations obtained when the point coordinates are substituted; therefore it represents the pair of parametric equations that describe the curve in the usual $xy$ Cartesian plane.

If we move the origin of the coordinates to point B of figure 8 and exploit the obvious symmetry, the similar triangles, the Bézier equation, and the fact that point A must be distant from O just as $\mathrm{P}_1$ and $\mathrm{P}_2$, it turns out that the length $K$ of the required vectors is

$$K = \frac{4}{3}\,\frac{1-\cos\theta}{\sin\theta}\,R$$

where $R$ is the arc radius, the length of the vector $\overrightarrow{\mathrm{P}_1 - \mathrm{O}}$. Again, with the complex number operations we have at our disposal, it is straightforward to exploit the information we have to trace the cubic Bézier curve from $\mathrm{P}_1$ to $\mathrm{P}_2$; the result is indistinguishable from a true circle when the total arc angle does not exceed 90° and is not noticeable with the naked eye when the total arc angle does not exceed 180°.

Therefore a good `\Arc` macro should check the amount of the total arc angle and possibly split the total arc into sub-arcs none of which exceeds a half circle, or, even better, a quarter circle. This is why in actual computations it is much better to measure angles in sexagesimal degrees than in radians; with radians the reduction of angles by amounts corresponding to quarter or half circles intrinsically requires approximation due to the irrational nature of $\pi$; TeX introduces its own approximation errors, so let us not contribute with further ones.

## 7 General curves

The abovementioned package `curves` [6] by Ian Maclaine offers the user the possibility of tracing arbitrary curves by stating just the curve nodes; METAFONT is entirely built on this possibility, although it uses much finer mathematics and it offers the user the opportunity to optionally specify node tangents and arc tensions.

From the user's point of view these differences are great by themselves, but there is another important difference: `curves` uses quadratic Bézier curves, while METAFONT uses cubic ones. This produces dramatic differences when the curve nodes imply the presence of inflection points. In this case the algorithm devised by Maclaine more often than not produces anomalous loops; such loops are very rare with cubic Bézier curves — it is necessary to work hard just to find examples of such loops.

Of course Maclaine had to sacrifice some graphical functionality in favor of simpler mathematics, which, as we know, is not TeX's best feature.

I tried to devise a chain of macros that trace one cubic Bézier arc at a time, and pass one another the end point tangent directions. These macros are

```
\StartCurveAt#1WithDir#2
\CurveTo#1WithDir#2
\CurveFinish
```

where the first argument is a point coordinate pair (a complex number) and the second argument is a direction (a complex number with unit magnitude). The first macro initializes the process and memorizes the first point direction; the second macro gives the destination program the necessary information on the arc nodes and control points, and the third macro eventually strokes the curve with the syntax of the destination program.

The first macro basically uses the `moveto` keyword, the second macro `curveto`, and the final macro `stroke`; we have already partially seen these keywords while discussing lines and polylines. The first and second macros also normalize the directions given, so the end user does not need to make preliminary calculations in order to normalize the direction magnitude. They also memorize the specified and normalized direction for the benefit of the next `\CurveTo` call.

The `\CurveTo` macro is the one that has to do the main work in determining the position of the control points. Obviously it must start by checking the trivial situations where the directions form zero or 180° angles with the arc chord; it must also distinguish the situations where the tangents form 90° angles with the chord. It must behave correctly even if the end nodes and the directions imply an inflection point. But in most cases it has to deal with normal situations where the control point directions relative to the respective end points are given but the distances of the control points from the nodes must be determined.

There is a great margin for arbitrary decisions. I decided to divide the chord in two parts that are more or less proportional to the projection of the directions on the chord and to determine the distance $K$ of each control point from its neighboring node with the same formula as for circular arcs. The chord fraction is treated as half the chord of a circular arc and the corresponding radius is determined so as to use the mentioned formula. This choice is totally arbitrary but, as it is easily understandable, it is a reasonable one.

This done, the usual complex number arithmetic can be used to locate the position of the control point and to give the internal command for instructing the destination program how to draw the desired curve. In figure 2 there is a simple example of a sine curve that has been drawn with three arcs: from the origin to the maximum, from this point to the minimum, and lastly from the minimum to the end of the cycle.

Actually the three above macros are the ingredients of the general macro `\Curve` that operates on an arbitrary number of couples of nodes and directions:

```
\Curve(⟨P_0⟩)<⟨dir_0⟩>(⟨P_1⟩)<⟨dir_1⟩>...
        (⟨P_n⟩)<⟨dir_n⟩>
```

whose code is the following:

```
\def\Curve(#1)<#2>{%
    \StartCurveAt#1WithDir{#2}%
    \@ifnextchar\lp@r\@Curve{%
    \PackageWarning{curve2e}{%
    Curve specifications must contain at least
    two nodes!\Messagebreak
    Please, control your Curve
    specifications\MessageBreak}}}

\def\@Curve(#1)<#2>{%
    \CurveTo#1WithDir{#2}%
    \@ifnextchar\lp@r\@Curve{%
    \@ifnextchar[\@ChangeDir\CurveFinish}}

\def\@ChangeDir[#1]{\ChangeDir<#1>\@Curve}
```

For each node it is necessary to specify the direction of the tangent to that node, but these tangent direction coefficients need not be normalized. They are normally given within angle brackets. If there is a cusp, the tangent changes abruptly so that a new direction must be specified before continuing to draw the curve; this "optional" change in direction is indicated with a direction enclosed in square brackets; see the code implementing the heart shape in figure 3.

In the light of that example it is not a burden to specify all the directions at each node, although a simpler syntax such as that used in METAFONT would be desirable.

## 8 Conclusion

I wanted to illustrate the use of fractional number TeX arithmetic applied to complex numbers. These are formidable tools for graphics applications and the necessary macros are actually within the range of every TeXnician; there is no need to be a guru.

I hope the ideas I gave here may be exploited better than I can do for extending the existing graphic packages so as to get the best from the pict2e package. This package in particular may benefit from some simple extensions, or may incorporate the user macros for drawing circular arcs and arbitrary curves, possibly even with the filling capabilities that are being offered by other programs.

There might even be some expert programmer who feels challenged to write a user graphical interface that exploits the suggested extensions.

The pict2e package may still have minor glitches,[6] but even right now it opens many possibilities that were unthinkable when the standard LaTeX 2.09 picture environment provided the only native graphics for LaTeXers. They eventually had to give up and move to other dedicated programs; these are fine, but they do not necessarily produce completely compatible code and generally, with some remarkable exceptions (such as pgf), require special treatment in order to insert the same fonts used in the main text.

I hope the features described here will be included in future releases of the mentioned packages. Until that time, I have made a package curve2e available from CTAN [9] for anyone who wishes to make use of them.

### References

[1] Gäßlein H. and Niepraschk R., *The pict2e package*, PDF document attached to the "new" pict2e bundle; the bundle may be downloaded from CTAN.

[2] Lamport L., *LaTeX: A Document Preparation System*. Addison Wesley Publishing Co., Reading, Massachusetts, 1994.

[3] van Zandt T., *PSTricks*, CTAN. See also http://www.tug.org/PSTricks for further documentation and examples.

[4] Beccari C., *Floating point numbers and Metafont, MetaPost, TeX, and PostScript Type 1 fonts*, *TUGboat* 23:3/4, 2002, pp. 261-269.

[5] Knuth D.E., *Computers & Typesetting volume A: The TeXbook*, Addison Wesley Publishing Co., Reading, Massachusetts, Millennium Edition.

[6] Maclaine I., *curves* and *curvesls*, CTAN.

[7] Tantau T., *User's Guide to the PGF Package*, included in the pgf bundle downloadable from CTAN.

[8] Knuth D.E., *Computers & Typesetting volume C: The METAFONTbook*, Addison Wesley Publishing Co., Reading, Massachusetts, Millennium Edition.

[9] Beccari C., *curve2e*, CTAN.

[6] With the version I have at hand, pict2e traces vectors a little bit thicker than segments, although the line thickness is maintained constant; with my redefinition of \vector this glitch appears to be corrected.

# Page design in LaTeX3

Morten Høgholm
LaTeX3 Project
morten dot hoegholm (at) latex dash project dot org

**Abstract**

Choosing a page layout in LaTeX is easy for the user with the help of packages like geometry and typearea. However, users face various problems when wishing to change the layout parameters mid-document — something which happens quite often: title pages, rotated pages (with rotated header and footer), special pages or spreads including large images, and other situations where manual fiddling is a difficult and error prone process.

This article describes an interface for defining, storing, and retrieving complete page layouts. It will take a look under the hood to see how the data structures and programming constructs provided by the LaTeX3 kernel ease the programming task.

## 1 Introduction

Setting and understanding the page layout parameters in standard LaTeX is not the easiest of tasks. There is no interface at all so it must be done by setting all parameters manually and then hope you got them right. The packages geometry and typearea both provide an interface for changing the parameters for the entire document but there are situations where one may wish to use a different layout for just a few pages of the document:

- Title and back pages, where content is often centered on the physical page.
- Rotated pages, where some users want to also rotate header and footer.
- Page spreads containing material crossing page borders.

Additionally, the ubiquitousness of PDF documents on the Internet has opened up a new window of opportunity: changing page size mid-document.

## 2 The current solutions

Before trying to figure out a way to deal with page layout in LaTeX3, it is probably a good idea to take a closer look at the existing solutions within the LaTeX 2$_\varepsilon$ framework.

### 2.1 The base distribution

The LaTeX kernel doesn't really have much of an interface when it comes to modifying the layout. The only way to change anything is to set the dimensions by hand by means of commands such as

```
\setlength\paperheight{29.7cm}
\setlength\paperwidth{21cm}
```

Using the "raw" LaTeX commands like this is not an ideal interface for a designer. The layout package from the tools bundle alleviates this a bit by drawing the layout for you so that you can check if it looks as intended. This way you may also discover conflicting settings, as the kernel does not check this itself.

Should one wish to change some parameters temporarily mid-document their values must either be stored or the changes done locally. However, the parameters are global (in the sense that they are set at the top level in the document preamble) and it is bad practice to do local changes to global parameters. In short: there are no technical hindrances for the adventurous user to change the parameters at will but it is at best a tedious and error prone procedure.

Another problem is that recto and verso pages are exactly the same except for `\evensidemargin` and `\oddsidemargin`. As a consequence, certain types of designs become much harder to do in LaTeX than you'd expect. For example, defining a page layout where every odd page is six lines shorter (because, say, this is reserved for supplementary text) is slightly difficult because one has to manually change the text height from page to page.

### 2.2 The geometry and typearea packages

The most popular package for changing the page layout is the geometry package [4]. It provides an easier interface, one using a $\langle key \rangle = \langle value \rangle$ syntax whereby the designer/user is shielded from the direct form of the LaTeX parameters. The syntax is almost self-explanatory:

```
\usepackage{geometry}
\geometry{
  paper = a4paper ,
  textwidth = 6in ,
  lines = 42
}
```

Not surprisingly this makes the textwidth 6 in and puts 42 lines of text on A4 paper. geometry does not try to enforce typographic rules of thumb on the user except for choosing margin ratios suitable for printed books. Other than that, the user is responsible for everything.

A different approach is taken by the typearea package from the koma-script bundle [3]. Based on the document font, it tries to produce a textwidth of about 60–70 characters and then defines the layout in accordance with good typographic practice, such as a 1:2 relationship between top and bottom margin.

For a more in-depth discussion of both geometry and typearea, see [2]. Common to both packages is that they work only for the entire document: All settings must be done in the preamble and thus we have not yet solved one of our initial problems: changing the parameters mid-document.

## 3 A new solution

In order to come up with a solution to the problems, let's take a step back and look at what sort of areas may appear on a page.

### 3.1 The parts of a page

Instead of defining recto and verso pages we will define one standard page description so that all different page types have the same set of parameters, each parameter corresponding to a letter as shown in Figure 1. The indices $w$, $h$, and $s$ denote width, height, and separation respectively. The central thing on the page is the textblock $T$ itself. The textblock is surrounded on all sides by header, footer, left area and right area. The left and right areas are where margin notes may be positioned or as seen in some designs where the designer uses the right margin to place captions, section headings etc. and outside these areas are the actual margins. Finally the final paper size is often different from the stock size so we must take this into account too. We can notice several relationships:

$$
\begin{aligned}
S_w &= t_L + t_R + p_w\\
S_h &= t_T + t_B + p_h\\
p_w &= m_L + L_w + L_s + T_w + R_w + R_s + m_R\\
p_h &= m_T + H_h + H_s + T_h + F_s + F_h + m_B
\end{aligned}
$$

These relationships are similar to what can be found in geometry and ease the task of autocompletion and error checking.

The output routine will have to know which values to use and the easiest way is probably to use a set of global parameters with names such as

```
\g_page_std_textheight_dim
\g_page_std_textwidth_dim
```

etc. For those unfamiliar with the LaTeX3 naming conventions, these are global dimension registers belonging to the page module and going by the name std_⟨*parameter*⟩. In a similar manner we will make each page type have an identical set of parameters, except these go by names like

```
\l_page_⟨type⟩_textheight_dim
\l_page_⟨type⟩_textwidth_dim
```

etc., i.e., they are local parameters and then the shipout routine can handle setting the global parameters to the value of the local ones. Defining a complete set of parameters for each page type uses quite a number of dimension registers but since LaTeX3 uses ε-TeX (or better) we are not tied to solutions working within the tightly confined space of the Knuthian TeX engine.

Since we have decided that all page types share the same parameters, we must also solve the problem of storing and retrieving parameter values since they must change between, for instance, recto and verso pages. For this task we turn our attention to some of the tools provided by the LaTeX3 kernel.

### 3.2 Tools

TeX comes with only a few kinds of registers and anything dealing with lists must be done in macros. The tools for list processing in the LaTeX 2ε kernel are restricted to token lists and comma separated lists, and even then there are only mapping functions. The LaTeX3 kernel alleviates this by both extending the list of data structures with sequences and property lists as well as providing a complete set of tools for each data type: mapping, push/pop operations, etc. For the problem at hand we will use property lists.

#### 3.2.1 Property lists

A property list is a list structure consisting of a series of keys, each with its own information field, of the form

```
\⟨key 1⟩{⟨info 1⟩}
\⟨key 2⟩{⟨info 2⟩} ...
```

For example, a small test file for the cross referencing module of LaTeX3 contains a property list \g_xref_mylabel_plist which expands to

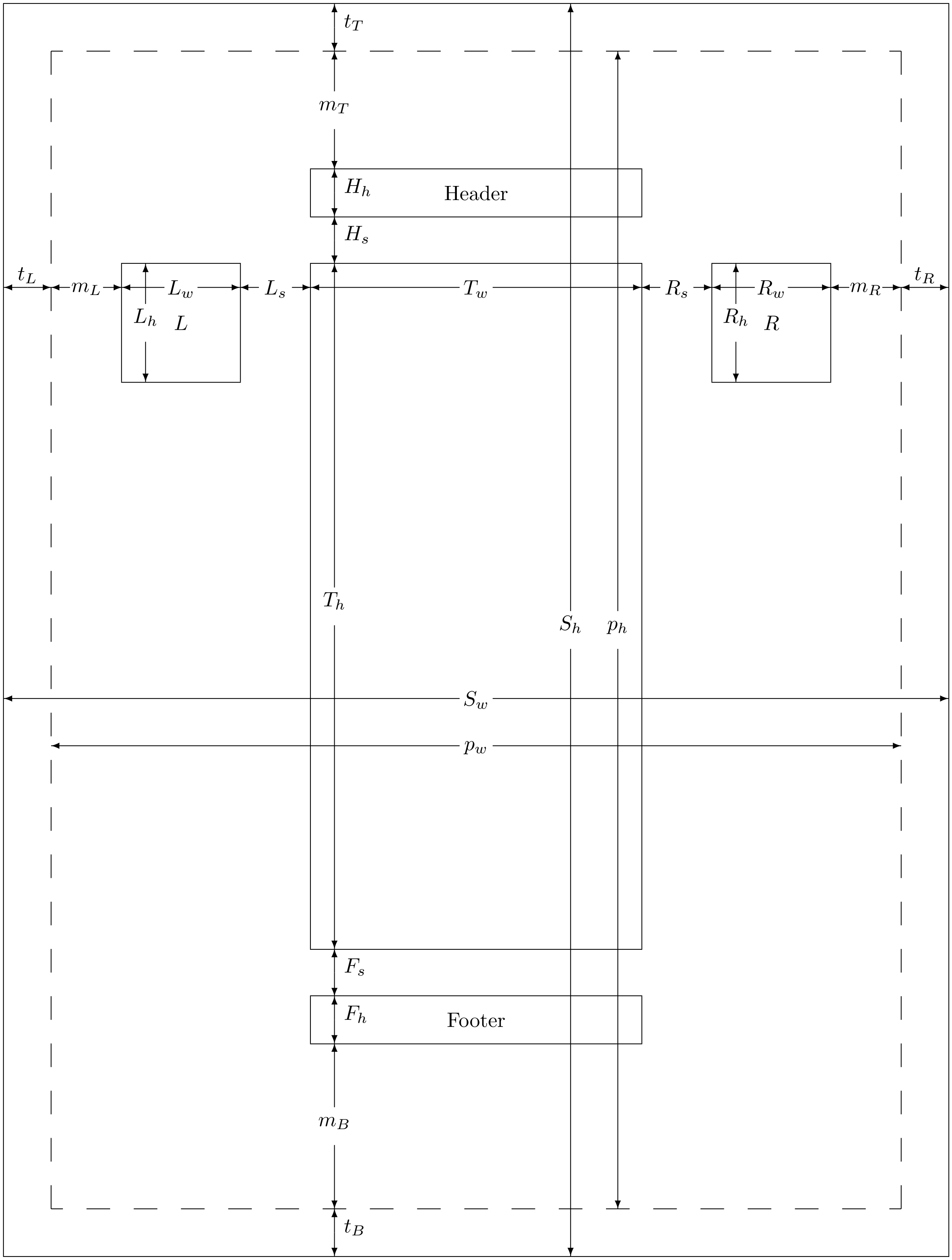


**Figure 1**: The parts of a page: $S$ is stock size, $p$ is paper size, $t$ are trims, $m$ is margin, $T$ is the text block, $L$ and $R$ are marginal note areas, and $F$ and $H$ are footer and header resp.

```
\xref_name_key {This is a name}
  \xref_valuepage_key {1}
    \xref_page_key {i}
```

Property lists are a natural way to store series/collections of information. One such application could be the babel language strings or, as we shall see, an array of length registers with saved values.

For our purposes we will store the property lists with names like `\g_page_recto_plist` with the following contents:

```
\l_page_recto_textheight_dim
  {\l_page_recto_textheight_dim}
\l_page_recto_textwidth_dim
  {\l_page_recto_textwidth_dim}
...
```

The reason is that we can have some application setting these parameters locally and then we can use the extended list processing tools of the LaTeX3 kernel to map a function on each info-pair and extract the data. Here's a small example of how this is done:

```
\tlp_gset:Nx \g_page_recto_tlp {
  \prop_map_function:NN
    \g_page_recto_plist
    \page_extract_dimensions:Nn
}
\def_new:NNn\page_extract_dimensions:Nn 2{
  \exp_not:N\dim_set:Nn #1{\dim_use:N #2}
}
```

`\tlp_gset:Nx` does an `\xdef` on its second argument and stores it in the global *token list pointer* `\g_page_recto_tlp`. `\prop_map_function:NN` is an expandable mapping operation which places its second argument in front of each info-pair of the property list, which is given as the first argument. In our example this argument is the auxiliary function `\page_extract_dimensions:Nn` which is called to get the current value of the parameter and preparing it to be set. The end result is a macro containing

```
\dim_set:Nn\l_page_recto_textheight_dim
  {536.0pt}
\dim_set:Nn\l_page_recto_textwidth_dim
  {310.0pt}
...
```

At the time of the shipout this list is run and then we also run a simple

```
\dim_gset:Nn
  \g_page_std_textheight_dim
  {\l_page_recto_textheight_dim}
\dim_gset:Nn
  \g_page_std_textwidth_dim
  {\l_page_recto_textwidth_dim}
...
```

Note that all of this does not touch the original property list, so a number of operations can be performed by mapping functions onto it. For example, one could reset all parameters to some special value (typically negative) and then a second mapping function could check if all parameters have indeed been set.

### 3.2.2 Templates

LaTeX3 provides the concept of *templates*, which are, in short, parameterized functions. As arguments, a template takes a list of named parameters given in the well-known 'keyval' syntax plus additional arguments, which are often user input. The template converts the named parameters into macro or register assignments and uses these when running the actual code in the template. While a template can be used directly, one often defines various *named* instances of a template, which is the template function run with a specific set of parameters. Defining and using an instance instead of running the entire template at runtime is faster in terms of execution time but also has the advantage that one can store several different versions of a template and then use specific instances depending on the needs at hand.

As a small example, let's imagine a template type for producing split level fractions such as $^{3}/_{7}$. The template type receives three arguments from the user:

1. Numerator.
2. Separator. In case of `\NoValue`, i.e., no argument, use a default symbol instead.
3. Denominator.

So for this template type one can define several different templates depending on the needs of the user and the font in question. With fonts not containing superior and inferior numbers one will have to manually raise and lower the characters whereas more advanced font sets such as Minion Pro contain these characters and then one can use a much simpler template, which basically just inserts the user input as-is. Figure 2 shows a complete example of a simple template and a possible user interface to the template using xparse, while Figure 3 shows the resulting output. The template defined in this example uses `\textfractionsolidus` as the default separator symbol.

In the example you can see how one can run the template directly, how to define instances and how to use instances as part of defining document syntax with xparse. An interesting observation is that using the solidus from Times works rather well with Computer Modern instead of the unusually large solidus found there.

```
\documentclass[12pt]{article}
\usepackage{template,xparse,textcomp,l3box}
\CodeStart
\dim_new:N \l_splitfrac_size_dim
\dim_new:N \l_splitfrac_pre_kern_dim
\dim_new:N \l_splitfrac_post_kern_dim
\box_new:N \l_splitfrac_tmpa_box
\DeclareTemplateType{splitfrac}{3}
\DeclareTemplate{splitfrac}{text}{3}{
  numerator-format    = f1 [#1] \splitfrac_numerator_format:n ,
  denominator-format  = f1 [#1] \splitfrac_denominator_format:n ,
  separator-format    = f1 [#1] \splitfrac_separator_format:n ,
  numerator-scale     = n  [.6]  \l_splitfrac_scale_tlp ,
  separator-symbol    = n [\textfractionsolidus]
                          \l_splitfrac_separator_symbol_tlp,
  separator-font      = n [\DelayEvaluation{\f@family}]  \l_splitfrac_separator_font_tlp ,
  separator-pre-kern  = l [\DelayEvaluation{-.1em}] \l_splitfrac_pre_kern_dim ,
  separator-post-kern = l [\DelayEvaluation{-.15em}] \l_splitfrac_post_kern_dim
}
{
  \DoParameterAssignments
  \sbox\l_splitfrac_tmpa_box{
    \splitfrac_separator_format:n {
      \fontfamily{\l_splitfrac_separator_font_tlp }\selectfont
      \IfNoValueTF{#2}{\l_splitfrac_separator_symbol_tlp}{#2}
    }  }
  \mbox{
    \dim_set:Nn\l_splitfrac_size_dim {
      \l_splitfrac_scale_tlp \dim_eval:n{\f@size pt}  }
    \raisebox{\box_ht:N \l_splitfrac_tmpa_box -\height }{
      \splitfrac_numerator_format:n {
        \fontsize{\l_splitfrac_size_dim }{\baselineskip}\selectfont
        #1  }  }
    \kern\l_splitfrac_pre_kern_dim
    \box_use:N \l_splitfrac_tmpa_box
    \kern\l_splitfrac_post_kern_dim
    \raisebox{-\box_dp:N \l_splitfrac_tmpa_box}{
      \splitfrac_denominator_format:n {
        \fontsize{\l_splitfrac_size_dim }{\baselineskip}\selectfont
        #3  }  }  }
}
\DeclareDocumentCommand\splitfrac{mom}{
  \IfExistsInstanceTF{splitfrac}{\f@family}
  {\UseInstance{splitfrac}{\f@family}}
  {\UseTemplate{splitfrac}{text}{}}
  {#1}{#2}{#3}
}
\CodeStop
\begin{document}
\UseTemplate{splitfrac}{text}{}{34}{\NoValue}{15},
\fontfamily{ptm}\selectfont\textonehalf,
\UseTemplate{splitfrac}{text}{
  separator-pre-kern=0pt,
  separator-post-kern=0pt}{1}{\NoValue}{2},
\DeclareInstance{splitfrac}{ptm}{text}{
  separator-pre-kern=0pt,
  separator-post-kern=0pt}
\UseInstance{splitfrac}{ptm}{1}{\NoValue}{2},
\splitfrac{34}{56},
\normalfont
\splitfrac{34}{56},
\UseTemplate{splitfrac}{text}{
  separator-font=ptm,
  separator-pre-kern=0pt,
  separator-post-kern=0pt}{34}{\NoValue}{15}
\end{document}
```

**Figure 2**: Example document for split level fractions using templates.

$^{34}/_{15}$, $\mathbf{{}^{1}/2}$, ${}^{1}/2$, ${}^{1}/2$, ${}^{34}/\mathbf{56}$, $^{34}/_{56}$, ${}^{34}/15$

**Figure 3**: Output from running the example document shown in Figure 2.

Templates are an integral part of the design aspect in LaTeX3 and worth a closer look; see the documentation in [1]. We will not discuss them further here, but this brief introduction should be enough to give you an idea of the potential. At this point it should come as no surprise that our present page layout parameters are well suited for templates.

### 3.3 Putting the pieces together

We've now presented the tools and are ready to define templates for setting the page layouts. A template for setting all parameters is likely to be very large, so we'll just show a minimal example. We define a template type `pagelayout` which takes one argument: the name for the page type. Next we declare the template `minimal` which has two keys: `textheight` and `textwidth` with default values of 8 in and 6.5 in respectively (same as in the standard LaTeX class file minimal). The template then sets the local parameters for the page type and stores them in the token list pointer as shown Section 3.2.1. This simple template setup looks like this:

```
\DeclareTemplateType{pagelayout}{1}
\DeclareTemplate{pagelayout}{minimal}{1}{
  textheight = l [8in]
            \g_page_std_textheight_dim ,
  textwidth  = l [6.5in]
            \g_page_std_textwidth_dim ,
}{
  \DoParameterAssignments
  \dim_set:cn {l_page_#1_textheight_dim}
              {\g_page_std_textheight_dim}
  \dim_set:cn {l_page_#1_textwidth_dim}
              {\g_page_std_textwidth_dim}
  \page_store_page_dimensions:n {#1}
}
```

Using the template is fairly straightforward. The defaults are used unless you use a key in which case the new value is used. For demonstration purposes, we will make the recto and verso layouts different in height; the recto is one inch longer than the verso.

```
\UseTemplate{pagelayout}{minimal}
  { textheight= 9in } {minimal-recto}
\UseTemplate{pagelayout}
            {minimal}{}{minimal-verso}
\UsePageLayout{2}
     {minimal-recto,minimal-verso}
```

The `\UsePageLayout` command instructs LaTeX to switch between two different parameter sets which are then given as a comma separated list in the second argument. The command takes effect on the following page so it can also be used mid-document.

The attentive reader may have noticed that the interface presented above does not allow for defining two page layouts at a time with common margin ratios etc., as would usually be required. This will be supported, but it requires a different template type, taking two arguments. Alternatively, a package like geometry in its LaTeX3 incarnation could easily stick to the same user interface as it has now and simply pass the information on to two templates at a time if two-sided documents are produced.

## 4 Concluding remarks

The interface for page layouts described in this article exists only as an unreleased prototype at the time of writing. There is still a bit of work to do on it, especially in the area of integrating it with the experimental output routine xor. Currently xor stands at 5000+ lines so this is something which has to be done very carefully! When this is done it will be possible to specify page layouts on a per-page basis, which is especially useful for float pages.

Other than that we will produce various different templates plus provide tools for autocompletetion and error checking. When ready for release it will be put in our publically available Subversion (SVN) repository which can be reached by pointing your SVN client to

`http://www.latex-project.org/svnroot/experimental/trunk/`

### References

[1] Various authors. LaTeX project web site directory for experimental code. `http://www.latex-project.org/svnroot/experimental/trunk`.

[2] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LaTeX Companion.* Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004.

[3] Frank Neukam, Markus Kohm, Axel Kielhorn, and Jens-Uwe Morawski. The KOMA-Script Bundle, March 2005.

[4] Hideo Umeki. The `geometry` package, July 2002.

# MKII–MKIV

Hans Hagen
Hasselt, The Netherlands
http://luatex.org
http://pragma-ade.com
pragma (at) wxs dot nl

### Abstract

Some time ago the pdfTEX development team set a road map for the next generation of pdfTEX. It was decided that within a reasonable timeframe pdfTEX would go 24/32 bit and support OpenType fonts. At the same time, after some preliminary experiments, it was decided that it made sense to embed the Lua scripting engine into TEX.

Currently, Taco Hoekwater and I spend a lot of time exploring the possibilities of a TEX engine enhanced in this way. Downward compatibility as well as the traditional stability are very important conditions in this proces. Because we're both actively involved in the development of ConTEXt and both know the internals quite well, we can easily test Lua based alternatives and explore possible routes and needed extensions.

This paper presents some possible features of LuaTEX (some may go, some may change, others will be added). I will discuss how they may influence the way ConTEXt will be organized in the future as well as how code development may change in nature and influence users. I will also demonstrate some of the features of that LuaTEX currently offers.

## 1 Introduction

The development of LuaTEX started with a few email exchanges between me and Hartmut Henkel. I had played a bit with Lua in SCITE and somehow felt that it would fit into TEX quite well. Hartmut made me a version of pdfTEX which provided a `\lua` command. After exploring this road a bit Taco Hoekwater took over and we quickly reached a point where the pdfTEX development team could agree on following this road to the future.

The development was boosted by a substantial grant from Colorado State University in the context of Professor Idris Samawi Hamid's Oriental TEX Project. This project aims at bringing features into TEX that will permits ConTEXt to do high quality Arab typesetting. Due to this grant Taco could spend substantial time on development, which in turn meant that I could start playing with more advanced features.

This document is not so much a manual as a report on the state of affairs. Things may evolve and the way things are done may change, but it felt right to keep track of the process.

## 2 From Mark II to Mark IV

In 2005 the development of LuaTEX started, a further development of pdfTEX and a precursor to pdfTEX version 2. This TEX variant will provide:

- 21–32 bit internals plus a code cleanup
- flexible support for OpenType fonts
- an internal UTF data flow
- the bidirectional typesetting of Aleph
- Lua callbacks to relevant TEX internals
- some extensions to TEX (for instance math)
- efficient communication with MetaPost

In the tradition of TEX this successor will be downward compatible in essential ways and in the end, there is still pdfTEX version 1 as fall back.

In the mean time we have seen another Unicode variant show up: XƎTEX, which is under active development, uses external libraries, provides access to the fonts on the operating system, etc.

From the beginning, ConTEXt always worked with all engines. This was achieved by conditional code blocks: depending on what engine was used, different code was put in the format and/or used at runtime. Users normally were unaware of this. Examples of engines are ε-TEX, Aleph, and XƎTEX. Because nowadays all engines provide the ε-TEX features, in August 2006 we decided to consider those features to be present and drop providing the standard TEX compatible variants. This is a small effort because all code that is sensitive for optimization already has ε-TEX code branches for many years.

However, with the arrival of LuaTEX, we need a more drastic approach. Quite a lot existing code can go away, to be replaced by different solutions. Where TEX code ends up in the format file, along with its state, Lua code will be initiated at run time, after a Lua instance is started. ConTEXt reserves its own instance of Lua.

Most of this will go unnoticed for the users because the user interface will not change. For developers however, we need to provide a mechanism to deal with these issues. This is why, for the first time in ConTEXt's history, we will officially use a kind of version tag. When we changed the low level interface from Dutch to English we jokingly talked of version 2. So, it makes sense to follow this lead.

- ConTEXt Mark I  At that moment we still had a low level Dutch interface, invisible for users but not for developers.
- ConTEXt Mark II  We now have a low level English interface, which we hoped (and indeed saw happen) would trigger more development by users.
- ConTEXt Mark IV  This is the next generation of ConTEXt, with parts re-implemented. At some points, it's a drastic system overhaul.

Keep in mind that the functionality does not change, although in some places, for instance fonts, Mark IV may provide additional functionality. Most users will not notice the difference (maybe apart from performance and convenience) since at the user interface level nothing changes (most of it deals with typesetting, not low level details).

The hole in the numbering permits us to provide a Mark III version as well. Once XƎTEX is stable, we may use that slot for XƎTEX specific implementations.

As of August 2006 the banner has been adapted to this distinction:

```
ver: 2006.09.06 22:46 MK II  fmt: 2006.9.6
ver: 2006.09.06 22:47 MK IV  fmt: 2006.9.6
```

This numbering system is reflected at the file level in such a way that we can keep developing the way we do, i.e. no files all over the place, in subdirectories, etc.

Most of the system's core files are not affected, but some may be, like those dealing with fonts, input and output encodings, file handling, etc. Those files may come with different suffixes:

- `somefile.tex`: the main file, implementing the interface and common code
- `somefile.mkii`: mostly existing code, suitable for good old TEX (ε-TEX, pdfTEX, Aleph).
- `somefile.mkiv`: code optimized for use with LuaTEX, which could follow completely different approaches
- `somefile.lua`: Lua code, loaded at format generation time and/or runtime

As said, some day `somefile.mkiii` code may show up. Which variant is loaded is determined automatically at format generation time and/or at run time.

## 3 How Lua fits in

### Introduction

Here I will discuss a few of the experiments that drove the development of LuaTEX. It describes the state of affairs around the time that we were preparing for TUG 2006. This development was rather demanding for Taco and me but also much fun. We were in a kind of permanent Skype chat session, with binaries flowing in one direction and TEX and Lua code the other way. By gradually replacing (even critical) components of ConTEXt we had a real test bed and torture tests helped us to explore and debug at the same time. Because Taco uses Linux as platform and I mostly use Windows, we could investigate platform dependent issues conveniently. While reading this text, keep in mind that this is just the beginning of the game.

I will not provide sample code here. When possible, the Mark IV code transparently replaces Mark II code and users will seldom notices that something happens in different way. Of course the potential is there and future extensions may be unique to Mark IV.

### Compatibility

The first experiments, already conducted with the experimental versions involved runtime conversion of one type of input into another. An example of this is the (TI) calculator math input handler that converts a rather natural math sequence into TeX and feeds that back into TeX. This mechanism eventually will evolve into a configurable math input handler. Such applications are unique to Mark IV code and will not be backported to Mark II. The question is where downward compatibility will become a problem. We don't expect many problems, apart from occasional bugs that result from splitting the code base, mostly because new features will not affect older functionality. Because we have to reorganize the code base a bit, we also use this opportunity to start making a variant of ConTeXt which consists of building blocks: MetaTeX. This is less interesting for the average user, but may be of interest for those using ConTeXt in workflows where only part of the functionality is needed.

## MetaPost

Of course, when I experiment with such new things, I cannot let MetaPost leave untouched. And so, in this early stage of LuaTeX development I decided to play with two MetaPost related features: conversion and runtime processing.

Conversion from MetaPost output to PDF is currently done in pure TeX code. Apart from convenience, this has the advantage that we can let TeX take care of font inclusions. The tricky part of this conversion is that MetaPost output has some weird aspects, like dvips specific linewidth snapping. Another nasty element in the conversion is that we need to transform paths when pens are used. Anyhow, the converter has reached a rather stable state by now.

One of the ideas with MetaPost version $1^+$ is that we will have an alternative output mode. From the perspective of LuaTeX it makes sense to have a Lua output mode. Whatever converter we use, it needs to deal with Metafun specials. These are responsible for special features like transparency, graphic inclusion, shading, and more. Currently we misuse colors to signal such features, but the new pre/post path hooks permit more advanced implementations. Experimenting with such new features is easier in Lua than in TeX.

The Mark IV converter is a multi-pass converter. First we clean up the MetaPost output, then convert the PostScript code into Lua calls. We assume that this Lua code can eventually be output directly from MetaPost. We then evaluate this converted Lua blob, resulting in TeX commands. Example:

```
1.2 setlinejoin
```

turned into:

```
mp.setlinejoin(1.2)
```

becoming:

```
\PDFcode{1.2 j}
```

which is, when the pdfTeX driver is active, equivalent to:

```
\pdfliteral{1.2 j}
```

Of course, when paths are involved, more things happen behind the scenes, but in the end an `mp.path` enters the Lua machinery.

When the Mark IV converter reached a stable state, tests demonstrated then the code was up to 20% slower that the pure TeX alternative on average graphics, and but faster when many complex path transformations (due to penshapes) need to be done. This slowdown was due to the cleanup (using expressions) and intermediate conversion. Because Taco both develops LuaTeX and maintains and extends MetaPost, we conducted experiments that combine features of these programs. As a result of this, shortcuts found their way into the MetaPost output.

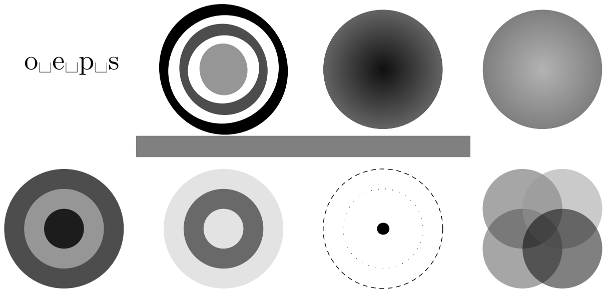


**Figure 1** Our converter test figure.

Cleaning up the MetaPost output using Lua expressions takes relatively much time. However, starting with version 0.970 MetaPost uses a preamble, which permits not only short commands, but also gets rid of the weird linewidth and filldraw related PostScript constructs. The moderately complex graphic that we use for testing (figure 1) takes over 16 seconds when converted 250 times. When we enable shortcuts we can avoid part of the cleanup and runtime goes down to under 7.5 seconds. This is significantly faster than the Mark II code. We did experiments with simulated Lua output from MetaPost and then the Mark IV converter really flies. The values on Taco's system are given in parentheses:

| **prologues/ mpprocset** | **1/0** | **1/1** | **2/02/1** |
|---|---|---|---|
| Mark II | 8.5 ( 5.7) | 8.0 (5.5) | 8.8 8.5 |
| Mark IV | 16.1 (10.6) | 7.2 (4.5) | 16.3 7.4 |

The main reason for the huge difference in the Mark IV times is that we do a rigorous cleanup of the older MetaPost output in order to avoid the messy (but fast) code that we use in the Mark II converter. Think of:

```
0 0.5 dtransform truncate idtransform
  setlinewidth pop
closepath gsave fill grestore stroke
```

In the Mark II converter, we push every number or keyword on a stack and use keywords as trigger points. In the Mark IV code we convert the stack based PostScript calls to Lua function calls. The top-level `0...pop` and `closepath...stroke` expressions are each converted to single calls first. When `prologues` is set to 2, such lines no longer show up and are replaced by simple calls accompanied by definitions in the preamble. Not only that, instead of verbose keywords, one or two character shortcuts are used. This means that the Mark II code can be faster when procsets are used because shorter strings end up in the stack and comparison happens faster. On the other hand, when no procsets are used, the runtime is longer because of the larger preamble.

Because the converter is used outside ConTeXt as well, we support all combinations in order not to get error messages, but the converter is supposed to work with the following settings:

```
prologues := 1 ;
mpprocset := 1 ;
```

We don't need to set `prologues` to 2 (font encodings in file) or 3 (also font resources in file). So, in the end, the comparison in speed comes down to 8.0 seconds for Mark II code and 7.2 seconds for the Mark IV code when using the latest greatest MetaPost. When we simulate Lua output from MetaPost, we end up with 4.2 seconds runtime and when MetaPost could produce the converter's TeX commands, we need only 0.3 seconds for embedding the 250 instances. This includes TeX taking care of handling the specials, some of which demand building moderately complex PDF data structures.

But, conversion is not the only factor in convenient MetaPost usage. First of all, runtime MetaPost processing takes time. The actual time spent on handling embedded MetaPost graphics is also dependent on the speed of starting up MetaPost, which in turn depends on the size of the TeX trees used: the bigger these are, the more time KPSE spends on loading the `ls-R` databases. Eventually this bottleneck may go away when we have MetaPost as a library. (In ConTeXt one can also run MetaPost between runs. Which method is faster depends on the amount and complexity of the graphics.)

Another factor in dealing with MetaPost, is the usage of text in a graphic (`btex`, `textext`, etc.). Taco Hoekwater, Fabrice Popineau and I did some experiments with a persistent MetaPost session in the background in order to simulate a library. The results look very promising: the overhead of embedded MetaPost graphics goes to nearly zero, especially when we also let the parent TeX job handle the typesetting of texts. A side effect of these experiments was a new mechanism in ConTeXt (and Metafun) where TeX did all typesetting of labels, and MetaPost only worked with an abstract representation of the result. This way we can completely avoid nested TeX runs (the ones triggered by MetaPost). This also works ok in Mark II mode.

Using a persistent MetaPost run and piping data into it is not the final solution if only because the terminal log becomes messed up too much, and also because intercepting errors becomes very messy. In the end we need a proper library approach, but the experiments demonstrated that we needed to go this way: handling hundreds of complex graphics that hold typeset paragraphs (being slanted and rotated and more by MetaPost), took mere seconds compared to minutes when using independent MetaPost runs for each job.

## Characters

Because LuaTeX is UTF based, we need a different way to deal with input encoding. For this purpose there are callbacks that intercept the input and convert it as needed. For ConTeXt this means that the regime-related modules get Lua-based counterparts. As a prelude to advanced character manipulations, we already load extensive Unicode and conversion tables, with the benefit of being able to handle case handling with Lua.

The character tables are derived from Unicode tables and Mark II ConTeXt data files, and generated using mtxtools. The main character table is pretty large, and this made us experiment a bit with efficiency. It was in this stage that we realized that it made sense to use precompiled Lua code (using `luac`). During format generation we let ConTEXt keep track of used Lua files and compile them on the fly. For a production run, the compiled files were loaded instead.

Because at that stage LuaTEX was already a merge between pdfTEX and Aleph, we had to deal with pretty large format files. Thus, on 2006-09-18 the ConTEXt format with the English user interface amounted to:

| **luatex** | **pdftex** | **xetex** | **aleph** |
|---|---|---|---|
| 9 552 042 | 7 068 643 | 8 374 996 | 7 942 044 |

One reason for the large size of the format file is that the memory footprint of a 32-bit TEX is larger than that of good old TEX, even with some of the clever memory allocation techniques used in LuaTEX. After some experiments where size and speed were measured Taco decided to compress the format using level 3 zip compression. This brilliant move lead to the following sizes on 2006-10-23:

| **luatex** | **pdftex** | **xetex** | **aleph** |
|---|---|---|---|
| 3 135 568 | 7 095 775 | 8 405 764 | 7 973 940 |

The first zipped versions were smaller (around 2.3 meg), but in the meantime we moved the Lua code into the format and the character related tables take some space.

## Debugging

In the process of experimenting with callbacks I played a bit with handling TEX error information. An option is to generate an HTML page instead of the usual blob of text on the terminal.

Playing with such features gives us an impression of what kind of access we need to TEX's internals. It also formed a starting point for conversion routines and a mechanism for embedding Lua code in HTML pages generated by ConTEXt.

## File I/O

Replacing TEX's input and output handling is non-trivial. Not only is the code quite interwoven in the Web2C source, but there is also the KPSE library to deal with. This means that quite a few callbacks are needed to handle the different types of files. Also, there is output to the log and terminal to deal with.

Getting this done took us quite some time and testing and debugging was good for some headaches. The mechanisms changed a few times, and TEX and Lua code was thrown away as soon as better solutions came around. Because we were testing on real documents, using a fully loaded ConTEXt we could converge to a stable version after a while.

Getting this I/O stuff done is tightly related to generating the format and starting up LuaTEX. If you want to overload the file searching and I/O handling, you need overload as soon as possible. Because LuaTEX is also supposed to work with the existing KPSE library, we still have that as fallback, but in principle one could think of a KPSE free version, in which case the default file searching is limited to the local path and memory initialization also reverts to the hard coded defaults. A complication is that the soure code has KPSE calls and references to KPSE variables all over the place, so occasionally we run into interesting bugs.

Anyhow, while Taco hacked his way around the code, I converted my existing Ruby based KPSE variant into Lua and started working from that point. The advantage of having our own I/O handler is that we can go beyond KPSE. For instance, since LuaTEX has, among a few others, the zip libraries linked in, we can read from zip files, and keep all TEX related files in TDS compliant zip files as well. This means that one can say:

```
\input zip::somezipfile::somefile.tex
\input zip://some.zip/subdir/somefile.tex
```

and use similar references to access files. Of course we had to make sure that KPSE like searching in the TDS (standardized TEX trees) works smoothly. There are plans to link the curl library into LuaTEX, so that we can go beyond this and access network repositories.

Of course, in order to be more or less KPSE and Web2C compliant, we also need to support this paranoid file handling, so we provide mechanisms for that as well. In addition, we provide ways to create sandboxes for system calls.

Getting to intercept all log output (well, most log output) was a problem in itself. For this I used a (preliminary) XML based log format, which will make log parsing easier. Because we have full control over file searching, opening and closing, we can also provide more information about what files are loaded. For instance we can now easily trace what TFM files TEX reads.

Implementing additional methods for locating and opening files is not that complex because the library that ships with ConTeXt is already prepared for this. For instance, implementing support for:

```
\input http://example.net/somefile.tex
```

involved just a few lines of code, most of which deals with caching the files. Because we overload the whole I/O handling, this means that the following works ok:

```
\placefigure
  {http handling}
  {\externalfigure
    [http://www.pragma-ade.com/show-gra.pdf]
    [page=1,width=\textwidth]}
```

Other protocols, like FTP, are also supported, so one can say:

```
\typefile {ftp://anonymous:@ctan.org/\
  tex-archive/systems/knuth/lib/plain.tex}
```

On the agenda is playing with databases, but by the time that we enter that stage linking the `curl` libraries into LuaTeX should have taken place.

## Verbatim

The advance of LuaTeX also permitted us to play with a long standing wish for catcode tables, a mechanism to quickly switch between different ways of treating input characters. An example of a place where such changes take place is verbatim (and, in ConTeXt, when dealing with XML input).

We had already encountered the phenomena that when piping back results from Lua to TeX, we needed to take care of catcodes so that TeX would see the input as we wished. Earlier experiments with applying `\scantokens` to a result and thereby interpreting the result conforming the current catcode regime was not sufficient or at least not handy enough, especially in the perspective of fully expandable Lua results. To be honest, `\scantokens` was rather useless for this purposes due to its pseudo file nature and its end-of-file handling but in LuaTeX we now have a convenient `\scantextokens` which has no side effects.

Once catcode tables were in place, and the relevant ConTeXt code adapted, I could start playing with one of the trickier parts of TeX programming: typesetting TeX using TeX, or verbatim. Because in ConTeXt verbatim is also related to buffering and pretty printing, all these mechanism were handled at once. It proved to be a pretty good test case for writing Lua results back to TeX, because anything you can imagine can and will interfere (line endings, catcode changes, looking ahead for arguments, etc). This is one of the areas where Mark IV code will make things look more clean and understandable, especially because we could move all kind of postprocessing (needed for pretty printing, i.e. syntax highlighting) to Lua.

Pretty printing 1000 small (one line) buffers and 5000 simple `\type` commands perform as follows:

| | TeX normal | TeX pretty | Lua normal | Lua pretty |
|---|---|---|---|---|
| buffer | 2.5 (2.35) | 4.5 (3.05) | 2.2 (1.8) | 2.5 (2.0) |
| inline | 7.7 (4.90) | 11.5 (7.25) | 9.1 (6.3) | 10.9 (7.5) |

Between braces the runtime on Taco's more modern machine is shown. It's not that easy to draw conclusions from this because TeX uses files for buffers and with Lua we store buffers in memory. For inline verbatim, Lua calls bring some overhead, but with more complex content, this becomes less noticeable. Also, the Lua code is probably less optimized than the TeX code, and we don't know yet what benefits a Just In Time Lua compiler will bring.

## XML

One interesting result is that the first experiments with XML processing don't show the expected gain in speed. This is due to the fact that the ConTeXt XML parser is highly optimized. However, if we want to load a whole XML file, for instance the formal ConTeXt interface specification `cont-en.xml`, then we can bring down loading time (as well as TeX memory usage) down from multiple seconds to a blink of an eye. Experiments with internal mappings and manipulations demonstrated that we may not so much need an alternative for the current parser, but can add additional, special purpose ones.

We may consider linking XSLTPROC into LuaTeX, but this is yet undecided. After all, the problem of typesetting does not really change, so we may as well keep the process of manipulating and typesetting separated.

## Multipass data

Those who know ConTeXt a bit will know that it may need multiple passes to typeset a document. ConTeXt not only keeps track of index entries, list entries, cross references, but also optimizes some of the output based on information gathered in previous passes. Especially so-called “two-pass data” and positional information put some demands on memory and runtime. Two-pass data is collapsed in lists because otherwise we would run out of memory (at least this was true years ago when these mechanisms were introduced). Positional information is stored in hashes and has always put a bit of a burden on the size of a so-called utility file (ConTeXt stores all information in one auxiliary file).

These two datatypes were the first we moved to a Lua auxiliary file and eventually all information will move there. The advantage is that we can use efficient hashes (without limitations) and only need to run over the file once. And Lua is incredibly fast in loading the tables where we keep track of these things. For instance, a test file storing and reading 10 000 complex positions takes 3.2 seconds runtime with LuaTeX but 8.7 seconds with traditional pdfTeX. Imagine what this will save when dealing with huge files (400 page 300 Meg files) that need three or more passes to be typeset. And, now we can without problems bump position tracking to the millionth decimal place.

## 4 Initialization revised

Initializing LuaTeX in such a way that it does what you want it to do your way can be tricky. This has to do with the fact that if we want to overload certain features (using callbacks) we need to do that before the originals start doing their work. For instance, if we want to install our own file handling, we must make sure that the built-in file searching does not get initialized. This is particularly important when the built in search engine is based on the KPSE library. In that case the first serious file access will result in loading the `ls-R` filename databases, which will take an amount of time more or less linear with the size of the TeX trees. Among the reasons why we want to replace KPSE are the facts that we want to access zip files, do more specific file searches, use HTTP, FTP and whatever comes around, integrate ConTeXt specific methods, etc.

Although modern operating systems will cache files in memory, creating the internal data structures (hashes) from the rather dumb files take some time. On the machine where I was developing the first experimental LuaTeX code, we’re talking about 0.3 seconds for pdfTeX. One would expect a Lua based alternative to be slower, but it is not. This may be due to the different implementation, but for sure the more efficient file cache plays a role as well. So, by completely disabling KPSE, we can have more advanced I/O related features (like reading from zip files) at about the same speed (or even faster). In due time we will also support progname (and format) specific caches, which speeds up loading. In case one wonders why we bother about a mere few hundreds of milliseconds: imagine frequent runs from an editor or sub-runs during a job. In such situation every speed up matters.

So, back to initialization: how do we initialize LuaTeX. The method described here is developed for ConTeXt but is not limited to this macro package; when one tells TeXexec to generate formats using the `--luatex` directive, it will generate the ConTeXt formats as well as mptopdf using this engine.

For practical reasons, the Lua based I/O handler is KPSE compliant. This means that the normal `texmf.cnf` and `ls-R` files can be used. However, their content is converted in a more Lua friendly way. Although this can be done at runtime, it makes more sense to do this in advance using the luatools utility. The files involved are:

| **input** | **raw input** | **runtime input / fallback** |
|---|---|---|
| | `ls-R` | `files.luc` / `files.lua` |
| `texmf.lua` | `temxf.cnf` | `configuration.luc` / `configuration.lua` |

In due time luatools will generate the directory listing itself (for this some extra libraries need to be linked in). The configuration file(s) eventually will move to a Lua table format, and when a `texmf.lua` file is present, that one will be used.

```
luatools --generate
```

This command will generate the relevant databases. Optionally you can provide `--minimize` which will generate a leaner database, which in turn will bring down loading time to (on my machine) about 0.1 sec instead of 0.2 seconds. The `--sort` option will give nicer intermediate (`.lua`) files that are more handy for debugging.

When done, you can use luatools roughly like kpsewhich, for instance to locate files:

```
luatools texnansi-lmr10.tfm
luatools --all tufte.tex
```

You can also inspect its internal state, for instance:

```
luatools --variables  --pattern=TEXMF
luatools --expansions --pattern=context
```

This will show you the (expanded) variables from the configuration files. Normally you don't need to go that deep into the belly.

The luatools script can also generate a format and run LuaTEX. For ConTEXt this is normally done with the TEXexec wrapper, for instance:

```
texexec --make --all --luatex
```

When dealing with this process we need to keep several things in mind:

- LuaTEX needs a Lua startup file in both ini and runtime mode
- these files may be the same but may also be different
- here we use the same files but a compiled one in runtime mode
- we cannot yet use a file location mechanism

A .luc file is a precompiled Lua chunk. In order to guard consistency between Lua code and tex code, ConTEXt will preload all Lua code and store them in the bytecode table provided by LuaTEX. How this is done is another story. Contrary to these tables, the initialization code can not be put into the format, if only because at that stage we still need to set up memory and other parameters.

In our case, especially because we want to overload the I/O handler, we want to store the startup file in the same path as the format file. This means that scripts that deal with format generation also need to take care of (relocating) the startup file. Normally we will use TEXexec but we can also use luatools.

Say that we want to make a plain format. We can call luatools as follows:

```
luatools --ini plain
```

This will give us (in the current path):

```
120,808 plain.fmt
  2,650 plain.log
 80,767 plain.lua
 64,807 plain.luc
```

From now on, only the plain.fmt and plain.luc file are important. Processing a file

```
test \end
```

can be done with:

```
luatools --fmt=./plain.fmt test
```

This returns:

```
This is luaTeX, Version 3.141592-0.1-alpha-
20061018 (Web2C 7.5.5)
(./test.tex [1] )
Output written on test.dvi (1 page, 260 bytes).
Transcript written on test.log.
```

which looks rather familiar. Keep in mind that at this stage we still run good old Plain TEX. In due time we will provide a few files that will making work with Lua more convenient in Plain TEX, but at this moment you can already use, for instance, \directlua.

In case you wonder how this is related to ConTEXt, well only to the extent that it uses a couple of rather generic ConTEXt related Lua files.

ConTEXt users can best use TEXexec which will relocate the format related files to the regular engine path. In luatools terms we have two choices:

```
luatools --ini cont-en
luatools --ini --compile cont-en
```

The first case uses context.lua as the startup file. This Lua file creates the cont-en.luc runtime file. In the second case, luatools will create a cont-en.lua file and compile that one. An even more specific call would be:

```
luatools --ini --compile \
  --luafile=foo.lua cont-en
luatools --ini --compile \
  --lualibs=foo1.lua,foo2.lua cont-en
```

This call does not make much sense for ConTEXt. Keep in mind that luatools does not set up user specific configurations, for instance the --all switch in TEXexec will set up all patterns.

I know that it sounds a bit messy, but till we have a more clear picture of where LuaTEX is heading this is the way to proceed. The average ConTEXt user won't notice those details, because TEXexec will take care of things.

Currently we follow the TDS and Web2C conventions, but in the future we may follow different or additional approaches. This may as well be driven by more complex I/O models. For the moment extensions still fit in. For instance, in order to support access to remote resources and related caching, we have added to the configuration file the variable:

```
TEXMFCACHE = $TMP;$TEMP;$TMPDIR;$HOME;\
             $TEXMFVAR;$VARTEXMF;.
```

## 5 An example: CalcMath

### introduction

For a long time TEX's way of coding math has dominated the typesetting world. However, this kind of coding is not that well suited for non-academics, such as schoolchildren. Often kids do know how to key in math because they use advanced calculators. So, when a couple of years ago we were implementing a workflow where kids could fill in their math workbooks (with exercises) on-line, it made sense to support so-called "Texas Instruments" math input. Because we had to parse the form data anyway, we could use `[[` and `]]` as math delimiters instead of `$`. The conversion took place right after the form was received by the web server.

By combining Lua with TEX, we can do the conversion from calculator math to TEX immediately, without auxiliary programs or complex parsing using TEX macros.

### TEX

In a ConTEXt source one can use the `\calcmath` command, as in:

```
The strange formula
\calcmath{sqrt(sin^2(x)+cos^2(x))}
boils down to ...
```

One needs to load the module first, using:

```
\usemodule[calcmath]
```

Because the amount of code involved is rather small, eventually we may decide to add this support to the Mark IV kernel.

### XML

Coding math in TEX is rather efficient. In XML one needs way more code. Presentation MathML provides a few basic constructs and boils down to combining those building blocks. Content MathML is better, especially from the perspective of applications that need to interpret the formulas. It permits for instance the ConTEXt content MathML handler to adapt the rendering to cultural driven needs. The OpenMath way of coding is like content MathML, but more verbose with fewer tags. Calculator math is more restrictive than TEX math and less verbose than any of the XML variants. It looks like this:

```
<icm>sqrt(sin^2(x)+cos^2(x))</icm> test
```

And in display mode:

```
<dcm>sqrt(sin^2(x)+cos^2(x))</dcm> test
```

### Speed

This script (which you can find in the ConTEXt distribution as soon as the Mark IV code variants are added) is the first real TEX related Lua code that I've written; before this I had only written some wrapping and spell checking code for the SCITE editor. It also made a nice demo for a couple of talks that I held at usergroup meetings. The script has a lot of expressions. These convert one string into another. They are less powerful than regular expressions, but pretty fast and adequate. The feature I miss most is alternation like `(l|st)uck` but it's a small price to pay. As the Lua manual explains: adding a POSIX compliant regexp parser would take more lines of code than Lua currently does.

On my machine, running this first version took 3.5 seconds for typesetting 2500 times the previously shown square root of sine and cosine. Of this, 2.1 seconds were spent on typesetting and 1.4 seconds on converting. After optimizing the code, 0.8 seconds were used for conversion. A stand alone Lua takes .65 seconds, which includes loading the interpreter. On a test of 25 000 sample conversions, we could gain some 20% conversion time using the LuaJIT just in time compiler.

# Unicode and multilingual typesetting with XƎTEX

Jonathan Kew
SIL International
Horsleys Green
High Wycombe HP14 3XL
England
jonathan_kew (at) sil dot org

This extended abstract demonstrates how extending TeX to natively handle the Unicode character set greatly simplifies the task of multilingual and multi-script typesetting. Because all characters of all the world's scripts are included in a single standard, it is not necessary to convert external encodings to a special internal representation, or to manage multiple input encodings for different languages, and any combination of scripts and languages can be freely mixed in a single document — even in a single line of text. The XƎTEX extension of TeX makes it simple to use Unicode throughout, from input text to hyphenation tables and font access.

In addition to adopting Unicode as the standard character encoding, XƎTEX has built-in support for modern font technologies (TrueType, OpenType, AAT), including glyph layout behavior defined in font tables. This means that complex scripts such as Indic and Arabic can be typeset with no special font setup and configuration. For example, using an off-the-shelf Arabic font, whether from a major vendor or a free font developer, involves no complex conversion processes or the creation of an "alphabet soup" of `.tfm`, `.vf`, `.ocp`, `.map`, `.enc`, `.fd`, etc. files; just drop the `.otf` or `.ttf` file into the computer's Fonts directory, and select the typeface in a TeX document.

Including Arabic in a LaTeX document can then be as simple as declaring the font to be used:

```
\usepackage{fontspec}
\newfontinstance{\arfont}[Script=Arabic]
  {Scheherazade}
% for in-line Arabic we need R-L control
\newenvironment{ar}
  {\beginR\arfont}{\endR}
```

To include الخط العربي in a document, we can just enter `\begin{ar} ... \end{ar}` in the source text, with Unicode Arabic text within the `ar` environment (not shown here because `cmtt` does not include Arabic characters).

For extended passages of Arabic, one additional factor needs to be taken into account: the overall paragraph direction should be made right-to-left, so that the paragraph indent and alignment of the last line behave as expected:

```
% simple environment for R-L paragraphs
\newenvironment{ArabicPar}
  {\everypar={\setbox0\lastbox \beginR
   \box0 \arfont}}{}
```

لكل إنسان حق التمتع بكافة الحقوق والحريات الواردة في هذا الإعلان،
دون أي تمييز، كالتمييز بسبب العنصر أو اللون أو الجنس أو اللغة أو الدين أو الرأي
السياسي أو أي رأي آخر، أو الأصل الوطني أو الإجتماعي أو الثروة أو الميلاد أو أي
وضع آخر، دون أية تفرقة بين الرجال والنساء.

وفضلاً عما تقدم فلن يكون هناك أي تمييز أساسه الوضع السياسي أو القانوني
أو الدولي لبلد أو البقعة التي ينتمي إليها الفرد سواء كان هذا البلد أو تلك البقعة
مستقلاً أو تحت الوصاية أو غير متمتع بالحكم الذاتي أو كانت سيادته خاضعة لأي
قيد من القيود.

**Figure 1**: Arabic text typeset by XƎTEX using the minimal declarations shown in the text

This environment allows Arabic paragraphs to be properly laid out, as in figure 1.

Because XƎTEX uses Unicode text and fonts, rather than a complex collection of macros to provide the script support, it is trivial to include other scripts such as Japanese, Devanagari, or many others in the same document. All we need is an appropriate Unicode font that covers the required character repertoire:

```
% Japanese, with proper line-breaking
\newfontinstance{\japfont}
  {Hiragino Kaku Gothic Pro}
\newenvironment{Japanese}
  {\XeTeXlinebreaklocale "jp"
   \XeTeXlinebreakskip0pt plus 1pt
   \japfont}{}
% Hindi
\newfontinstance{\devfont}{Devanagari MT}
\newenvironment{Hindi}
  {\devfont}{}
```

With these declarations, we can set Japanese and Hindi just as easily as Arabic. Figure 2 shows two examples using fonts included as standard with

すべて人は、人種、皮膚の色、性、言語、宗教、政治上その他の意見、国民的もしくは社会的出身、財産、門地その他の地位又はこれに類するいかなる自由による差別をも受けることなく、この宣言に掲げるすべての権利と自由とを享有することができる。

さらに、個人の属する国又は地域が独立国であると、信託統治地域であると、非自治地域であると、又は他のなんらかの主権制限の下にあるとを問わず、その国又は地域の政治上、管轄上又は国際上の地位に基ずくいかなる差別もしてはならない。

सभी को इस घोषणा में सन्निहित सभी अधिकारों और आज़ादियों को प्राप्त करने का हक़ है और इस मामले में जाति, वर्ण, लिंग, भाषा, धर्म, राजनीति या अन्य विचार-प्रणाली, किसी देश या समाज विषेश में जन्म, सम्पत्ति या किसी प्रकार की अन्य मर्यादा आदि के कारण भेदभाव का विचार न किया जायेगा।

इसके अतिरिक्त, चाहे कोंई देश या प्रदेश स्वतन्द हो, संरक्षित हो, या स्वशासन रहित हो या परिमित प्रभुसत्ता वाला हो, उस देश या प्रदेश की राजनैतिक, क्षेत्रीय या अन्तर्राष्ट्रीय स्थिति के आधार पर वहां के निवासियों के प्रति कोई फ़रक़ न रहा जाएगा।

**Figure 2**: Japanese and Hindi text set by XeTeX

$$\left\{\begin{aligned}\alpha &= f(z)\\ \beta &= f(z^2)\\ \gamma &= f(z^3)\end{aligned}\right\} \qquad \left\{\begin{aligned}x &= \alpha^2 - \beta\\ y &= 2\gamma\end{aligned}\right\}$$

$$\left\{\begin{aligned}\alpha &= f(z)\\ \beta &= f(z^2)\\ \gamma &= f(z^3)\end{aligned}\right\} \qquad \left\{\begin{aligned}x &= \alpha^2 - \beta\\ y &= 2\gamma\end{aligned}\right\}$$

$$p_1(n) = \lim_{m\to\infty}\sum_{\nu=0}^{\infty}\bigl(1-\cos^{2m}(\nu!^n\pi/n)\bigr)$$

$$p_1(n) = \lim_{m\to\infty}\sum_{\nu=0}^{\infty}\bigl(1-\cos^{2m}(\nu!^n\pi/n)\bigr)$$

**Figure 3**: Math displays in Computer Modern (with custom encodings and multiple fonts) and Cambria Math (a single Unicode font, with no `.tfm`, etc.), typeset from the same source text

Mac OS X; similar results are obtained with OpenType fonts available on Windows, GNU/Linux, and other systems.

A more thorough implementation of script and language switching should of course also change hyphenation patterns, quote-mark styles, and other typographic niceties according to the language in use. These minimal examples show how easily multilingual fonts can be used; producing high-quality typography in varying scripts may require additional refinements.

Ongoing work on XeTeX includes some experimental features to support the use of OpenType math fonts, which can contain a huge collection of math alphabets (italic, bold, blackboard, fraktur, script, etc.) and symbols, all encoded according to the Unicode standard. Forthcoming Microsoft products will include the Cambria Math font, and other projects such as the STIX fonts can be expected to support the same OpenType standard for math metrics. XeTeX aims to be able to use such fonts directly, without needing to create custom-encoded subfonts, `.tfm` files, etc., and the current status of these features will be demonstrated. A couple of examples from *The TeXbook* are shown in figure 3, in both the original Computer Modern and Unicode-compliant Cambria Math fonts.

# The New Font Project: TeX Gyre

Hans Hagen
Pragma ADE, Holland
pragma (at) wxs dot nl

Jerzy B. Ludwichowski
Nicholas Copernicus University, Toruń, Poland
Jerzy.Ludwichowski (at) uni dot torun dot pl

Volker RW Schaa
GSI, Darmstadt, Germany
v.r.w.schaa (at) gsi dot de

http://www.gust.org.pl/e-foundry/tex-gyre

### Abstract

In this short presentation, we will introduce a new project: the "LM-ization" of the free fonts that come with TeX distributions. We will discuss the project objectives, timeline and cross-LUG funding aspects.

## 1 Introduction

The New Font Project is a brainchild of Hans Hagen, triggered mainly by the very good reception of the Latin Modern (LM) font project by the TeX community. After consulting other LUG leaders, Bogusław Jackowski and Janusz M. Nowacki, aka "GUST type.foundry", were asked to formulate the project.

The next section contains its outline, as prepared by Bogusław Jackowski and Janusz M. Nowacki. The remaining sections were written by us.

## 2 Project outline

Our aim is to prepare a family of fonts, equipped with a broad repertoire of Latin diacritical characters, based on the freely available good quality fonts. We think of an "LM-ization" of freely available fonts, i.e., providing about as many diacritical characters per font as we prepared for the Latin Modern font package (ca. 400 characters) which would cover all European languages as well as some non-European ones (Vietnamese, Navajo).

Since the provided character sets would be so close, such "LM-ized" fonts would work with all the TeX packages that the LM fonts work with, which would ease their integration. The result would be distributed, like the LM fonts, in the form of PostScript Type 1 fonts, OpenType fonts, MetaType1 sources and the supporting TeX machinery.

We would like to emphasize that the preparing of fonts in the OpenType format is an important aspect of the project. OpenType fonts are becoming more and more popular, they are Unicode-based, can be used on various platforms and claim to be a replacement for Type 1 and TrueType fonts. Moreover, Type 1 fonts were declared obsolete by Adobe a few years ago.

Since TFM format is restricted to 256 distinct character widths, it will still be necessary to prepare multiple metric and encoding files for each font. We look forward to an extended TFM format which will lift this restriction and, in conjunction with OpenType, simplify delivery and usage of fonts in TeX.

We especially look forward to assistance from pdfTeX users, because the pdfTeX team is working on the implementation on the support for OpenType fonts.

An important consideration from Hans Hagen: "In the end, even Ghostscript will benefit, so I can even imagine those fonts ending up in the Ghostscript distribution."

The idea of preparing such font families was suggested by the pdfTeX development team. Their proposal triggered a lively discussion by an informal group of representatives of several TeX user groups — notably Karl Berry (TUG), Hans Hagen (NTG), Jerzy Ludwichowski (GUST), Volker RW Schaa (DANTE) — who suggested that we should approach this project as a research, technical and implementation team, and promised their help in taking care of promotion, integration, supervising and financing.

The amount of time needed to carry out the task depends on the number of fonts to be included in the collection, but it can be safely estimated that a 4-font family (regular, italic, bold, and bold italic) can be prepared within 1–2 months, depending on the state of original material), which, for the collection of fonts mentioned below, would mean that the project can be accomplished within about two years. Assuming that the launch of the project could be made in the middle of 2006, the conclusion of the first stage of the project might be expected by the end of 2008.

The following fonts are presently considered worthy of enhancement:

1. The collection of 33 basic PostScript fonts, donated by URW++ and distributed with Ghostscript, consisting of eight 4-font families:
    - URW Gothic (i.e., Avant Garde)
    - URW Bookman
    - Century Schoolbook
    - Nimbus Sans (i.e., Helvetica)
    - Nimbus Sans Condensed (i.e., Helvetica condensed)
    - Nimbus Roman (i.e., Times)
    - Nimbus Mono (i.e., Courier)
    - URW Palladio (i.e., Palatino)

    and one single-font "family":
    - URW Chancery (i.e., Zapf Chancery)

    (Symbol and Zapf Dingbats fonts are left out.)
2. Donated by Bitstream (4-font families):
    - Charter
    - Vera
3. Donated by Adobe (4-font family):
    - Utopia
4. Other families donated by URW++:
    - Letter Gothic
    - URW Garamond

Perhaps other interesting free fonts will emerge in the future.

As to the budget — for the fonts listed above — there are altogether thirteen 4-font families and the 1-font URW Chancery. We would be satisfied if we could get a support of 1,500 EUR per a 4-font family and 500 EUR per a 1-font family. In the case of the fonts listed above this adds up to 20,000 EUR. We propose that the funding is made step-wise, i.e., the payments are made after the release of an enhanced family of fonts.

Obviously, there is scope for a second stage: the fonts can be further developed, as one can think for example of adding cap-small-caps, old style digits, proportional digits, and more. Once the first stage of the project is finished, we could embark on further enhancements. This would be somewhat simpler, but still a laborious task; we estimate the effort to be about 60–70% of the first stage.

## 3 Funding

The project can be divided into three stages:

- stage 1: combining existing fonts into LM compatible layouts and identifying gaps
- stage 2: filling in the gaps
- stage 3: math companion fonts

For these the estimated fundings needed are:

- stage 1: around 20,000 EUR
- stage 2: around 15,000 EUR
- stage 3: unknown

The exact figure for the second stage depends on what is needed and missing. We imagine the project to be extended with additional stages in order to bring more scripts into Open Type and/or to clean up other fonts as well.

The current financial state of the project:

- TUG India: 2,000 USD for the first year of the project and — if possible — a contribution for the next year.
- CSTUG: 500 EUR per year for the project duration, and expenses directly connected with implementing good Czech and Slovak support.
- NTG: 20% of the project costs, i.e., 4,000 EUR for the first stage.
- DANTE: 7,500 EUR this year; additional funding will be proposed at the September user group meeting.
- GUST: local infrastructure and expenses of font developers.
- TUG: positive but the amount is yet unknown, depends on contributions to the project fund, more info later this year.

Since the first stage runs this year and next year (start: May 2006) and since payments will take place after delivery of each completed font family, we can safely conclude that project expenses in 2006 can be covered and given the above we can also be confident that the rest of the first stage is secured.

The second stage is partly secured as well. We will try to broaden funding as soon and as much as possible. It would be good if this project can also take care of the needs of Indian, Greek, Cyrillic, Hebrew and Arab scripts, but we'll have to see ...

The suite of fonts will be officially presented on a special font CD for members of user groups.

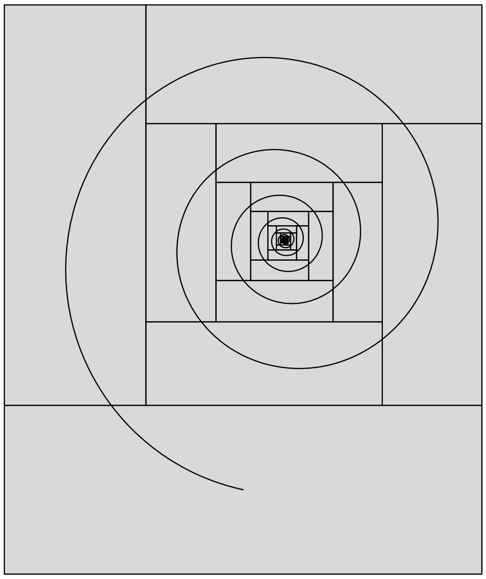


**Figure 1**: The TEX Gyre logotype

## 4 Co-ordination

This will done by DANTE e.V. in close co-operation with (at least) NTG, GUST, and TUG.

## 5 Late-breaking news: the name of the game

Nowhere above is explained the mysterious "TEX Gyre" which appears in the title. After this paper, consisting of the preceding sections, was presented at the BachoTEX 2006 conference, a discussion on how to name the collection of the New Font Project fonts was started by Bogusław Jackowski. A collective name for the project itself as well as names for the individual font families were looked for. In another lively mail exchange among the above people and others, between 23rd May 2006 and 5th June 2006, the final proposal was hammered out.

Some of the collective names proposed: TEX-Modern, TEXSurge, TEXFountain, TEXSuper, TEX Font Foundry, TEXelent, not to mention stranger concoctions for individual families like TEXOnitalap (Palatino reversed) or — for Bookman reversed — TEXMankoob. A minimalist proposal of TEXF prevailed for a while, then TEX Fountain Collection fought against TEX Fount and even TEXFun was proposed until TEX Gyre was coined by Karl Berry.

This was liked by Bogusław: it nicely plays with the logotype he used in his presentations of the LM project, Figure 1. It is meant to symbolize the neverending striving for perfection as well as the "LM-ization" of the families.

The reader might not have previously encountered the word "gyre". For pleasure, here are a few dictionary explanations and examples of use in literature and on the web.

- the Collins Dictionary of the English Language defines "gyre" as:
  - a circular or spiral movement or path
  - a ring, circle, or spiral
- the Webster's Third New International Dictionary, Unabridged, says the following:
  - to cause to turn around: REVOLVE, SPIN, WHIRL; to move in a circle or spiral
  - circular motion by a moving body: REVOLUTION; a circular or spiral form: RING, VORTEX
  - in Scottish: a malignant spirit or spook
- more entertaining nearby terms from Webster's:
  - *gyre carline*, in Scottish: WITCH, HAG
  - *gyrencephalate*: a group of higher mammals: having the surface of the brain convoluted
- Wikipedia (`http://en.wikipedia.org/wiki/Gyre`) reports this:
  - a gyre is any manner of swirling vortex
  - W. B. Yeats uses the word in many of his poems, including "The Second Coming"
  - Lewis Carroll used the word as a verb in the opening stanza of his poem "Jabberwocky":

    'Twas brillig, and the slithy toves<br>
    Did gyre and gimble in the wabe;<br>
    All mimsy were the borogoves,<br>
    And the mome raths outgrabe.

    defining it as "to go round and round like a gyroscope"
- "The Widening Gyre", home of slistuk, the UK survival listserver (`http://dnausers.d-n-a.net/dnetIULU`), describes itself as "A site for the preparedness minded"
- Gyre, The Old Sow Whirlpool (`http://www.oldsowwhirlpool.com`), the biggest whirlpool in the Western Hemisphere, of over 70 meters in diameter, on the border of Canada and the United States, on the east coast of North America.

We will spare the reader's patience by not presenting the details of the discussion leading to the names of the individual font families. The general motivation when trying to find the names was that people dealing with fonts are likely to be familiar with the original PostScript names, so names resembling the original ones were desired.

We first tried to find the shortest English nouns that have some relationship to the original PostScript names, with possibly positive or neutral connotations, but this proved futile. In the end, Latin words were adopted. The result is presented in the following table (check the meanings at, e.g., `http://archives.nd.edu/latgramm.htm`).

| **Original URW name** | **PostScript name**<br>**OpenType name**<br>**TFM name root***  |
|---|---|
| URW Gothic L | `TeXGyreAdventor`<br>`TeX Gyre Adventor`<br>`qag` |
| URW Bookman L | `TeXGyreBonum`<br>`TeX Gyre Bonum`<br>`qbk` |
| Nimbus Mono L | `TeXGyreCursor`<br>`TeX Gyre Cursor`<br>`qcr` |
| Nimbus Sans L | `TeXGyreHeros`<br>`TeX Gyre Heros`<br>`qhv` |
| URW Palladio L | `TeXGyrePagella`<br>`TeX Gyre Pagella`<br>`qpl` |
| Nimbus Roman No9 L | `TeXGyreTermes`<br>`TeX Gyre Termes`<br>`qtm` |
| Century Schoolbook L | `TeXGyreSchola`<br>`TeX Gyre Schola`<br>`qcs` |
| URW Chancery L | `TeXGyreChorus`<br>`TeX Gyre Chorus`<br>`qzc` |

* For example, TFM files for the members of the Pagella family are named `qplr.tfm, qplri.tfm, qplb.tfm, qplbi.tfm` for the regular, italic, bold and bold italic faces, respectively. Encodings will be specified as a prefix such as `ec-`, as in Latin Modern.

The families listed in items 2, 3 and 4 of Section 2 have not yet been given TeX Gyre names. We are going to continue in the same spirit.

At the time of this writing (November, 2006), the Pagella and Termes families are at version 1.00, the Bonum family is at version 0.995 and should very soon be fully released, i.e., arrive at version 1.00. They can be found at `http://www.gust.org.pl/e-foundry/tex-gyre`. There is more to the fonts than was promised in Section 2:

- over 1100 glyphs are available, including Cyrillic (though this was just carried over from the original without more ado);
- the complete Greek alphabet (for technical purposes rather than typesetting in Greek) is included — comments are welcome;
- cap-small-caps and old style digits are provided; this was initially planned for the second stage of the project.

Watch that space and enjoy!

P.S. XeTeX (`http://scripts.sil.org/xetex`), an addition to the TeX family developed relatively recently by Jonathan Kew, shows that the decision to provide TeX Gyre OpenType font versions was a good one: XeTeX already uses the OTF version of the LM fonts. Now also the OTF TeX Gyre fonts bring the advantages of this format to the TeX world . . .

## Jackowocky

’Twas brillig and the slithy Poles
Did GYRE and GIMBLE in the wabe
All mimsy were the borogoves
And the mome raths OUTGRABE

With apologies to Lewis Carroll (and Poland!)
— Chris Rowley

# Outline font extensions for Arabic typesetting

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague, Czech Republic
piska (at) fzu dot cz

**Abstract**

The contribution demonstrates applications of the programs FontForge (by George Williams) and MetaType1 (by Bogusław Jackowski et al.) for development and maintenance of outline versions of Arabic fonts and shows tools for glyph transformations. Generating of stretchable glyphs is also discussed. Building of Type 1 fonts for Arabic typesetting with MetaType1 (“LM-ization”) is under development process.

## 1 Introduction

The current text describes selected “technical techniques” of creating, modifying and maintaining outline fonts for use with Arabic typography.

We start with some font adaptations and modifications using FontForge for relatively simple glyph transformations. Executing more complex changes would be inefficient.

The next examples demonstrate representation of glyphs in MetaType1 (in fact, it is METAPOST), font modifications and some results of Type 1 fonts dynamically generated by MetaType1.

## 2 Font transformations with FontForge

The open source font editor FontForge [15], developed by George Williams, contains many commands for the creation and modification of fonts in numerous standard formats. Along with its interactive facilities, FontForge has a scripting language which allows automatic batch processing. Thus, existing fonts, e.g., Computer Modern Type 1 mathematical fonts, can be adapted into fonts oriented to Arabic presentation by executing appropriate glyph transformation commands. The example in fig. 1 demonstrates how to flip (reverse, mirror) a symbol to achieve the effect described in the paper on ‘Dynamic Arabic Mathematical Fonts’ ([11], fig. 1) in our Type 1 representation. The original symbol (left) has been flipped horizontally (middle) and then vertically (right):

```
#!/usr/local/bin/fontforge
Open("cmex10.pfb");
 Select("summationtext");
 HFlip(CharInfo("Width")/2); VFlip();
 CorrectDirection();
 SetFontNames("amcmex10",\
 "ArabicComputerModern",\
 "ArabicMathCMEX10");
Generate("amcmex10.pfb","",0x240000);
# no flex hints, hints, round,
# no afm, no tfm
```

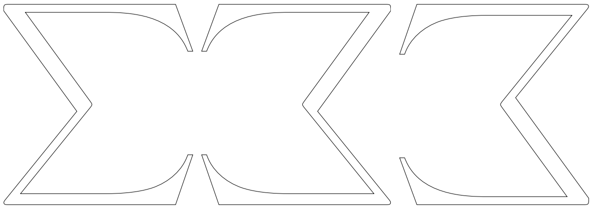


**Figure 1**: Flip twice or rotate.

The identical transformation could be done by rotation:

```
 Select("summationtext");
 Rotate(180);
```

## 3 Font development with MetaType1

MetaType1 [6], developed by Bogusław Jackowski, Janusz M. Nowacki, and Piotr Strzelczyk, is a METAPOST-based package for producing, auditing, enhancing and otherwise handling outline PostScript fonts in the Type 1 format. One part of the package is a converter from Type 1 to MetaType1 source. A practical approach to maintaining a font with MetaType1 is to start from an existing Type 1 font or from a font just converted into Type 1 (e.g., from METAFONT sources). In the present case for Arabic we can and *want* to use as a base (for further modifications, extensions, etc.) the `xnsh14` font in the Naskhi style available from the ArabTEX distribution [9].

The most important ideas of the MetaType1 package are:

- programmable font description in source form, (the language is METAPOST with extensions for font support);
- the glyphs are defined in their outline representation (the recommended approach);
- very simple definition of composite glyphs, especially glyphs with accents, which is significant for all Latin fonts;
- automatic generation of glyph and metric files (`pfb, afm, tfm, pfm`) from scratch;
- the glyphs are (may be and must be) denoted by (PostScript) names, therefore the number of glyphs is not limited, although the Type 1, `tfm` output and encodings are restricted (no more than 256 encoded characters available);
- the glyph definitions also contain metric data: dimensions (width, height, depth, italic correction), ligatures, kerning pairs and other information, to allow for easier further conversion into OpenType or Type 3.

These features have been applied during the development of the Latin Modern collection [7] and other fonts produced by the authors. Similar "LMization" could be executed for Arabic because there is no fundamental difference between Latin accents and Arabic diacritic marks.

The following example and fig. 2 illustrate producing composite Arabic glyphs with MetaType1.

```
def mark_down(text glyh_acc_,glyh_,acc_) =
standard_introduce(glyh_acc_);
beginglyph(glyh_acc_);
 use_glyph(glyh_);
 use_glyph(acc_) % offset of the accent =
 (round((wd.uni_name(glyh_)-wd.uni_name(acc_))/2),
 dp.uni_name(glyh_)-ht.uni_name(acc_));
% recalculation of metrics
 wd.uni_name(glyh_acc_)=wd.uni_name(glyh_);
 ht.uni_name(glyh_acc_)=ht.uni_name(glyh_);
 dp.uni_name(glyh_acc_)=dp.uni_name(glyh_)
 -ht.uni_name(acc_)+dp.uni_name(acc_);
 fix_hsbw(wd.uni_name(glyh_acc_),0,0);
endglyph;
enddef;

mark_down("bah")("bah_s")("one_dot_down");
mark_down("pah")("bah_s")("three_dots_down");
% definition of "mark_up" macro is similar
mark_up("tah")("bah_s")("two_dots_up");
```

The next example shows an excerpt from the Type 1 representation in the readable form disassembled by `t1disasm` from the `t1utils` package. The dot mark in `nun` is omitted.

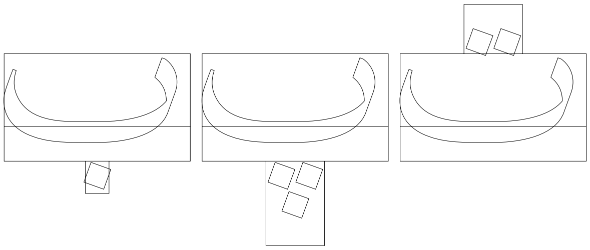


**Figure 2**: Composite glyphs: b — p — t.

```
/nun.fin {
        0 433 hsbw
        -278 70 hstem
        0 71 hstem
        0 35 vstem
        356 36 vstem
        451 67 rmoveto
        -11 4 rlineto
        -27 0 -19 13 -22 22 rrcurveto
        -11 4 rlineto
        -24 -67 rlineto
        13 -38 6 -40 0 -40 rrcurveto
        0 -13 -1 -12 -2 -13 rrcurveto
        -49 -84 -78 -11 -34 0 rrcurveto
[...]
        closepath
        endchar
} ND
```

The result of conversion into the METAPOST/ MetaType1 representation (in an absolute coordinate system!) is:

```
beginglyph(_nun.fin);
 save p; path p[];

 z0 0=(451,67);
 z0 1=(440,71); z0 1a=(413,71); z0 2b=(394,84);
 z0 2=(372,106);
 z0 3=(361,110);
 z0 4=(337,43); z0 4a=(350,5); z0 5b=(356,-35);
 z0 5=(356,-75); z0 5a=(356,-88); z0 6b=(355,-100);
 z0 6=(353,-113); z0 6a=(304,-197); z0 7b=(226,-208);
[...]
 z0 21=(426,0);
 p0=compose_path.z0(21); Fill p0;
 fix_hstem(71)(p0) candidate_list(y)(0, 71);
[...]
 standard_exact_hsbw("nun.fin");
endglyph;
```

And my subsequent conversion of the path definition into relative coordinates, as it will be necessary to eliminate the dependence on absolute coordinate values:

```
def nunf_z(suffix nz) =
z.nz 0=(451,67);
z.nz 1=z.nz 0+(-11,4);
z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
z.nz 2=z.nz 2b+(-23,23);
[...]
z.nz 7=z.nz 7b+(-34,0);
z.nz 7a=z.nz 7+(-61,0);
z.nz 8b=z.nz 7a+(-70,35);
[...]
z.nz 15=z.nz 15b+(111,0);
z.nz 15a=z.nz 15+(111,0);
```

```
[...]
z.nz 22=z.nz 0;
enddef;
```

This path has been slightly modified and will be further adapted in an extended stretchable definition of the letter `nun` in the final position (without the dot) — see later.

The METAPOST macro definitions may define glyphs or their parts, and we can use them to describe glyphs already present in a font or to compose modified or new glyphs.

The transformations we saw in fig. 1 can be expressed in METAPOST to produce the same results:

```
 numeric l,d; l:=wd._summationtext;
  d=dp._summationtext;

%  Horizontal Flip:
 p0=compose_path.z0(22)
  reflectedabout((l/2,0),(l/2,1));
 correct_path_directions(p0)(p);

%  Horizontal and Vertical Flip:
 p1=compose_path.z0(22)
  reflectedabout((l/2,0),(l/2,1))
  reflectedabout((0,d/2),(1,d/2));
 correct_path_directions(p1)(p);

%  Rotation
 p2=compose_path.z0(22)
  rotated 180 shifted (l,d);
```

## 4 Glyph stretching with MetaType1

Azzeddine Lazrek et al., in the papers about typesetting Arabic (RyDArab [12] and CurExt packages [10] and dynamic fonts [11]), describe the use of dynamic Type 3 fonts and corresponding `tfm`, `map` and `enc` files for generating final PostScript documents with (LA)TEX and `dvips`. The supported version of MetaType1 supports producing only Type 1 fonts, and Type 1 (and also metric) files cannot be dynamic.

But `tfm` (as the RyDArab system does) and Type 1 can be generated dynamically. As the next consecutive step after generating metrics we can produce (dynamically with MetaType1) the Type 1 font corresponding to the equivalent Type 3 font and substituting it.

The variable-width kashida is demonstrated in fig. 3. The Type 3 commands from [11] were rewritten into METAPOST macros giving the similar commands in Type 1 where, of course, each glyph should have its own charstring definition. Glyphs, like metrics, can be generated on the fly for a given width.

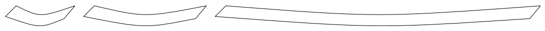


**Figure 3**: Stretchable kashida in Type 1.

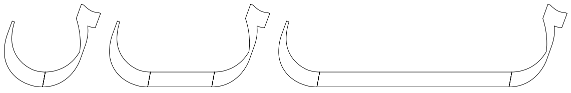


**Figure 4**: Stretchable glyph with an hrule filler.

Fig. 4 shows a primitive glyph elongation where only a horizontal rule as the filler is inserted and the right and left part of the glyph `nun` in the final position (without dot) are shifted.

The following METAPOST commands and fig. 5 demonstrate a more sophisticated elongation algorithm: the parameter *addwx* changes control points and control vectors. The solution was inspired by methods described by Daniel M. Berry [1]. Here we have decided to preserve glyph heights and also their right and left parts. Probably more complex outline contour curves could be defined.

```
def nunf_z(suffix nz)(expr addwx) =
addwxa:=round(addwx/2);
z.nz 0=(451,67)+(addwx,0);
z.nz 1=z.nz 0+(-11,4);
z.nz 1a=z.nz 1+(-28,0); z.nz 2b=z.nz 1a+(-17,12);
[...]
z.nz 7=z.nz 7b+(-34,0)-(addwxa,0);
z.nz 7a=z.nz 7+(-61,0)-(addwxa,0);
z.nz 8b=z.nz 7a+(-70,35);
[...]
z.nz 15=z.nz 15b+(111,0)+(addwxa,0);
z.nz 15a=z.nz 15+(111,0)+(addwxa,0);
[...]
z.nz 22=z.nz 0;
enddef;

def nun_fin_v(suffix code)(expr addx) =
 standard_introduce("nun.fin_v" & decimal(code));
 wd._nun.fin_v.code:=wd._nun.fin+addx;
 ht._nun.fin_v.code:=ht._nun.fin;
 dp._nun.fin_v.code:=dp._nun.fin;
 beginglyph(_nun.fin_v.code);
  save p; path p[];
  nunf_z(0)(addx); p0=compose_path.z0(21);
  Fill p0;
  standard_exact_hsbw("nun.fin_v" & decimal(code));
 endglyph;
enddef;
nun_fin_v(300)(+300);
```

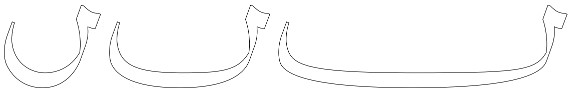


**Figure 5**: Stretchable glyph.

D. Berry [1] and A. Lazrek [11] propose to use PostScript Type 3 fonts and W. Bzyl [2, 3] reintroduced Type 3 fonts and showed how to extend the MetaType1 package to produce Type 3 (but it did not find the support in recent MetaType1 distributions). For me the future of Type 3 fonts is not clear, screen rendering algorithms for Type 3 are worse than for other font formats, probably nobody is going to improve them, and I do not include Type 3 in my contribution.

## 5 Conclusion

I have no detailed information about commercial and copyrighted products of Thomas Milo [13] (and I do not have these products). I expect the development and use of Arabic fonts will be discussed with authors of packages for multilingual typesetting including Arabic: Y. Haralambous [14], H. Fahmy [4], K. Lagally [9], J. Kew [8], A. Lazrek, and others. I have no support for integrating dynamically generated or stretchable fonts into TEX. Some new line breaking and justification algorithms could be developed, for example, to spread the word box to a specified width and then to generate dynamically the appropriate glyph instance by demand on the fly to composite a compound "ligature". Or to produce glyphs only in a restricted set of point sizes and apply some variant of a micro-typographic alignment or justification as in pdfTEX [5].

This approach uses "small" (max. 256 glyphs) `tfm` and `pfb` files. We could convert them into OpenType (as the LM fonts have been converted). But I do not know: "Could we integrate Type 3 into OpenType?" or "Is it possible to create dynamic OpenType?"

A limitation to one direction, the current trend towards a single, huge and static outline OpenType font file (for each typeface in important point sizes), may not be wise. It's unlikely this will be the final termination point of font development, and thus will not be the best solution in the future development of computer font technology.

## References

[1] Daniel M. Berry. Stretching Letter and Slanted-baseline Formatting for Arabic, Hebrew, and Persian with ditroff/ffortid and Dynamic PostScript Fonts. *Software—Practice & Experience*, 29(15), 1417–1457, 1999.

[2] Włodzimierz Bzyl. Reintroducing Type 3 fonts to the world of TEX. *Proceedings of the XII European TEX Conference*, pp. 219–243, Kerkrade, the Netherlands, 23–27 September 2001, 2001.

[3] Włodzimierz Bzyl. The Tao of Fonts, *TUGboat* 23(1):27–40, 2002.

[4] Hossam A. H. Fahmy. AlQalam for typesetting traditional Arabic texts. In this volume, pp. 159–166.

[5] Hàn Thế Thành. Micro-typographic extensions to the TEX typesetting system. *TUGboat* 21(4), 317–434, 2000.

[6] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):575–581, 2003; `ftp://bop.eps.gda.pl/pub/metatype1`.

[7] Bogusław Jackowski, Janusz M. Nowacki. Enhancing Computer Modern with accents, accents, accents. *TUGboat* 24(1):64–74, 2003; `CTAN:/fonts/lm`.

[8] Jonathan Kew. XƎTEX, the Multilingual Lion: TEX meets Unicode and smart font technologies. *TUG 2005 Conference Proceedings*, *TUGboat* 26(2):115–124, 2005.

[9] Klaus Lagally. ArabTEX — Typesetting Arabic with vowels and ligatures. *EuroTEX 92: Proceedings of the 7th European TEX Conference*, ed. J. Zlatuška, pp. 152–172, Brno, Czechoslovakia, 1992.

[10] Azzeddine Lazrek. CurExt, typesetting variable-sized curved symbols. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):323–327, 2003.

[11] Mostafa Banouni, Mohamed Elyaakoubi, and Azzeddine Lazrek. Dynamic Arabic mathematical fonts. *Preprints for the 2004 Annual Meeting*, Xanthi, Greece, pp. 48–53, 2004.

[12] Azzeddine Lazrek. RyDArab — Typesetting Arabic mathematical expressions. *TUGboat* 25(2):141–149, 2004.

[13] Thomas Milo. ALI-BABA and the 4.0 Unicode characters — Towards the ideal Arabic working environment. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):502–511, 2003.

[14] Yannis Haralambous and John Plaice. Multilingual Typesetting with Ω, a Case Study: Arabic. *Proceedings of the International Symposium on Multilingual Information Processing*, pp. 137–154, Tsukuba, 1997.

[15] George Williams. Font creation with FontForge. *EuroTEX 2003 Proceedings*, *TUGboat* 24(3):531–544, 2003; `http://fontforge.sourceforge.net`.

# Arabic font building for LaTeX

F. Mounayerji, M. A. Naal
Department of Computer Engineering
University of Aleppo, Syria
`Fares_Mounayerji (at) hotmail dot com`

### Abstract

This contribution aims to describe a new solution for building arabic font for LaTeX. We focus on the font generation for Arabic calligraphy. This solution is based on the determination of control points that gives precise METAFONT code for the given Arabic font glyph. Using the METAFONT compiler, the new font is compiled and finally installed on a LaTeX distribution.

## 1 Introduction

The Arabic language is one of the ten most commonly used languages worldwide. Over 300,000,000 people living in the Arab world use this language in everyday and official writing. This creates an important potential for Arabic text editor users.

LaTeX is an elegant and advantageous typesetting program and is strongly recommended for scientific writing. Its capabilities and especially its mathematical capabilities are well known.

LaTeX uses a logical structure or WYMIWYG (What You Mean Is What You Get) concept instead of WYSIWYG (What You See Is What You Get), which makes LaTeX unique in its approach for building texts, and building structure-oriented rather than formatting-oriented text, which reduces errors and increases concentration on the idea of the text.

There is a growing interest in globalizing LaTeX by supporting the most important languages all over the world, and as we know, the Arabic language is important and widely used, which generates a need for even more extensive support than that of the existing ArabTeX and other Arabic support packages. We also need to give the Arabic user of LaTeX more options, such as font selection, font adding, special Arabic effects, etc.

Because of all the previous factors mentioned, we see the importance to expand the support for this important language.

## 2 LaTeX Principles

### 2.1 What is LaTeX?

LaTeX is a programming language used to build large text documents (e.g., books, articles, theses) presentations, etc. We can also use the term LaTeX for a LaTeX compiler. Figure 1 shows a simplified view of the structure of LaTeX processing.

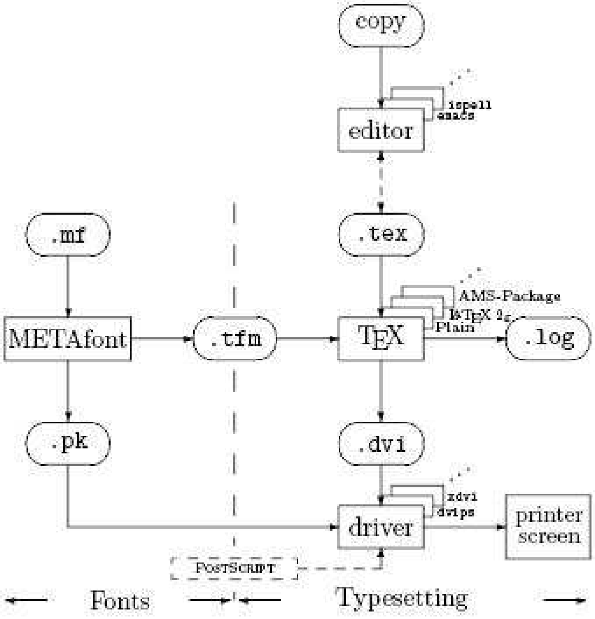


**Figure 1**: Simplified LaTeX system structure

### 2.2 What is a LaTeX distribution?

A (La)TeX distribution is a set of folders and files containing a (La)TeX compiler, in addition to supporting tools such as a previewer, METAFONT compiler, and much more. Some LaTeX distribution is needed to run LaTeX on a given machine; optionally, a front-end like TeXnicCenter or WinEdt can also be used. LaTeX is free and open source software, which has led to an enormous number of distributions, but only a few of them are recognized worldwide, such as MiKTeX on Microsoft Windows, MacTeX on Mac OS X, and teTeX which is usually installed by default on Linux platforms.

Standard distributions follow the TDS (TeX Directory Structure) conventions, which describe how to organize a distribution's files and folders, and in relation to our main subject of font files, the TDS specifies where to put the font files such as `.pk`, `.tfm` and `.mf`.

### 2.3 METAFONT

METAFONT is a font building programing language used to build fonts for (LA)TeX. It has a special syntax to do this, depending on the control points of the letter and their ordering. A line is drawn passing between the control points, sometimes a straight line, and sometimes a curved line [5].

## 3 The solution

Because of the importance of the Arabic language, Arabic fonts are important too. Every user should have the ability to design the font that he wants to use with his distribution of LaTeX. To achieve this goal we built a solution that permits any character, or actually any shape, to be processed in different stages in order to achieve the font as the user wants it to be. The main stages in achieving this goal are:

- drawing the font on paper,
- scanning the font as an image,
- analyzing the image,
- finding the relations between control points,
- and generating the METAFONTcode.

We will talk about each of these stages in detail in the following sections.

### 3.1 Drawing the font on paper

This step is drawing the Arabic character like و, ص or any other characters on paper, respecting Arabic calligraphy. We don't need to draw the double-struck letter forms because there is an algorithm to extract it from the standard form.[1]

### 3.2 Scanning the font

This will be a simple procedure of scanning the font in order to enter it into the computer machine, and deal with it as a simple colored image.[2] We also include applying some changes to the image to be ready for the next step.

**First stage** Transform the image into a grayscale image, and then into a black and white image.

**Next stage** Applying certain algorithms in order to prepare the image for determining the control points, such as the contour algorithm for the double-struck letter, and the skeleton algorithm for the standard form, as we can see in Figure 2.

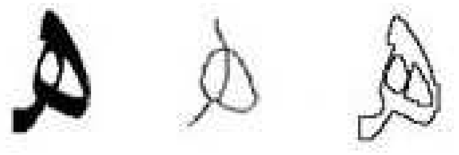


**Figure 2**: Standard, skeleton and contour

### 3.3 Analyzing the image

This step will deal with the output image of the previous step, and it will be responsible for two main objectives:

- Determining the control points of the letter as precisely as possible. Since these points will be used later to build the METAFONT code, this procedure is the main work here. We will scan the image line by line until we find a black point; this point will be the start of the letter (generally but not always — the font could be composed of lines and curves, separated or connected). After that, we will move on those curves and lines point by point, and during this pass we will save the information that we get about them in matrices, in order to be used in a later stage for building the font as it appears in the image.
- Determining the path between those points is very important here because we will lose the original shape of the letter without knowing the exact path between its control points.[3]

Figure 3 shows how we handle the standard case by using the skeleton algorithm, and Figure 4 shows how we handle the contour case or the double-struck character.

### 3.4 Finding the relations between the control points

This is an important step for simplifying the METAFONT code, because many relations between the control points could be found. Simplifying will make the

[1] It is worth mentioning that we can build our font as a dynamic font [2] by using the kashida, a curvilinear variable lengthening of letters along the baseline.

[2] Usually black and white, with the font being black and the background white.

[3] The control points cannot be repeated but the path points can be repeated.

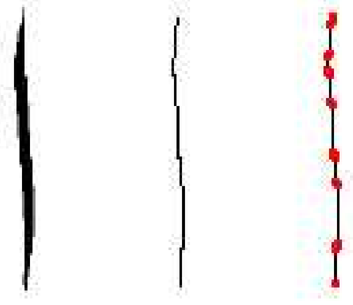


**Figure 3**: Standard, skeleton and control points

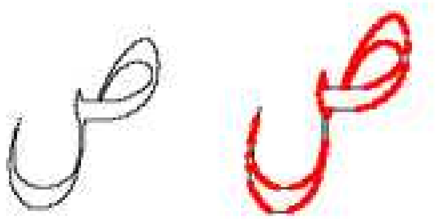


**Figure 4**: Double-struck and control points

work for building the METAFONT file easier, and the result more professional. Those relations can vary from two points or more which have the same $x$ or $y$, to two points have a point which fall between them in three cases:

- $x_3 = (x_1 + x_2)/2$
- $y_3 = (y_1 + y_2)/2$
- $z_3 = (z_1 + z_2)/2$

### 3.5 Generating the METAFONT code

This step will include transforming the results of the previous steps into code readable by the METAFONT compiler. First of all we need to transform the Cartesian coordinates of the solution environment into the Cartesian coordinates of the METAFONTcompiler. Secondly we will have to transform the points of the path into the METAFONT form. And finally add the METAFONT instructions that will draw the lines between the previous points.

### 3.6 Installing the new font

This step actually depends on the particular (LA)TEX distribution, but we can specify the main stages in a general way [1]. Assuming that the METAFONT file is `arab.mf`:

- From the command line, run

```
mf '\mode=ljfour; mode_setup; input arab.mf'
```

- The output files will be `arab.tfm` and `arab.600gf` (or similar).[4]
- Now you still need to pack the `gf` file into a usable format. You do so with this command:

```
gftopk arab.600gf arab.pk
```

- And that's it! Your font is now available to every LATEX document whose source is in the same directory as the font files. I mean here both file with the extensions `.pk` and `.tfm`. We can also install it permanently, which is a process that depends on the distribution [3].

## References

[1] Christophe Grandsire, *The METAFONT tutorial*, Version 0.33.

[2] Mostafa Banouni, Mohamed Elyaakoubi and Azzeddine Lazrek, Dynamic Arabic Mathematical Fonts. *Preprints for the 2004 Annual Meeting*, Xanthi, Greece, pp. 48–53, 2004.

[3] TUG Working Group on a TEX Directory Structure (TWG-TDS), *A Directory Structure for TEX Files*, version 1.1, June 23, 2004.

[4] Azzeddine Lazrek, *NasX Arabic literal symbols font*, May 2, 2004.

[5] Peter Flynn, *Formatting information: A beginner's introduction to typesetting with LATEX*, 2005.

[6] *The UK TEX FAQ, Your 396 Questions Answered*, version 3.15a, date 2005/11/29.

[7] Jeff Clark, *LATEX Tutorial*, revised February 26, 2002.

[8] Klaus Lagally, *ArabTEX, Typesetting Arabic and Hebrew, User Manual*, Version 4.002, March 11, 2004.

[4] The number 600 indicates that the precision is 600 dpi (dots per inch).

# Everything we want to know about Font Resources

Chris Rowley
Open University, UK
c dot a dot rowley (at) open dot ac dot uk

**Abstract**

A brief discussion of Font Resources: subsystems that know how to 'visualize' a string of (Unicode) characters. We mention existing Font Resources, desired capabilities for LaTeX, and questions for further study.

## 1 Introduction

This article is based on my introduction to a discussion of font resources and their use that took place towards the end of the TUG 2006 conference. By that time, a lot of useful detailed information (much of it correct!) about modern font resources had been provided, both formally and informally, to me by many delegates (whilst others, naturally, assumed that we all knew everything before they started talking).

I therefore decided to focus on more general ideas about the philosophy behind the new technology and how this fits with the classical approach of TeX-based typesetters.

The bibliography points to a few of the many relevant web resources.

## 2 LaTeX-related work

There are many specific questions to be asked and decisions to be made before building a LaTeX [4] system for automated selection from the available choices offered by modern font resources, due to the large range of variants within what was once a single font. We shall also need to extend the font-changing commands; work on glyph selection in math mode has been started already. Most of these ideas are being pioneered by Will Robertson in his work with XeTeX [8], particularly the fontspec [3] package.

## 3 What is a Font Resource?

For current purposes I shall use the following informal answer. A Font Resource is a subsystem that 'knows how to visualize a string of (Unicode) characters'. Note that this is not quite the same as saying that a font resource is itself capable of rendering the character string; thus, for example, a TeX `.tfm` file is a Font Resource although it contains only metric information, not rendering information.

So is that all a Font Resource can do? And must it be able to visualise *any* text string? These are two questions that do not need precise answers, I pose them only to get you thinking!

An important extra property of any Font Resource that can be really useful to a typesetter (either human or software) is flexibility and the major advantage of modern fonts is that they have a lot more abilities than classical fonts. Equally important for flexible Font Resources is that they should be self-aware: they need to know about these abilities and how much flexibility they provide.

## 4 What can we ask of a Font Resource?

A flexible Font Resource needs to be self-aware so that the font selection system of a typesetter (such as LaTeX's NFSS) can ask it about its abilities. Typical modern fonts can be queried in this way by typesetting software such as XeTeX (although the interface is not very intuitive), so it will be possible to extend the NFSS to exploit their added flexibility.

## 5 Exemplary types of Font Resource

This is not a classification of types of Font Resource but merely some more-or-less mythical examples of the range of possibilities that are now, or maybe soon, available.

**simple** just stacks aligned glyphs in one writing direction (maybe 'returns the advance width')

**TeX tfm** … adds (a fixed set of)

- kerns
- ligatures (mandatory, aesthetic, Knuthian)
- italic corrections

**AAT/OpenType** … adds (the possibility of)

- (importantly) script plus language
- (usefully) choice of ligatures
- (for fun fonts) choice of swashes, etc.
- … many other things (see Apple Advanced Typography [1] and the Microsoft/Adobe version [5])

**FreeType 2** … use various font resource types [2]

**$\Omega_2$** … adds many more features (see the article by Yannis Haralambous in these proceedings)

**ParaType** … adds line-breaking and justification (perhaps?)

## 6 The discussion

At this point some well-known gurus were dragged onto the platform to answer questions and provoke further discussion. We started off with some rhetorical questions that allowed the panelists to remove many of my misunderstandings. Particular thanks are due to Yannis Haralambous, Taco Hoekwater, Jonathan Kew & Arthur Reutenauer.

The questions:

- What is a Font Resource?
- Should there be a clear interface between Font Resource and the Typesetter?

The discussion ranged far and wide, drifting into political as well as technical areas. Here are some highlights.

- Clarification of the various parts of current font technology and their relationship to aspects of micro-formatting — including paragraph making!
- Font Resources may do a lot ... and the typesetter may need control too; BUT there is *no* clear division between them.
- Currently, middleware is important for typesetting, especially for complex scripts: examples are Pango [6] and Uniscribe [7].
- IMPORTANT: a modern typesetter should be able to use whatever Font Resources are available and not need specialised formats such as `.tfm` files.

## 7 The questions continue

Here are some further questions which are still open. I hope they will stimulate further discoveries leading to articles in *TUGboat*.

### 7.1 Questions for a Font Resource

- What can be asked about a visualisation?
- What can be asked about the abilities?
- How practical is it to query the tables in a modern Font Resource?
  - Is it inefficient?
  - Can all information be extracted?

### 7.2 Handling deficiencies in a Font Resource

How should a typesetting system best handle a modern font resource that does not explicitly provide all the information required by a typesetter (such as accurate vertical metrics or italic corrections)? Some possibilities, assuming that the extra information can be calculated or found elsewhere:

- Add to existing tables using standard methods: do these exist?
- Add extra tables in the Font Resource: is this feasible?
- Produce external tables that enhance/override internals: is this feasible?

Such activities also raise legal and moral questions about the licencing terms of modern font resources: should there be a distinction between the rendering information (the glyphs themselves), the metric information and the 'use information' (i.e., whether to use ligatures and other features).

Finally some questions for (or demands of) font designers:

- What tables are needed for high quality typesetting?
- How should the glyphs, metrics and other information be enhanced/extended/corrected?

## References

[1] `http://www.apple.com/macosx/features/fontbook`

[2] `http://www.freetype.org`

[3] `http://www.ctan.org/tex-archive/macros/xetex/latex/fontspec`

[4] `http://www.latex-project.org`

[5] `http://www.adobe.com/uk/type/opentype`

[6] `http://www.pango.org`

[7] `http://www.microsoft.com/typography/developers/uniscribe`

[8] `http://scripts.sil.org/xetex`

# Names in BibTeX and MlBibTeX

Jean-Michel Hufflen
LIFC (FRE CNRS 2661)
University of Franche-Comté
16, route de Gray
25030 Besançon Cedex
France
`hufflen (at) lifc.univ-fcomte.fr`
`http://lifc.univ-fcomte.fr/~hufflen`

## Abstract

Within the bibliographical entries managed by BibTeX, the bibliography processor usually associated with LaTeX, person and organisation names are specified with a rough syntax, whose details are not very well known. Likewise, the features related to formatting names within bibliography styles are often viewed as obscure. We explain these points in detail, giving some cases difficult or impossible to handle with BibTeX. Then we show how these problems can be solved within MlBibTeX, our reimplementation of BibTeX focusing on multilingual features and using an extension of XSLT as the language for bibliography styles.

**Keywords** BibTeX, MlBibTeX, Bibliographies, bibliography styles, specifying and formatting person names.

## 1 Introduction

Specifying meta-information related to persons is a difficult problem within databases from a general point of view. It is well known that the parts constituting the name of a person are insufficient to characterise only one person. However, we do not go thoroughly into this point and only focus on names we can find within bibliographical databases. Managing this information is crucial: it may be used for searching bibliographical databases, and many bibliography styles sort references w.r.t. alphabetical order of authors or editors.

When users typeset documents with the LaTeX word processor [15], searching bibliographical databases for citations and assembling references into a 'Bibliography' section, put at the end of a printed document, is usually done with the BibTeX bibliography processor [20]. Formats for names are defined as part of bibliography styles: first names may be abbreviated or put *in extenso*, they can be written before or after the last name, ... BibTeX uses a rough syntax for specifying authors' and editors' names within bibliographical entries. This syntax is suitable for simple cases, and powerful since it includes many interesting and useful features, sometimes not well known due to their complexity. In addition, some of these features are documented only partially, as far as we know, although many details are given in the second edition of the *LaTeX Companion* [16, § 13.2.2]. Likewise, the primitive `format.name$` function, which formats names within BibTeX's bibliography styles [19] is powerful but uses quite complicated patterns, documented only partially. In fact, only a small fraction of its expressive power is used in practice.

So we begin by describing BibTeX's syntax for names, with its advantages and limitations. We also show how names are formatted, as precisely as possible. Then we explain how these points have been improved within MlBibTeX (for '**M**ulti**L**ingual BibTeX) [7], our reimplementation focusing on multilingual aspects. Last, we discuss aspects of internationalization.

Reading this article requires good knowledge of BibTeX as an end-user, and a bit of experience with bibliography styles, that is, a good knowledge of the notions described in [16, § 13.6]. We will recapitulate more technical points, however.

## 2 How names are processed by BibTeX

### 2.1 Names' components

Inside `AUTHOR` and `EDITOR` fields, BibTeX allows the specification of successive names, separated by the '`and`' keyword, as shown in Figure 1. A large list of names that is not typed *in extenso* is ended with '`and others`':

```
{Karl-Heinz Scheer and Clark Darlton and
 others}
```

```
@BOOK{feist-wurts1991,
       AUTHOR = {Raymond E. Feist and
                 Janny Wurts},
       TITLE = {Servant of the Empire},
       PUBLISHER = {Grafton Books},
       ADDRESS = {London},
       YEAR = 1991}
```

**Figure 1**: Example of a BibTeX bibliographical entry.

A name consists of four components: *First* (for a first name), *von* (for a particle), *Last* (for a last name), and *Junior* (for a suffix), and recognizes them in the following possible syntaxes [20, § 4]:

(i) *First von Last*

(ii) *von Last*, *First*

(iii) *von Last*, *Junior*, *First*

As suggested by the word capitalisation used within this terminology — originating from BibTeX — the words belonging to the *von* field are supposed to begin with a lowercase character, whereas the words belonging to the *First* and *Last* fields are supposed to begin with an uppercase character, e.g.:

$$\underbrace{\texttt{Catherine Crook}}_{\textit{First}}\ \underbrace{\texttt{de}}_{\textit{von}}\ \underbrace{\texttt{Camp}}_{\textit{Last}}$$

The rule common to these three syntaxes: if there is only one word, it is taken as the *Last* part, even if this word does not begin with an uppercase letter, e.g.:

$$\underbrace{\texttt{\{-\}ky}}_{\textit{Last}}$$

If we consider the (i) syntax, two other rules are used when a name is split into its components:

- the *von* part takes as many words as possible, provided that its first and last words begin with a lowercase letter, e.g.:[1]

  $$\underbrace{\texttt{Jean}}_{\textit{First}}\ \underbrace{\texttt{de la Fontaine du}}_{\textit{von}}\ \underbrace{\texttt{Bois Joli}}_{\textit{Last}}$$

  let us notice that the *First* part can be empty, whereas the *Last* part cannot:

  $$\underbrace{\texttt{jean de la fontaine du bois}}_{\textit{von}}\ \underbrace{\texttt{joli}}_{\textit{Last}}$$

- if all the words begin with an uppercase letter, the last word is the *Last* component, and the *First* part groups the other words, e.g.:

  $$\underbrace{\texttt{Kim Stanley}}_{\textit{First}}\ \underbrace{\texttt{Robinson}}_{\textit{Last}}$$

If we consider the (ii) or (iii) syntaxes, the *von* part takes as many words as possible, provided that only its last word begins with a lowercase letter, e.g.:

$$\underbrace{\texttt{De la Fontaine du}}_{\textit{von}}\ \underbrace{\texttt{Bois Joli}}_{\textit{Last}}\texttt{,}\ \underbrace{\texttt{Jean}}_{\textit{First}}$$

### 2.2 Using braces

If the first letter of a word is surrounded by braces,[2] it is supposed to be uppercase, unless it follows a TeX command and is lowercase. Here are some examples:

- $\underbrace{\texttt{Alfred Elton \{van\}}}_{\textit{First}}\ \underbrace{\texttt{Vogt}}_{\textit{Last}}$
- $\underbrace{\texttt{Alfred Elton}}_{\textit{First}}\ \underbrace{\texttt{\{\textbackslash relax van\}}}_{\textit{van}}\ \underbrace{\texttt{Vogt}}_{\textit{Last}}$

  (`\relax` is a dummy command [14, Ch. 24]),
- $\underbrace{\texttt{Alfred Elton \{\textbackslash relax Van\}}}_{\textit{First}}\ \underbrace{\texttt{Vogt}}_{\textit{Last}}$

**Remark 1** *Non-letter characters are ignored when BibTeX determines whether the first letter of a word is lower- or uppercase. That is why it considers '`{-}ky`' (see above) to begin with a lowercase letter, ignoring the hyphen sign between braces.*

As mentioned in [16, § 13.2.2], enclosing some characters between braces serves four purposes:

- treating accented letters — accented by means of TeX commands — as single letters:

  ``Christian Vil{\`{a}}``
- treating multiple words as one, especially when a component (*Last*) consists of a single word by default:

  $$\underbrace{\texttt{Michael}}_{\textit{First}}\ \underbrace{\texttt{\{Marshall Smith\}}}_{\textit{Last}}$$

  this feature being commonly used for specifying an organisation name, which consists of only a *Last* part:

  ```
  EDITOR =
  {{Science Fiction and Fantasy
    Writers of America, Inc.}}
  ```
- treating '`and`' as an ordinary word[3] (see the example above);
- delimiting substrings which should *not* change case when the `change.case$` function [16, Table 13.8] is applied.[4]

[1] Only the following name is imaginary (although it is derived from a French poet's name). All the others — some being pseudonyms — name real persons, even if some look strange. That is, the problems raised by BibTeX in the examples we give may arise in real situations.

[2] In the following, we do not consider the braces that might delimit the value of a BibTeX field. 'Braces' should be understood as 'additional braces' in such a case.

[3] '`and`' is viewed as a keyword only at the topmost level, not surrounded by additional braces.

[4] . . . although this use is rare within `AUTHOR` or `EDITOR` fields. See more details in Figure 3.

```
                "Charles" text.length$  pushes  #7            % '#' begins a number.
              "{Ch}arles" text.length$  ......  #7
       "{\relax Ch}arles" text.length$  ......  #6
      "{ \relax Ch}arles" text.length$  ......  #15           % Space after '{'.
           "Charles" #1 #1 substring$   ......  "C"
         "{Ch}arles" #1 #1 substring$   ......  "{"           % Unbalanced brace!
  "{\relax Ch}arles" #1 #2 substring$   ......  "{\"
   "B{\'a}rt{\'o}k" #-2 #3 substring$   ......  "'o}"
             "Charles" #1 text.prefix$  ......  "C"
           "{Ch}arles" #1 text.prefix$  ......  "{C}"
          "}{Ch}arles" #1 text.prefix$  ......  "}{C}"
    "{\relax Ch}arles" #1 text.prefix$  ......  "{\relax Ch}"
   "{ \relax Ch}arles" #8 text.prefix$  ......  "{ \relax }"
   "{ \relax Ch}arles" #3 text.prefix$  ......  "{ \r}"        % Truncate command name!
```

**Figure 2**: Examples of functions dealing with strings in bst.

### 2.3 Words vs tokens

In the above subsections, 'word' has been used in the common sense, that is, 'an independent unit of the vocabulary of a language' [18], so that adjacent words are syntactically separated by space characters or punctuation signs. In fact, if we wish to characterise this notion in the sense of successive tokens handled by BibTeX inside a string representing a name, the separators are whitespace characters[5] — space, tabulation, line feed, form feed, and carriage return — unbreakable space characters, specified by '~' as in TeX, *and the hyphen sign* '-'. Such a choice allows BibTeX to process all the components of a first name more easily when it is abbreviated, even if these tokens are separated by hyphen signs, as in French:

`Jean-Pierre Andrevon`

However this design choice causes strange behaviour to happen in some particular cases:

$$\underbrace{\texttt{Jean-Claude Smit}}_{\textit{First}}\texttt{-}\underbrace{\texttt{ le }}_{\textit{von}}\texttt{-}\underbrace{\texttt{B\{\textbackslash'e\}n\{\textbackslash'e\}dicte}}_{\textit{Last}}$$

whereas this name has only *First* and *Last* parts, as suggested by the single space character. Of course, this name should be specified by:

$$\underbrace{\texttt{Jean-Claude}}_{\textit{First}}\ \underbrace{\texttt{\{Smit-le-B\{\textbackslash'e\}n\{\textbackslash'e\}dicte\}}}_{\textit{Last}}$$

More precisely, BibTeX retains only the first separator between two tokens, omitting any additional separators from the output:

```
Edgar␣␣Rice ⟹ Edgar␣Rice
Edgar␣~Rice ⟹ Edgar␣Rice
Edgar~␣Rice ⟹ Edgar~Rice
Karl-␣Heinz ⟹ Karl-Heinz
```

It also omits any separator before the first token.

**Remark 2** *That is why we put the hyphen sign between braces in '*`{-}ky`*' (see above). If the braces are removed, the hyphen disappears.*

Now let us assume that there is no *von* part.

- If the *Last* part follows the '~' sign, the '~' is omitted:

$$\underbrace{\texttt{Kenneth}}_{\textit{First}}\texttt{\~{}}\underbrace{\texttt{Robeson}}_{\textit{Last}}$$

- If the *Last* part follows the '-' character, the word before belongs to this part, too:

$$\underbrace{\texttt{Louis-Albert}}_{\textit{Last}}$$

### 2.4 Applying a bibliography style

In general, we can see that strings are not handled homogeneously in bst, the language used to write BibTeX's bibliography styles [19]. Let us consider the three following bst functions:

$\mathcal{S}$ `text.length$`

$\mathcal{S}$ $\mathcal{I}$ `text.prefix$`

$\mathcal{S}$ $\mathcal{I}_1$ $\mathcal{I}_2$ `substring$`

where $\mathcal{S}$ is a string and $\mathcal{I}, \mathcal{I}_1, \mathcal{I}_2$ are integers. The bst language is based on handling a stack, so the arguments are passed to a function by putting their values *before* the function name.[6] These three functions respectively push (return) the length of $\mathcal{S}$; the first $\mathcal{I}$ characters of $\mathcal{S}$; and $\mathcal{I}_2$ characters of $\mathcal{S}$, starting at position $\mathcal{I}_1$ if $\mathcal{I}_1 > 0$, or ending at this position if $\mathcal{I}_1 < 0$, in which case positions are counted backward from the end of the string.

Braces do not count when the `text.length$` function is called, but they do for the `substring$` function, as shown by the examples of Figure 2. We

[5] This terminology originates from the Scheme programming language [11, § 6.3.4]: such characters are recognised by the `char-whitespace?` function of this language.

[6] MlBibTeX's compatibility mode allows users to see how this stack works [9].

```
"frank"                ⟹ "FRANK"
"{frank}"              ⟹ "{frank}"
"{\relax frank}"       ⟹ "{\relax FRANK}"
"{\relax {frank}}"     ⟹ "{\relax {FRANK}}"
"{\relax {{frank}}}"   ⟹ "{\relax {{FRANK}}}"
"{\a frank \b f}"      ⟹ "{\a FRANK \b F}"
"{ \relax frank}"      ⟹ "{ \RELAX FRANK}"
"{{\relax frank}}"     ⟹ "{{\relax frank}}"
```

**Figure 3**: ... `"u"` `case.change$` results.

also see that the `substring$` function may push a string where braces are unbalanced. Braces do not count for the `text.prefix$` function, either, and we can observe strange behaviour:

- if an unbalanced left or right brace is encountered, it is put into the string pushed,
- right braces closing unbalanced left braces are added at the end of the string pushed.

The `substring$` function counts each character, in the sense that any typed character is relevant, including enclosed braces and characters surrounded by braces. Other functions — `text.length$` and `text.prefix$` — consider that if a left brace is immediately followed by a '\' character, the complete group between the left enclosing brace and the corresponding right one is viewed as one single character. Such a group is called **special character** within BibTeX's terminology [16, pp. 768–769]. This distinction between special characters and other groups surrounded by braces explains some results shown in Figure 2. Anyway, let us notice that this distinction is recognised by the `change.case$` function:

$\mathcal{S}$ $\mathcal{S}_0$ `change.case$`

— where $\mathcal{S}$ is a string — converts $\mathcal{S}$ to lowercase (resp. uppercase) if $\mathcal{S}_0$ is the string `"l"` (resp. `"u"`), and pushes the result. If $\mathcal{S}_0$ is `"t"`, $\mathcal{S}$ is converted to lowercase except for the first character or the first group if $\mathcal{S}$ begins with a left brace. As shown by the examples given in Figure 3, the non-command parts of a special character are processed like ordinary characters — even if some subparts are surrounded by braces — whereas all the other groups surrounded by braces are left unchanged by the `change.case$` function. That recalls what is written in [16, p. 768] about protecting some uppercase letters ... provided that the first character after a left brace is not a '\' character.

Let us now go back to the operations returning subparts of a string. We see that this data structure is handled with difficulty in the bst language, except for simple strings. If we consider values of the `AUTHOR` and `EDITOR` fields, the usual way to deal with them is the `format.name$` function, used as follows:

```
"Frank Frazetta" ⟹ "Frazetta, +Frank:"
"{-}ky"          ⟹ "{-}ky, :"
```

**Figure 4**: ... `"{ll}, {+ff}:"` `format.name$` results.

$\mathcal{S}_1$ $\mathcal{I}$ $\mathcal{S}_2$ `format.name$`

where $\mathcal{S}_1, \mathcal{S}_2$ are strings and $\mathcal{I}$ is a natural number. This function formats the $\mathcal{I}$th name of $\mathcal{S}_1$ according to the pattern given by $\mathcal{S}_2$, and pushes the result. The pattern is written as follows:

- if characters are not enclosed by braces, they are unconditionally inserted into the result;
- at the first level:
  - if braces enclose other characters than letters, they also are unconditionally inserted into the result;
  - if braces enclose letters other than '`f`', '`j`', '`l`', '`v`' (for '*First*', '*Junior*', '*Last*', '*von*'), it is an error;[7]
  - a single letter ('`f`', etc.) inserts an abbreviation of the corresponding part (see below), while a doubled letter ('`ff`') inserts the complete part.

Examples are given in Figure 4.

Let us say that '`f`' or '`ff`', '`j`' or '`jj`', ... are called **subpatterns**.[8] Each part is processed as follows:

- if this part is absent within the person name, the complete specification surrounded by braces is ignored;
- if this part is not empty:
  - all the characters before the subpattern are inserted before the corresponding part;
  - if the subpattern is immediately followed by a group surrounded by braces, this last group — which may be empty — replaces the separator between two adjacent tokens of the corresponding part;
  - the other characters following the subpattern are inserted after the corresponding part.

If '`~`' characters are put at the end of characters following a subpattern or a separator replacement:

- one '`~`' causes a space character ('␣') to be inserted after the name's part,

[7] If an error occurs, such a bst function pushes a dummy value: an empty string in this case.

[8] This terminology is used within the source files of MlBibTeX's compatibility mode.

```
       last => Le Clerc De La Herverie

"{ll}"            ⟹ Le~Clerc De La~Herverie
"{ll/}"           ⟹ Le~Clerc De La~Herverie/
"{ll/,}"          ⟹ Le~Clerc De La~Herverie/,
"{ll{/},}"        ⟹ Le/Clerc/De/La/Herverie,
"{ll{},}"         ⟹ LeClercDeLaHerverie,
"{ll~}"           ⟹ Le~Clerc␣De␣La~Herverie␣
"{ll~~}"          ⟹ Le~Clerc De La~Herverie~
"{ll{~}~}"        ⟹ Le~Clerc~De~La~Herverie␣
"{ll{~}~~}"       ⟹ Le~Clerc~De~La~Herverie~
"{ll{/},~}"       ⟹ Le/Clerc/De/La/Herverie,␣
"{ll{/}~,~}"      ⟹ Le/Clerc/De/La/Herverie~,␣
"{ll{/}~~,~~}"    ⟹ Le/Clerc/De/La/Herverie~~,~

       last => Zeb Chillicothe Mantey

"{ll}"            ⟹ Zeb~Chillicothe~Mantey
```

```
    first => Jean-Michel-Georges-Albert

"{f}"             ⟹ J.-M.-G.-A
"{f/}"            ⟹ J.-M.-G.-A/
"{f/,}"           ⟹ J.-M.-G.-A/,
"{f{/},}"         ⟹ J/M/G/A,
"{f{},}"          ⟹ JMGA,
"{f~}"            ⟹ J.-M.-G.-A
"{f~~}"           ⟹ J.-M.-G.-A~
"{f{~}~}"         ⟹ J~M~G~A␣
"{f{~}~~}"        ⟹ J~M~G~A~
"{f{/},~}"        ⟹ J/M/G/A,␣
"{f{/}~,~}"       ⟹ J/M/G/A~,␣
"{f{/}~~,~~}"     ⟹ J/M/G/A~~,~

       last => Cousin De Grainville

"{ll}"            ⟹ Cousin De~Grainville
```

**Figure 5**: Examples of using patterns with the `format.name$` function of BibTeX.

- if there are several '~' characters, the first is dropped, while the others are inserted after the name's part.

At other places, tilde characters are inserted like ordinary characters.[9]

Let us assume that there are several tokens for a name's part. By default — without separator redefinition — a '~' character is inserted:

- always between the next-to-last and last tokens,
- between the first and second tokens: if there are three or more tokens, and the first token is one or two letters long, a special character or period belonging to an abbreviated word is counted as one letter.

All these cases are summarised in the examples given in Figure 5.

A token is abbreviated by retaining only its first letter. So, other characters inserted before this first letter may be dropped. When BibTeX abbreviates a name part, it recognises special characters, as shown in Figure 6. But it does not insert braces for other groups surrounded by braces, as the `text.prefix$` function would do: compare the examples given in Figures 2 & 6 for the string `{Ch}arles`. Last, let us mention that by default — without separator redefinition — this first letter is followed by a period character and the separator — a space, '~' or '-' character — put after this token. This mark after the first letter is not appended after the abbreviation of the last token. Some of these rules might seem strange if we think of them only in relation to abbreviating first names, but let us recall that some styles use initials of the *von* and *Last* parts as labels of bibliographical references. For example, the alpha bibliography style uses the `{v{}}{l{}}` pattern to generate these labels.

### 2.5 Criticism

The functions provided by BibTeX work fine in most practical cases, in the sense that most BibTeX end-users accept the results of the standard bibliography styles. However, the cases not 'naturally' included in this framework are difficult to handle. A simple example is given by abbreviations. Let us consider the two following names:

Edgar Rice Burroughs
Jon L White

The `{f.}` subpattern would cause a period to be appended after 'R' in 'E. R. Burroughs', but incorrectly abbreviates the second example to 'J. L. White'.[10] The `{f}` subpattern correctly puts 'J. L White' in the second case, but then a period is missing after Edgar Rice Burroughs' middle name. In addition, it is difficult to specify a particular abbreviation for a middle name only. For example, Nicholas deBelleville Katzenbach's middle name is correctly abbreviated to 'deB.'.

From a general point of view, some workarounds exist, but may be viewed as *hacks*. Often they consist in inserting LaTeX commands into fields' values. For example:

- a command deferring the right case [16, p. 767]:

$$\texttt{Maria }\underbrace{\texttt{\{\textbackslash MakeUppercase\{d\}e La\}}}_{\textit{von}}\texttt{ Cruz}$$

[9] Let us notice that, in contrast, any occurrence of the space character is processed w.r.t. a 'standard way'.

[10] Anyway, [2, § 14.4] recommends the systematic use of a final period. But such a sign is supposed to replace some letters omitted.

```
Charles           ⟹ C
{Ch}arles         ⟹ C
{\relax Ch}arles  ⟹ {\relax Ch}
{-}ky             ⟹ k
```

**Figure 6**: Abbreviating first names in bst.

- so BibTeX interprets ‘{de...}’ as the beginning of the *von* part, because the ‘d’ letter is lowercase, even though LaTeX will typeset ‘De’ when the command is processed;
- inserting a dummy accent command [13, § 251]:

  `{\relax Ph}ilippe Djian`

  in order for this French first name to be abbreviated to ‘Ph.’ because French digraphs — ‘ch’ is a digraph for ‘[ʃ]’ — should not be reduced.

Other ‘tricks’ are more subtle. For example, standard bibliography styles put a space character between the *von* and *Last* part. That is suitable for most cases, e.g., ‘Lyon Sprague de Camp’, but not when the particle ends with an elision, e.g., ‘Guy d’Antin’. Testing whether or not the *von* part ends with an apostrophe character (“ ’ ”) is tedious, because the bst language does not provide a natural way to store part of a name in a variable. In addition, let us not forget that there may be several names inside an AUTHOR or EDITOR field, which complicates the search of the accurate indices. One solution is:

```
Guy d'\unskip Antin
```

— see [14, Ch. 24] about the \unskip command — but this is not fully satisfactory: it works only if we are sure that the text.prefix$ function is not applied to the *von* part for a prefix length greater than 2. Likewise, the change.case$ function should not be used (*cf. supra*). Protecting this \unskip command by braces:

```
Guy d'{\unskip} Antin
```

would solve these problems but annihilate the effect of this command. The insertion of an \aftergroup command [14, Ex. 24.7], deferring some tokens until the end of a group is processed:

```
Guy d'{\aftergroup\unskip} Antin
```

is useless because the offending space character *follows* the brace closing the group with \aftergroup. And this space, not surrounded by braces, is needed when BibTeX separates the *von* and *Last* parts. The best solution — cf. [14, Ch. 24] — is:

```
Guy d'{\aftergroup\ignorespaces} Antin
```

in the sense that it works in most cases, provided that the abbreviated first name is not to be followed by a delimiter, as in ‘[Guy d’]Antin’.

```
<author>
  <name>
    <personname>
      <first abbrev="L. Sprague">
        Lyon Sprague
      </first>
      <von>de</von>
      <last>Camp</last>
    </personname>
  </name>
  <and/>
  <name>
    <personname>
      <first>David</first>
      <last>Drake</last>
    </personname>
  </name>
</author>
```

**Figure 7**: The internal representation of names in MlBibTeX.

As mentioned above, the format.name$ function allows good control of separators between the tokens belonging to a same part. However, some limitations exist. For example, let us consider:

```
Ursula Kroeber {Le~Guin}
```

By default — that is, using the {ff} pattern — an unbreakable space character will be inserted between the ‘actual’ first name (‘Ursula’) and the middle name (’Kroeber’). If we would like to allow a linebreak after the first name because the bibliography will be typeset on a small text width, we can do that by the ‘{ff{ }}’ pattern. But such a redefinition is impossible for the two tokens of the last name (‘Le Guin’), surrounded by braces. In such a case, the name may be specified by:

```
Le Guin, Ursula Kroeber
```

but braces are needed for an organisation name:

```
AUTHOR = {{Hidalgo~Trading~Company}}
```

and redefining the separator between words becomes impossible by means of the format.name$ function since BibTeX considers there to be only one token. We can program this operation with the functions substring$ and * — concatenation of two strings [16, Table 13.8] — but it is very tedious.

From our point of view, the use of additional braces belongs to BibTeX’s ‘philosophy’, but is complicated in the sense that some functions process braced groups opening a TeX command differently from other groups surrounded by braces. Often this design choice is good — for example, it allows the change.case$ function to change the case of accented letters typed by means of TeX commands —

```
<nbst:template match="von">
  <nbst:variable name="the-part" select="."/>
  <nbst:value-of select="$the-part"/>
  <nbst:if test='substring($the-part,string-length($the-part),1) != "&quot;"'>
    <!--  '&quot;' is a predefined entity for an apostrophe character [21, p. 48].      -->
    <nbst:text> </nbst:text>
  </nbst:if>
</nbst:template>
```

**Figure 8**: Putting a particle's name down.

but other operations are difficult to perform. The insertion of (LA)TeX commands seems to us to be just a workaround. It works since BibTeX is used in conjunction with LaTeX. That was true when BibTeX came out,[11] but is not always the case nowadays. BibTeX may be used to generate bibliographies displayed on Web pages written in HTML,[12] by means of a converter like BibTeX2HTML [3]. Another example, closer to LaTeX, is Hans Hagen's format ConTeXt [4]. Using LaTeX commands within values associated with BibTeX fields can cause trouble when these programs run (some examples concerning ConTeXt and solutions are given in [10]).

## 3 How names are processed by MlBibTeX

### 3.1 Implementation issues

We suggest that the components of a name should be directly accessible by means of different placeholders within the functions of a bibliography style. As explained in [7], parsing bibliographical entries from a .bib file results in an XML[13] tree in MlBibTeX.[14] The organisation of our elements is a revision and extension of the DTD[15] given in [6], influenced by BibTeX. Concerning names, fields related to authors and editors are split into subtrees, as shown in the example of Figure 7. In fact, this figure is a 'pretty-printing' of such a tree: in reality, the contents of text nodes — e.g., `first`, `von`, `last` — are 'space-normalised', that is, stripped of leading and trailing whitespace characters, multiple consecutive occurrences of whitespace characters being replaced by a single space character. Likewise, most of the blank nodes[16] pictured in Figure 7 do not exist in the 'actual' representation. Besides, this representation uses Latin 1 encoding,[17] and some special characters of LaTeX are processed; for example, the '~' character is replaced by an unbreakable space character (numbered 160 in Latin 1). In addition, some LaTeX commands are expanded; for example, accent commands applied to suitable letters are replaced by the corresponding accented letters included in Latin 1 [9].

Bibliography styles are written using the nbst[18] language, close to XSLT,[19] a language of transformations used for XML texts. This nbst language is described in [7].

An example written using nbst is given in Figure 8. If the *von* part of a name exists, this template is invoked, the contents of this *von* part is written down. This part is followed by a space character, unless its last character is an apostrophe. This template allows the two examples given above — 'Lyon Sprague de Camp' and 'Guy d'Antin' — to be displayed nicely. Of course, this is an *ad hoc* solution, but it shows that we get the full expressive power of a programming language. In addition, we can call functions written in Scheme — the implementation language of MlBibTeX — for difficult cases [7, 8].

### 3.2 Syntactic issues

MlBibTeX can process any .bib file designed for 'old' BibTeX, except that square brackets are syntactic delimiters used for multilingual features [7]. So, most of the 'tricks' used within 'old' .bib files should work. In addition, MlBibTeX allows more explicit syntax for the components of a person name and the abbreviation of a first name, when needed:

```
first => ..., von => ..., last => ...,
junior => ..., abbr => ...
```

[11] Initially, BibTeX was designed to work with Scribe [22].

[12] **H**yper**T**ext **M**arkup **L**anguage. See [17] for an introduction.

[13] E**X**tensible **M**arkup **L**anguage. See [21] for an introduction.

[14] More precisely, an XML tree represented in Scheme using the conventions of SXML (Scheme implementation of XML) [12].

[15] **D**ocument **T**ype **D**efinition. A DTD defines a document markup model [21, Ch. 5].

[16] Anonymous text nodes whose contents are only whitespace characters; two examples can be found around the `and` tag in Figure 7. In XML, all characters are preserved [21, p. 38].

[17] Future versions of MlBibTeX should be able to deal with Unicode characters [24].

[18] **N**ew **B**ibliography **ST**yles.

[19] E**X**tensible **S**tylesheet **L**anguage **T**ransformations. See [21, Ch. 6] for ashort introduction.

The order of the keywords is irrelevant and some may be absent, provided that the last name is specified. For example:

```
first => Kim Stanley, last => Robinson
```

where the *von* field is empty, and the abbreviation of the first name is standard, that is, '`K.~S.`' Let us notice that in MlBIBTEX, the period character ending an abbreviation belongs to it by default. In addition, this new syntax can be used with 'old' bibliography styles, written in bst, as we show in Appendix A.

You can mix the 'old' and 'new' syntaxes, in which case a name is parsed like (i) if no comma occurs before a keyword, like (ii) (resp. (iii)) if the number of commas not followed with a keyword is one (resp. two) and the keywords give additional information. Let us give some examples:

```
Robinson, first => Kim Stanley
```

is allowed, because '`Robinson`' is parsed as the *Last* part, so '`Kim Stanley`' is allowed to be the *First* part. But:

```
Kim Stanley, last => Robinson
```

***WRONG!***

is an incorrect specification, because '`Stanley`' is supposed to be the *Last* part, so this part cannot be redefined to '`Robinson`'.

In practice, mixing the old and new syntaxes is useful when we have just to give a specific abbreviation for a first name:

```
Lyon Sprague de Camp, abbr => L. Sprague
```

Roughly speaking, this syntax is close to Ada's for passing values inside a subprogram call [23, § 6.4].

Other keywords can be used for both an organisation name and a key for sorting:

```
EDITOR = {org => \TUG 2006,
          sortingkey => TUG2006}
```

As in BIBTEX, co-authors are connected with the '`and`' keyword. After the different co-authors, MlBIBTEX allows the addition of *collaborators*, introduced by the '`with`' keyword:[20]

```
Clive Cussler with Paul Kemprecos
```

The format for several co-authors and collaborators:

```
... and ... and ... with ... with ...
```

In the present article's bibliography, reference [16] gives an example of how such an entry using several co-authors and collaborators is formatted. As in BIBTEX, the '`others`' keyword can be used when additional names are left unspecified: '`and others`' (resp. '`with others`') for additional unspecified co-authors (resp. collaborators).

Multilingual switches with a default[21] are allowed for names, which is useful for names originating in languages using other alphabets:

```
AUTHOR =
  {[Сергей Сергеевич Прокофьев] * russian
   [Sergey Sergeyevich Prokofiev]}
```

## 4 Internationalization of names

MlBIBTEX allows names to be displayed according the cultural background of a language. For example, accurate templates of the nbst language [8] allow names of Hungarian people:

```
AUTHOR = {[Béla Bartók] : magyar}
```

to be displayed like 'Bartók Béla' — that is, the last name at first, followed by the first name — whereas other names (English, French, ...) are displayed as usual, that is, the first name followed by the last name. However, the specification of any person name has to be dispatched into the four components inherited from BIBTEX. This is not a problem related to parsing, because the new keywords allow us to specify each component separately. In practice, some cases are solved by extending the *First* part in order to include:

- a middle name for American people:

  Ursula K. Le Guin

- a *patronym* (father's first name) for a Russian name, here 'Sergey, Sergey's son':

  Сергей Сергеевич Прокофьев

Other cases may more difficult to handle. Here are some exotic examples given in [27]:

- an Assyrian name consisted of a personal name, the father's name, and the grandfather's name,
- in South India, a personal name may be preceded by the father's name, usually written as an initial, and possibly replaced or supplemented by the birthplace or mother's house name, e.g. 'Trivandrum R. S. Mani'.[22]

We plan to go thoroughly into this notion when the document model of the bibliographies handled by MlBIBTEX is revised by using *schemas.* For small examples of using the XML Schema standard for describing an organisation for names, [25, pp. 91–107] can be consulted. DocBook, an XML system for writing structured documents [26], proposes a more open approach: some optional elements are

[20] ... at the topmost level only. See Footnote 3, p. 244.

[21] This means that something is *always* produced, even if no information is available in the reference's language. In other words, this kind of switch never yields nothing. See [7] for more details.

[22] In such a case, periods are sometimes omitted.

```
<author>
  <honorific>Sir</honorific>
  <firstname>Arthur</firstname>
  <surname>Conan Doyle</surname>
</author>

<author>
  <firstname>Edgar</firstname>
  <surname>Burroughs</surname>
  <othername role="mi">Rice</othername>
</author>

<author lang="fr-BE">
  <firstname>J.-H.</firstname>
  <surname>Rosny</surname>
  <lineage>
    <!--  Junior, Senior, etc. [26, p. 308].      -->
    aîné
  </lineage>
</author>
```

**Figure 9**: Names specified using DocBook.

defined — e.g., `honorific` (see Figure 9) — including a hold-all element, `othername`, for information that does not fit in other categories. This element — which may be repeated — has a `role` attribute that classifies the different kinds of these 'other' names. However, official documents do not make precise the possible values for this attribute: so names' taxonomy is not really fixed in DocBook, except for classic cases, for example, '`mi`' for 'middle names', as given in [26, p. 149]. Figure 9 shows a DocBook example, including language information. This information is not only a single identifier, like an option of the babel package, but allows the specification of variants of a language. It consists of a two-letter language code using lowercase letters, optionally followed by a two-letter country code using uppercase letters [1]: '`fr`' is for the French language, '`BE`' for Belgium. Such elements are not limited to a bibliography: as another example, the specification of an author's article also uses them [26, p. 143].

Another formalism belonging to the XML family widely used for bibliographical metadata is the *Dublin Core*,[23] but here the contents of the elements representing people's names are just strings; such elements are not structured by means of subelements as in DocBook. For example, an entity primarily responsible for making the content of a resource may be a person name and is specified by the `dc:creator` element. BibTeX-like syntax is used in practice, as shown in Figure 10; [27] gives some guidelines.

```
<dc:creator>Kay, Guy Gavriel</dc:creator>
<dc:creator xml:lang="fr">
  <!--  xml:lang is a predefined attribute in XML
        [21, p. 41], its values are codes described in
        [1].
    -->
  Pierre Pelot
</dc:creator>
```

**Figure 10**: Examples with the *Dublin Core*.

## 5 Further development and conclusion

As mentioned above, MlBibTeX includes a compatibility mode, so it can apply 'old' bibliography styles, written using bst.[24] To put this implementation of bst into action, of course, we studied the behaviour of this language's functions precisely, preparing very many tests. That is why we hope that, in particular, we know precisely how BibTeX deals with names. This task of reverse engineering showed us that it was designed in order for current format operations to be specified concisely. The price paid is complicated specification of general cases.

Due to the huge number of BibTeX database files written by end-users, a successor of BibTeX has to ensure compatibility with existing .bib files. So we have to pay attention whenever we introduce new syntactic sugar: could existing files be processed as previously? Another question is: could we add more and more syntactic sugar to a formalism without damaging it? Redefining a new one might be better ...

We think that the work on the new features of MlBibTeX should be done based on a specification in XML, and some improvements can be given syntactic sugar, usable within .bib files. For example, we plan to add a `space-after` attribute to the `von` element:

```
<von space-after="no">d&apos;</von>
```

and specify empty space after a particle by an additional comma in AUTHOR and EDITOR fields:

```
{first => Guy, von => d’,, last => Antin}
```

or — when the *von* part is specified last — :

```
{first => Gilles, last => Argyre,
 von => d’,}
```

On the contrary, future features related to person names whose structure is to be studied will be added only to the XML document model of bibliographies, when it is ready. We think that BibTeX has not been much used for such names, so there should be no need for additional syntactic sugar and

[23] The Dublin Core metadata standard is a simple yet effective element set for describing a wide range of networked resources. See [5] for more details.

[24] See [9] for an overview of the functions of this mode.

```
first => J.-H., last => Rosny, junior => jeune and
first => Pierre Alexis, last => Ponson du Terrail and
first => Eric, von => Van, last => Lustbader

((*name*)
 "Rosny, jeune, J.-H. and Ponson du Terrail, Pierre Alexis and Van Lustbader, Eric" .
 #(#((14 . 19) (7 . 12) (0 . 5) #f)   ; The order is First, Junior, Last, von. The First part of the first
                                      ; element starts at position 14 and ends just before position 19. This
                                      ; element does not have a von part.
   #((43 . 56) #f (24 . 41) #f)
   #((76 . 80) #f (65 . 74) (61 . 64))))
```

**Figure 11**: How MlBibTeX's names are sent to original BibTeX's functions.

MlBibTeX should be able to get such information by parsing XML files.

## 6 Acknowledgements

Many end-users told me that they had difficulty understanding names' specification within BibTeX. I was thinking of them when I was writing this article and I hope they will enjoy reading it as much as I enjoyed writing it. Many thanks to Karl Berry and Barbara Beeton, both of whom proofread a first version, suggested some improvements and additional examples.

## A MlBibTeX's names within compatibility mode

As mentioned above, MlBibTeX allows users to run 'old' bibliography styles of BibTeX [19], by means of a *compatibility mode*, sketched in [9]. This compatibility mode, programmed in Scheme, actually uses a stack and allows users to learn the bst language easily, since they can run a bst program step by step. The data used within the result of parsing .bib files are serialised w.r.t. types used within the bst language's functions, that is, strings, integers, and literals [16, Table 13.8]. Each function belonging to the bst library is implemented by a Scheme function.

If you push a string and would like to give it to the Scheme function implementing `format.name$`, only the conventions of BibTeX will be put into action. If the value of an `AUTHOR` or `EDITOR` has been processed by MlBibTeX's parser and has to be passed to the compatibility mode, the transmitted value is a list consisting of a string $\mathcal{S}$, followed by a *vector*,[25] whose size is the number of the names of $\mathcal{S}$. Let us consider a name belonging to $\mathcal{S}$, for each part of it — *First*, *Junior*, *Last*, *von*, w.r.t. this order — a pair of indices shows where this part begins and ends. This part is replaced by the false value — `#f` — if it does not exist. Unless a name only consists of a *Last* part, the possible syntaxes for the string $\mathcal{S}$ are either (ii) or (iii) — cf. § 2.1.

An example is given in Figure 11: given three person names specified by means of keywords, we show the three elements of the corresponding structure passed to the compatibility mode. Index pairs are organised into vectors: one vector for the parts of each, these three vectors being elements of a vector of vectors. If such a list, beginning with the '`*name*`' marker, is popped by function other than `b-bst-format.name$` and `b-bst-num.names$`, the implementations of `format.name$` and `num.names$`, only the string is retained.

This design choice allows us to have BibTeX's behaviour by default if these two functions are given only a string. But if end-users take advantage of the keyword specification of name parts, we do not need workarounds within the strings passed to these two functions. We do not have to use additional braces or dummy LaTeX commands. Besides, our indices are coherent with the way used by Scheme for selecting substrings: they are characters beginning with a first inclusive index and ending with a second exclusive index. For example, let `s` be the long string given in Figure 11: the expression `(substring s 14 19)` evaluates to the string `"J.-H."`.

Last, the names of collaborators, introduced after an occurrence of the '`with`' keyword (cf. § 3.2), are ruled out. Finally, let us remark that these names would be processed improperly by BibTeX's bibliography styles.

## References

[1] Harald Tveit ALVESTRAND: *Request for Comments: 1766. Tags for the Identification of Languages.* UNINETT, Network Working Group. March 1995. `http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1766.html`.

[25] Vectors of Scheme are analogous to arrays of 'classical' programming languages.

[2] *The Chicago Manual of Style*. The University of Chicago Press. The 14th edition of a manual of style revised and expanded. 1993.

[3] Jean-Christophe Filliâtre and Claude Marché: *The BibTeX2HTML Home Page*. June 2006. `http://www.lri.fr/~filliatr/bibtex2html/`.

[4] Hans Hagen: *ConTeXt, the Manual*. November 2001. `http://www.pragma-ade.com/general/manuals/cont-enp.pdf`.

[5] Diane Hillman: *Using Dublin Core*. November 2005. `http://dublincore.org/documents/usageguide/`.

[6] Jean-Michel Hufflen: "Multilingual Features for Bibliography Programs: From XML to MlBibTeX". In: *EuroTeX 2002*, pp. 46–59. Bachotek, Poland. April 2002.

[7] Jean-Michel Hufflen: "MlBibTeX's Version 1.3". *TUGboat*, Vol. 24, no. 2, pp. 249–262. July 2003.

[8] Jean-Michel Hufflen: "Bibliography Styles Easier with MlBibTeX". In: *Proc. EuroTeX 2005*, pp. 179–192. Pont-à Mousson, France. March 2005.

[9] Jean-Michel Hufflen: "BibTeX, MlBibTeX and Bibliography Styles". *Biuletyn GUST*, Vol. 23, pp. 76–80. In *BachoTeX 2006 conference*. April 2006.

[10] Jean-Michel Hufflen: "MlBibTeX Meets ConTeXt". In: *EuroTeX 2006 conference, preprints*, pp. 71–76. Debrecen, Hungary. July 2006.

[11] Richard Kelsey, William D. Clinger, Jonathan A. Rees, Harold Abelson, Norman I. Adams iv, David H. Bartley, Gary Brooks, R. Kent Dybvig, Daniel P. Friedman, Robert Halstead, Chris Hanson, Christopher T. Haynes, Eugene Edmund Kohlbecker, Jr, Donald Oxley, Kent M. Pitman, Guillermo J. Rozas, Guy Lewis Steele, Jr, Gerald Jay Sussman and Mitchell Wand: "Revised$^5$ Report on the Algorithmic Language Scheme". *HOSC*, Vol. 11, no. 1, pp. 7–105. August 1998.

[12] Oleg E. Kiselyov: *XML and Scheme*. September 2005. `http://okmij.org/ftp/Scheme/xml.html`.

[13] Marie-Paule Kluth : *FAQ LaTeX française pour débutants et confirmés*. Vuibert Informatique, Paris. Également disponible sur `CTAN:help/LaTeX-FAQ-francaise/`. Janvier 1999.

[14] Donald Ervin Knuth: *Computers & Typesetting. Vol. A: The TeXbook*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1984.

[15] Leslie Lamport: *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts. 1994.

[16] Frank Mittelbach and Michel Goossens, with Johannes Braams, David Carlisle, Chris A. Rowley, Christine Detig and Joachim Schrod: *The LaTeX Companion*. 2nd edition. Addison-Wesley Publishing Company, Reading, Massachusetts. August 2004.

[17] Chuck Musciano and Bill Kennedy: *HTML & XHTML: The Definitive Guide*. 5th edition. O'Reilly & Associates, Inc. August 2002.

[18] *Oxford Advanced Learner's Dictionary of Current English*. Oxford University Press. 1989.

[19] Oren Patashnik: *Designing BibTeX Styles*. February 1988. Part of the BibTeX distribution.

[20] Oren Patashnik: *BibTeXing*. February 1988. Part of the BibTeX distribution.

[21] Erik T. Ray: *Learning XML*. O'Reilly & Associates, Inc. January 2001.

[22] Brian Keith Reid: *SCRIBE Document Production System User Manual*. Technical Report, Unilogic, Ltd. 1984.

[23] S. Tucker Taft and Robert A. Duff, eds.: *Ada 95 Reference Manual. Language and Standard Libraries*. No. 1246 in LNCS. Springer-Verlag. International Standard ISO/IEC 8652:1995(E). 1995.

[24] The Unicode Consortium: *The Unicode Standard Version 4.0*. Addison-Wesley. August 2003.

[25] Eric van der Vlist: *XML Schema*. O'Reilly & Associates, Inc. June 2002.

[26] Norman Walsh and Leonard Muellner: *DocBook: The Definitive Guide*. O'Reilly & Associates, Inc. October 1999.

[27] Andrew Waugh: *Representing People's Names in Dublin Core*. February 1998. `http://dublincore.org/documents/name-representation/`.

# Abstracts

### How to Create a TEX Journal: A Personal Journey

Barbara Beeton

When TUG was first formed, the Internet was not generally available; the logical channel for communication with and among TUG's members was on paper. So *TUGboat* came into being.

As TEX has matured, the needs of the community have evolved, but paper is still a logical medium for showcasing a typesetting tool.

This talk will introduce high- and low-lights in the history of *TUGboat*, some reasons for choosing its particular format and mode of presentation, several experiments, and lots of my personal experiences as editor.

(This was the keynote address at the Practical TEX 2006 conference, and the full paper will be published in that proceedings. *Ed.*)

### TEX, typography & art together

Gyöngyi Bujdosó

In my previous talk presented at EuroTEX 2006, the TEX side of a TEX and typographical e-learning system was presented.

In this talk the main aspects of the other, typography-related, part of the system will be shown. The design of this part contains topics that can be of assistance to TEX users in designing their documents by presenting various fonts, page layouts and illustrations. The presentations of the type designers, typographers and artists of the sample creations will also be discussed.

### ProTEXt, a complete TEX system for beginners

Thomas Feuerstack and Klaus Höppner

One of TEX's largest strengths is the high modularity and flexibility of the program and related tools. Besides the processor itself, every TEXnican may use the editors, post processor programs, etc. he prefers most. On the other hand, for beginners or only interested persons this advantage can lead to difficulties, especially in times, where users have gotten accustomed at "complete environments".

To come to the details: While the overall information technology scene is regarded as fast moving, the principle of working with data remains essentially unchanged. People still "feed" programs with sources and expect transformed data as a result — the difference is in the bandwidth of the programs they use. When I started working with TEX in the early 1980s, I used IBM's `xedit` for input, which was the same editor I used for all other programs, i. e. SAS or PL/1 sources. Nowadays all those former "programs" have increased to suites, which means they are shipped with their own editor, post-processing routines, etc., while TEX is still an exception, because of its traditional delivery of core functionality.

At this point you may imagine the difficulties a newbie will run into when planning to use TEX. Expecting a single and simple installation, like other tools, he will soon find himself confronted with the fact that he has to take care for several tools in addition to TEX — most of which he has never imagined might be needed. Asking "the experts" which procedures are best to follow, may in the worst case even increase confusion.

According to our experiences, leading beginners to a new system will hopefully end in success, if we take heed of the following simple advice:

- A beginner prefers a system which is big enough to satisfy his needs, and which is likewise small enough so that he doesn't lose a general overview. For example, he normally doesn't want to waste time in choosing the best editor, especially when he can't estimate the result.
- A newbie expects an complete system instead of single components, especially when he can't see the connection between them. If this isn't possible (like with TEX), collect the *smallest size* of components needsed to achieve a first success, and give him a *simple and short* overview/introduction of the relationship which these modules have to each other.
- In addition to the last point, it is nice when you can start with the newest tools, but this is not a requirement. In other words: in only the rarest cases will you start your driving career with a brand new Mercedes Benz, and most likely you won't regard this as a disadvantage.

To come to the end, in our mind the best solution for the problem mentioned above would be a seasonable single-installation comparable to the suites described above.

In the meantime, ProTEXt was introduced to enable even beginners to easily setup a complete running system, eliminating one of the main obstacles in using TEX. We will present the current status of the project, experiences from the previous releases and recent or planned changes.

### OpenMath and MathML in practice

Hans Hagen

The Dutch Mathadore project is an experiment that has as main objective to provide content that can be used for math courses in secondary education as well as supplementary courses in permanent education. The content is highly interactive and this made the team decide for OpenMath. This choice is also driven by the fact that the University of Eindhoven is on of the participants.

In this talk I will discuss the role that TEX plays in this game. I will present the way we deal with OpenMath, intermediate MathML and the final representation on paper. I will also discuss the way authors are dealing with the input, what problems they encounter and how they need to deal with the lack of control in XML based environments.

### MetaPost developments — autumn 2006

Taco Hoekwater

The new release of MetaPost includes some new features as well as a number of bug fixes. The new functionality includes: the possibility of using a template for the naming of output files; support for CMYK and greyscale color models; per-object PostScript specials; the option to generate Encapsulated PostScript files adhering to Adobe's Document Structuring Conventions; the ability to embed re-encoded and/or subsetted fonts; and support for the GNU implementation of troff (groff).

(This talk was also presented at EuroTEX 2006, and therefore the full paper was already published in that proceedings, *TUGboat* 27:1. The paper is also available online at `http://tug.org/metapost/articles`. *Ed.*)

### TEX producing legal documents

Jerzy Ludwichowski

A subsystem for producing legally binding documents for the admission system of the Nicolas Copernicus University, Toruń, Poland, will be presented. The documents are generated based on data that comes from the system's database. It will be shown how using the simplest means helps to achieve results which look almost impossible to those uninitiated to TEX.

The presentation will give the motivation for using plain TEX and LATEX as the base for the production of the documents, difficulties encountered and solved, the general design and the (not very advanced) TEX mechanisms used.

The approach used is a standard one, used before and publicly described an uncountable number of times. Nonetheless, it should be interesting for the conference participants who are not expert TEXnicians.

### TEX Live — Life with TEX

Gerben C. Th. Wierda and
Renée M. E. van Roode

When Mac OS X first appeared, people soon started to work on getting TEX to run on it. Being a Unix system with PDF as the screen language, soon front ends appeared that made use of a (hidden) standard Unix TEX (with pdfTEX) in the background and the built-in ability of Mac OS X to handle PDF. teTEX was an obvious choice as a downloadable distribution. Soon, the process of downloading, compiling and installing was done on behalf of users and the first TEX installers (based on a precompiled teTEX) appeared.

Over time, one of these became popular. This redistribution of TEX first was a redistribution of teTEX (precompiled with basic configuration options like paper size), then it quickly became a mix of TEX Live as the basis for the programs (because it was richer than teTEX) and teTEX for the basic texmf tree (because it was well maintained and a near perfect starter set).

Now that teTEX as a separate distribution is no longer maintained it has become time to migrate this Mac OS X redistribution (which for want of a better name we will call gwTEX) to something that is based on a subset of TEX Live. The talk will focus on this migration and what was learned about TEX along the way.

The presentation will end with a short accompanying guest talk entitled "Life with TEX".

## *MAPS* 33 (Fall 2005), 34 (Spring 2006)

*MAPS* is the publication of NTG, the Dutch language TEX user group. Their web site is `http://www.ntg.nl`.

### *MAPS* 33, Fall 2005

Wybo Dekker, Redactioneel [From the editor]; p. 1

Overview of the issue's contents.

Maarten Sneep, Wachten op een Ca-tas-tro-fe [Waiting for a Cata-strophe]; pp. 2–3

Opinion on the new Dutch spelling rules.

Dennis van Dok, Jewel case listings for MP3 CD-ROMs; pp. 4–13

Making jewel case listings for MP3 CD-ROMs is a particular challenge, since up to about ten times as much information has to be on them as on jewel cases for regular audio discs. Here TEX's abilities to adjust entire paragraphs, as opposed to just single lines, shine.

Siep Kroonenberg, Font installation the shallow way; pp. 14–18

[Printed in *TUGboat* 27:1.]

Idris Samawi Hamid, Installing expert fonts: Minion Pro; pp. 19–35

Installing fonts for ConTEXt can be an intimidating business. In this issue we take on a real monster: a collection of Adobe Minion Pro expert fonts. We hope our installation of this collection will provide an illustrative example for ConTEXt users, and help to ease the pain of installing new fonts (if you can install Minion Pro, Myriad Pro and Poetica, you can install just about anything!).

Hans Hagen, Hyphenation patterns; pp. 36–40

[Printed in *TUGboat* 26:3.]

Taco Hoekwater, What do you do with ConTEXt?; pp. 41–45

User responses to the question: "What do you do with ConTEXt?".

Piet van Oostrum, Een uittreksel uit recente bijdragen in het CTAN archief [An extract from recent contributions to the CTAN archive]; pp. 46–51

This article describes a number of recent contributions to the CTAN archive (and possibly other Internet sources). The selection is based upon what I find interesting myself and what I think will be interesting for others. It is not a complete overview. Neither are the summaries to be considered as manuals. Look on it as a kind of menu to whet your appetite.

This installment focuses on graphical packages (amongst others PGF/TikZ, pst-pdf), some graphical programs (fig2vect, LATEXPiX, sketch, and some conversion programs) and special applications like chemistry, music and sudoku.

Jan van de Craats, Color separation; pp. 52–53

The book Basisboek wiskunde (*Basic Mathematics*) by Jan van de Craats and Rob Bosch was typeset in LATEX and submitted for printing as one big PDF file. In this book one extra color (blue) was used for titles, headings, footings, important formulas, figures and also as a background color for certain pages or parts of text. Jan van de Craats, who did the typesetting, reports on a trick for obtaining color separation without flaws.

Hendri Adriaens, Powerdot — making presentations with LATEX; pp. 54–58

This article describes some technical details of the `powerdot` class which was developed during the summer holidays of 2005.

Siep Kroonenberg, Managing a network TEX installation under Windows; pp. 59–64

[Printed in *TUGboat* 27:1.]

Ovidiu Gheorghies, An Introduction to MetaUML—Exquisite UML Diagrams in MetaPost; pp. 65–86

MetaUML is a GNU GPL MetaPost library for typesetting exquisite UML (Unified Modeling Language) diagrams. MetaUML offers a highly customizable, object-oriented API, designed with ease of use in mind. This paper presents usage examples as well as a description of MetaUML infrastructure. This infrastructure may prove useful for general MetaPost typesetting, providing object-oriented replacements and enhancements to the functionality offered by the `boxes` package.

### *MAPS* 34, Spring 2006

Taco Hoekwater & Wybo Dekker, Editorial; p. 1

Taco Hoekwater, Een laatste dingetje ... [One last thing ...]; p. 2

Hans Hagen, What tools do ConTEXt users have?; pp. 3–7

[Printed in *TUGboat* 27:1.]

Taco Hoekwater & Hans Hagen, Announcement: ConTeXt user meeting 2007; p. 8

Hans Hagen, MKII–MKIV; pp. 9–21

[Printed in this issue of *TUGboat*, pp. 256= 219 256–227.]

Aditya Mahajan, Display Math in ConTeXt; pp. 22–34

This article explains how to do various kinds of alignments in ConTeXt. A visual output is presented, and it is then shown how that effect can be achieved in LaTeX and ConTeXt. We hope that article will make the transition from LaTeX with the 'amsmath' package to ConTeXt easier.

Taco Hoekwater, MetaPost developments; pp. 35–37

[Printed in *TUGboat* 27:1.]

Jerzy Ludwichowski, Announcement: EuroBachoTeX 2007; p. 38

Bogusław Jackowski, Appendix G illuminated; pp. 39–46

[Printed in *TUGboat* 27:1.]

Hans Hagen & Jerzy Ludwichowski & Volker Schaa, The New Font Project: TeX Gyre; pp. 47–50

[Printed in this issue of *TUGboat*, pp. 256= 230 256–233.]

Taco Hoekwater & Hans Hagen, The making of a (TeX) font; pp. 51–54

[Printed in *TUGboat* 27:1.]

Paul Lemmens, Je proefschrift in LaTeX zetten [Typesetting your thesis in LaTeX]; pp. 55–64

In this article I will describe how I typeset my thesis in LaTeX. I will mention my work environment, the extra packages I used, the (local) tricks I applied inside the source document, the problems I found, and the solutions for those.

Hans van der Meer, Random bit generator in TeX; pp. 65–67

A random bit generator code in TeX macros makes it possible to integrate random numbers and decisions in the production of documents.

Paweł Jackowski, Enjoy TeX pearls diving!; pp. 68–77

The BachoTeX 2006 conference continued the Pearls of TeX Programming open session, introduced in 2005, during which volunteers present TeX-related tricks and short answers.

Siep Kroonenberg, Epspdf; pp. 78–80

This article introduces epspdf, a converter between EPS, PostScript and PDF which can be run either via a graphical interface or from the commandline.

Frans Goddijn, David Walden interview; pp. 81–84

There's a treasure of a url (`http://tug.org/interviews/`) on the TUG site, where Dave Walden has collected a number of excellent interviews with key people of the TeX community — a lively "Who's Who" for anyone who has met some TeX luminaries or seen them in action during conferences. The interviews go way beyond the obvious as Dave invites his guests to respond to his lucid questions. The result is a growing collection of significantly detailed portraits of the people who have made the TeX landscape the way it is today. Even if you've known one of the featured people for years, you're certain to discover something interesting about this person that you've never been aware of. Although there *is* an excellent interview with Dave himself (`http://tug.org/interviews/interview-files/dave-walden.html`) on the site, conducted by Karl Berry, we decided to interview Dave for MAPS, exploring some subjects that were mentioned in his online conversation with Berry. You might want to read that interview first to have context for some of the questions and answers in this interview.

Wybo Dekker, The 'isodoc' class; pp. 85–101

The 'isodoc' class can be used for the preparation of letters, invoices, and, in the future, similar documents. Documents are set up with options, thus making the class easily adaptable to user wishes and extensible to other document types.

Geert C.H.M. Verhaag, Creating a Dust-cover in ConTeXt; pp. 102–104

This short article describes how to set up a dust-cover for a book, using the standard features available in ConTeXt.

Taco Hoekwater, TUG 2006 report; pp. 105–108

[Printed in this issue of *TUGboat*, pp. 256= 131 256–136.]

[Received from Wybo Dekker and
Taco Hoekwater]

## *ArsTEXnica* Contents of issue 1 (April 2006)

Editor's note: *ArsTEXnica* is the journal of GuIt, the Italian TEX user group. The journal's web site is `http://www.guit.sssup.it/arstexnica`.

Massimiliano Dominici, Editoriale [From the editor]; p. 3

Brief exposition of the journal's purposes with an overview of the issue's contents.

Gianluca Pignalberi, Intervista a Donald Knuth [Interview with Donald Knuth]; pp. 5–7

The Italian translation of the interview with Donald Ervin Knuth, which first appeared in *Free Software Magazine* no. 7, and was republished in *TUGboat* volume 26, number 6.

[Translation by the author]

Claudio Beccari, I registri *token*: questi sconosciuti [The *token* registers: these mysteries]; pp. 8–13

In this tutorial I show how to use some of TEX's primitive commands and objects, not directly accessible from LATEX, to define useful macros to be used with (pdf)LATEX. These are *token*s and *token* registers, that are seldom, not to say never, treated in LATEX manuals. [Translation by G. Pignalberi]

Enrico Gregorio, Ridefinire i comandi primitivi TEX e applicazioni a LATEX [Redefining TEX's primitive commands and applications in LATEX]; pp. 14–19

LATEX's typesetting is based on several primitive commands directly implemented inside the TEX program. It is possible, if you know what you are doing, to *redefine* such commands. I will discuss some examples of this kind, which will give ideas for other command redefinitions.

[Translation by the author]

Massimo Caschili, Semplici figure con l'ambiente picture [Simple figures using the picture environment]; pp. 20–28

A short guide on how to use picture, the LATEX standard environment for creating simple but effective figures.

[Translation by G. Pignalberi]

Gustavo Cevolani, Norme tipografiche per l'italiano in LATEX [The typographic rules of the Italian language, as applied to LATEX]; pp. 29–42

This article briefly describes the Italian language's main typesetting rules, to be followed when typesetting general purpose articles, theses and books. For each rule discussed, it shows how to apply the rule in LATEX. Perhaps the article could be useful to users of other typesetting programs also.

[Translation by G. Pignalberi]

## *Biuletyn GUST* 22 (2005), 23 (2006)

Editor's note: *Biuletyn GUST* is the publication of GUST, the Polish language TEX user group. Their web site is `http://www.gust.org.pl`.

### *Biuletyn GUST* 22, 2005

Paweł Jackowski, Ciekawe pętle i iteracje na drugą nóżkę [Interesting loops and iterations]; pp. 3–6

Tomasz Łuczak, SlaX-TL — budowa, rozwój i wykorzystanie w praktyce [SlaX-TL — structure, development and usage]; pp. 7–11

Wojciech Myszka, LATEX a logotyp Politechniki Wrocławskiej [LATEX and the Wrocław University of Technology logotype]; pp. 12–16

The authorities of the Wrocław University of Technology (WUT) has published Guidelines for Use of the University logotype. The guide contains not only the history and very formal description of the WUT's logo but also examples of acceptable and unacceptable use and templates. The question for LATEX users are: 'how to use TEX tools for preparing templates of documents following the guide?'; 'how to use Pantone (spot) colors and prepare material for printers?'; 'how to choose and prepare optimal LATEX slide environments following PowerPoint templates?'; and 'how to mimic all these Office tools?' The author has prepared a web environment using (pdf)LATEX for generating stationery (business cards, letterhead, envelope). Some questions still remain open — for example, an elegant way to change page layout between the first and next pages.

Jean-Michel Hufflen, MlBibTEX in Scheme; pp. 17–22

We present the main functions of MlBibTEX's implementation using Scheme. In particular, that allows us to see how the modules are organised and how to run the different parts of MlBibTEX step by step. Let us recall that MlBibTEX deals with several data formats (syntaxes w.r.t. TEX, bibliography files, XML) and we show how such coexistence is managed.

Jean-Michel Hufflen, TEX's language within the history of programming languages; pp. 23–32

We connect some representative statements of TEX's language to analogous features belonging to other programming languages, from a historical point of view. Some features that look strange now are explained easily if we consider the era when TEX was created. By comparing programming in TEX with other paradigms, we also show what TEX can do easily and what is tedious for it.

Bogusław Jackowski and Janusz M. Nowacki, Latin Modern fonts: how less means more; pp. 33–39

This article concerns the latest release of the Latin Modern family of fonts. At present the LM family consists of 57 fonts, each containing 665 glyphs on the average, mainly diacritics. For the first time source files for the glyphs were released as well (the LM fonts were developed with the MetaType1 system, which is based on MetaPost).

Tomasz Barbaszewski, Oprogramowanie otwarte. Dlaczego czasem nam nie idzie? [Why it is sometimes hard to succeed with open source software]; pp. 40–45

Krzysztof Leszczyński, Świat parserów [The world of parsers]; pp. 46–52

Andrzej Tomaszewski, Sto pociech i dwieście utrapień z realizacją pomysłów redaktora [Implementing editor's ideas — lots of fun, sometimes even more trouble]; pp. 53–54

Piotr Bolek, IPC w TEXu [IPC in TEX]; pp. 55–58

Taco Hoekwater, METAPOST developments; pp. 59–63

The METAPOST system (by John Hobby) implements a picture-drawing language very much like that of METAFONT except that it outputs Encapsulated PostScript files instead of run-length-encoded bitmaps. METAPOST is a powerful language for producing figures for documents to be printed on PostScript printers, either directly or embedded in TEX documents. It includes facilities for directly integrating TEX text and mathematics with graphics.

Jean-Michel Hufflen, Introduction to XSLT; p. 64

We propose a didactic demonstration of XSLT, the language of transformations used for XML texts. We use the `xsltproc` program, built via the `libxml2` library. Both are written using the C programming language and are parts of the Gnome project. Both are running on Windows and Linux, but our demonstration is performed on the latter.

David Kastrup, The `bigfoot` bundle for critical editions; pp. 65–70

The LATEX package bigfoot and supporting packages solve many of today's problems occurring in the contexts of single and multiple blocks of footnotes, and more. The main application is with philological works and publications, but simpler problems can be solved painlessly as well without exercising all of the package's complexities. For other problems not yet tackled in this area, a solid framework is provided.

David Kastrup, Designing an implementation language for a TEX successor; p. 71

Managing the complexity of TEX's codebase is an arduous task, so arduous that few mortals can hope to manage the underlying complexity. Its original author's computational roots date back to a time where the maturity and expressive power of existing programming languages was such that he chose to employ the assembly language of a fictional processor for the examples in his seminal work *The Art of Computer Programming*. In a similar vein, TEX is written in a stripped-down subset of a now-extinct Pascal dialect. Current adaptations of the code base include more or less literal translations into Java (NTS and exTEX), C++ (the Omega-2.0 codebase), mechanically generated C (Web2C) and a few others. In practically all currently available cases, the data structures and control flow and overall program structure mimic the original program to a degree that again requires the resourcefulness of a highly skilled programmer to manage its complexity. As a result, almost all of those projects have turned out to be basically single-person projects, and few projects have shown significant progress beyond providing an imitation of TEX. It is the persuasion of the author that progressing significantly beyond the state of the art as represented by TEX will require the expressiveness and ease of use of a tailor-made implementation and extension language. Even a language as thwarted as Emacs Lisp has, due to its conciseness, rapid prototyping nature, extensibility and custom data types and its coevolution with the Emacs editor itself, enabled progress and add-ons reaching far beyond the original state as conceived by its original authors. This talk tries to answer the question what basic features an implementation platform and language for future typesetting needs should possess.

Barbara Beeton, Karl Berry, David Carlisle, Taco Hoekwater, David Kastrup, Bogusław Jackowski, Krzysztof Leszczyński, Frank Mittelbach, Petr Olšák, Bernd Raichle, Martin Schroeder, Philip Taylor, Pearls of TEX programming; pp. 72–79

Published in *TUGboat* 26:3.

### *Biuletyn GUST* 23, 2006

Jonathan Kew, The XeTEX project: typesetting for the rest of the world; pp. 3–8

This paper will introduce the XeTEX project, an extension of TEX that integrates its typesetting capabilities with the Unicode text encoding standard, supporting all the world's scripts, and with modern font technologies provided by today's operating systems and text layout services.

XeTEX offers the potential to be "TEX for the rest of the world" in several senses, as will be discussed and demonstrated. Much of the intimidating complexity of managing a TEX installation — in particular, the process of installing and using new fonts — is eliminated by XeTEX's integration with the host operating system's font management. This greatly reduces the "barrier to entry" into the TEX world for many non-technical users, and provides a richer and more flexible typographic environment.

Because XeTEX is based on Unicode, the universal character encoding standard, and uses OpenType and AAT layout features in modern fonts to support complex non-Latin writing systems, it can work with Asian, Middle Eastern, and other traditionally "difficult" languages just as readily as with European languages.

XeTEX was initially designed and implemented for Mac OS X, leveraging some key technologies available on that platform. However, this meant it was available only to a fairly small minority of potential users. However, with the introduction of XeTEX for Linux, the benefits of XeTEX become available to a new and wider community of users.

Bogusław Jackowski and Janusz M. Nowacki, Rodzina fontów Latin Modern [The Latin Modern font family]; pp. 9–12

The Latin Modern family of fonts is a collection of fonts, based on Donald E. Knuth's Computer Modern family. At present, Latin Modern fonts are released in the PostScript Type 1 and OpenType formats; METAPOST (MetaType1) sources are also available.

The main feature of the Latin Modern fonts is its rich collection of diacritical characters, covering all European Latin scripts as well as some other languages that use Latin alphabets, e.g., Vietnamese.

The aim of the Latin Modern project was to replace (but only as the default) a plethora of various relatives of the Computer Modern fonts, such as the CS, PL and VN fonts.

We hope that the presentation will convince at least Polish TEX users that the LM fonts are a good replacement for the PL fonts at least in MeX and only as the default.

Hans Hagen and Jerzy B. Ludwichowski and Volker RW Schaa, The new font project; pp. 13–14

In this short presentation, we will introduce a new project: the LM-ization of the free fonts that come with TEX distributions. We will discuss the project objectives, timeline and cross-LUG funding aspects.

Jerzy B. Ludwichowski, GUST font licenses; pp. 15–18

For some time the problem of font licenses was discussed at the BachoTEX conferences and in various mail exchanges. The approach presented here tries to address the following issues: first, making sure that fonts developed for the TEX world, where backwards compatibility is very important, will not be broken by "uncontrolled" modifications and second, addressing the issue of the notion of font source code files. As a result, two GUST font licenses were formulated.

Petr Olšák, OFS — A macro package to manage your fonts; pp. 19–30

OFS (Olsak's Font System) gives you a possibility to keep track of your fonts; especially if you have many fonts. It provides tools for making font catalogues, a comfortable user environment for font selection etc. The OFS was presented at EuroTEX 2003 (Brest, France) but many new features were implemented in 2004. This article presents the latest version of this package.

Taco Hoekwater, METAPOST: terminally ill or just playing dead?; pp. 31–34

In recent years, there is evidence of a renewed interest in the use of METAPOST for various drawing tasks. Simultaneously, it seems that just about every METAPOST user runs into some kind of limitation that makes the use of METAPOST far from ideal for the proposed task.

The diagnosis we have to make is whether these symptoms indicate a fatal disease in the program, or if they are only idiosyncrasies and scratches that can be cured with some therapy and a few band-aids.

Halina Wątróbska and Ryszard Kubiak, Od XML-a do TEX-a, używając Emacsa i Haskella [From XML to TEX, using Emacs and Haskell]; pp. 35–39

A bi-language Old Church Russian-Polish dictionary is being created at the University of Gdańsk. The dictionary is based on a relic of Old Slavonic writing, called Izbornik of the XIII century. All word forms from the relic are translated into Polish.

The whole dictionary was written in an XML notation. Specific features of the XML tagging applied as well as a collection of Haskell programs for processing the material of the dictionary are discussed in the article. The programs assist in building of the dictionary using Emacs, serve to analyse its various aspects and to convert it to the language of TEX.

Bogusław Jackowski and Marcin Woliński, Prolegomena do fenomenologii parametrycznego behawioru TEX-a [Prolegomena to the phenomenology of the parametric behaviour of TEX]; pp. 40–42

TEX users certainly encouter situations where TEX surprises them. Of course, TEX wizards may never be surprised, but they are not addressed by our presentation. Our aim is to show a few such surprising — at least at the first glance — cases, without detailed analysis. It is worthy of knowing that such things may happen in the realm of TEX.

Michał Wronka, Wersjonowanie dokumentów TEX-owych w pracy samodzielnej i grupowej [Versioning of TeX documents in individual and group work]; pp. 43–46

Using version control in TEX typesetting gives users a new range of possibilities. Tracking changes in documents to begin with, followed by synchronization across multiple machines, coordination of group work and managing simultaneous versions. In this article, I show how to adapt Subversion for use in typesetting with TEX.

Joanna Ryćko, Typografia dla początkujących; pp. 47–52

As we all know, hours and hours can be spent talking and listening about typography. But on this occasion I will only write about absolute basics of the subject and this also in a telegraphic manner. The aim of the article is to show how to construct simple and typographically correct documents with LATEX.

Joanna Ryćko, Minimalny przykład [How to report (LA)TEX problems: a minimal example and other rules]; pp. 53–57

This is a Polish translation of the German text written by Christian Faulhammer. It shows how to prepare a minimal example, which can be sent to a newsgroup while asking a question about LATEX. The original text in German and its English translation can be found at `http://www.minimalbeispiel.de`.

Wojciech Myszka, Jak przeżyć w nieprzyjaznym środowisku WYSIWYG [How to survive in a hostile WYSIWYG environment]; pp. 58–62

Hans Hagen, LuaTEX: Howling to the moon; pp. 63–68

Occasionally we reach the boundaries of TEX and programming then becomes rather cumbersome. This is partly due to the limitations of the typesetting engine, but more important is that a macro language is not always best suited for the task at hand.

Jean-Michel Hufflen, Advanced techniques in XSLT; pp. 69–75

This talk focus on some advanced techniques used within XSLT, such as sort procedures, keys, interface with identifier management, and priority rules among templates matching an XML node. We recall how these features work and propose some examples, some being related to bibliography styles. A short comparison between XSLT and nbst, the language used within BibTEX for bibliography styles, is given, too.

Jean-Michel Hufflen, BibTEX, mlBibTEX and bibliography styles; pp. 76–80

The first part of this talk about BibTEX will focus on some difficult points related to syntax of bibliography files, e.g., the specification of person and organisation names. In addition, we show how some successors of BibTEX (BibTEX8, Bibulus, mlBibTEX) improve them. In a second part, we explain how bibliography styles are built. Some demonstrations of the BibTEX program are given as part of this talk, and some technical points could be made clearer by using functions belonging to mlBibTEX.

Gabriela Grusza, BibTEX jako narzędzie automatyzujące pracę z bibliografią [BibTEX as a tool for automating tasks around bibliographies]; pp. 81–87

This article is designed for TEX beginners. It presents bases of BibTEX usage including its advantages. Some of attention is given to bib styles with extended discussion about jurabib package and program makebst. Besides, some tools that simplify the creation of the file containing the bibliographic data (Emacs, JabRef, Tkbibtex) are introduced.

Tomasz Łuczak, TEX w biurze i jego wykorzystanie również przez nie-TEXników [TEX in the office, and for non-TEXies]; pp. 88–90

It is possible to deploy TEX in an office for preparing typical documents by employees who have not even have heard of TEX.

A simple and functional user interface allows for a wider use of LATEX documents in daily correspondence, offers or contracts. Batch processing allows combining with other programs (as printing support through pdf) and wrapping with different interfaces (GUI as a stand-alone application or a browser interface, both using TEX in the background).

Besides dedicated programs I will also mention the LyX editor which allows for an easy transition from the WYSIWYG world to the world of pretty documents. When describing the TEX-nical part of these solutions I will also present some observations regarding the "soft" part of the deployment (the employees attitude towards TEX), i.e., why it became a success.

Robert Bialik, LATEX w środowisku naukowym. Spostrzeżenia, uwagi i propozycje [LATEX in the research community]; pp. 91–92

How to convince people to use the TEX environment?! "Arguments versus habits" — a few words about introducing TEX to academics. During the presentation, results of an "academics attitude towards TEX" survey will be presented.

Jacek Kmiecik and Marek Wójtowicz, Z TEX-em w tle [With TEX in the background]; pp. 93–95

Wojciech Birula, Czy TEX polubi CATy? [TEX and computer aided translation]; pp. 96–97

The article briefly describes some problems encountered while trying to use CAT programs in the process of translating TEX files. The author's experience concerns mainly LATEX, however presented problems are similar for other TEX formats.

Maciej Jan Głowacki, LiTeX: łatwość pierwszego kroku [LiTeX: a TEX micro-distribution]; pp. 98–98

LiTeX is a new project aimed at reducing the size and ensuring ease of use of the TEX system in the Linux system environment for PC machines (i586). The set of packages was designed primarily with TEX beginners, typesetting in the Polish language, in mind: it contains amongst others the newest versions of all the Polish and polonized fonts which are available with a license allowing free usage and copying. A complete and clear Polish language documentation constitutes an integral part of the distribution. LiTeX works completely independently of other TEX installations, e.g., it does not use the TDS. Instead, it is firmly set within the Linux Standard Base.

Piotr Bolek and Jakub Kulesza, Gentoo — powrót do źródeł [Gentoo Linux: back to sources]; pp. 99–103

Gentoo Linux is a good and useful source-based Linux distribution. The presentation will cover the following topics: The history of Gentoo Linux and the history of open source and the Linux system; facts and numbers, popularity and aplications of Gentoo Linux; the reasons of growing popularity of Gentoo Linux; advantages and flaws of Gentoo Linux; Gentoo and TEX.

Jean-Michel Hufflen, Writing structured and semantics-oriented documents: TEX vs. XML; pp. 104–108

Using XML-like syntax for documents gives them a tree structure, inducing a notion of structured document. Defining domain-dependent tags introduces a notion of semantics-oriented writing. These two points result in a new view about document production. In fact, they have already existed within TEX, but in another shape. This article aims to point out these notions and the differences between them. It ends with some proposals about the evolution of the tools belonging to TEX's world.

Przemysław Scherwentke, Te nieszczęsne wiszące litery [Those wretched at-end-of-line conjunctives]; pp. 109–110

A set of macros to automate some Polish typesettig rules is presented. In particular, the problem of single characters at the end of a line is solved.

Jerzy B. Ludwichowski, Bogusław Lichoński, Tomasz Przechlewski and Stanisław Wawrykiewicz, Edukacyjny portal GUSTu [The open educational GUST portal]; pp. 111–112

Hans Hagen, Taco Hoekwater, Bogusław Jackowski, Paweł Jackowski, Frank Mittelbach, Bernd Raichle, Piotr Strzelczyk, Enjoy TEX Pearls diving!; pp. 113–121

This year's perils of TEX programming will be presented by a team led by Paweł Jackowski.

[Received from Tomasz Przechlewski]

# Calendar

## 2006

Dec 4 – 7 <XML2006>, Boston, Massachusetts. For information, visit http://2006.xmlconference.org/.

Dec 8 – Jan 24 Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. Special Collections, Michigan State University Libraries, East Lansing, Michigan. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

## 2007

Jan 12 Type Directors Club typography and typeface design competitions, deadline for entries. For information, visit http://www.tdc.org/calls.

Feb 1 **TUG Election:** Deadline for nominations
For information and forms, visit http://www.tug.org/election.

Mar 7 – 9 DANTE 2007, 36th meeting, Westfälische Wilhelms-Universität, Münster, Germany. For information, visit http://www.dante.de/dante2007.

Feb 9 – Mar 18 Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. Utah Museum of Fine Arts, Salt Lake City, Utah. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

Mar 24 – 25 First international ConTEXt User Meeting, Epen, The Netherlands. For information, visit http://context.aanhet.net/epen2007.

Apr 9 – May 20 Guild of Book Workers 100th Anniversary Exhibition: A traveling juried exhibition of books by members of the Guild of Book Workers. Branford P. Millar Library, Portland State University, Oregon. Sites and dates are listed at http://palimpsest.stanford.edu/byorg/gbw.

Apr 28 – May 2 17th EuroTEX Conference + 15th BachoTEX Conference = EuroBachoTEX 2007, Bachotek, Poland. For information, visit http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007.

**TUG 2007**
**Practicing TEX,**
**San Diego, California.**

Jul 17 Workshops (free for attendees).

Jul 18 – 20 The 28th annual meeting of the TEX Users Group. For information, visit http://www.tug.org/tug2007.

Aug 5 – 9 SIGGRAPH 2007, San Diego, California. For information, visit http://www.siggraph.org/s2007/.

Aug 6 – 10 *Extreme* Markup Languages 2007, Montréal, Québec. For information, visit http://www.extrememarkup.com/extreme/.

Aug 28 – 31 ACM Symposium on Document Engineering, University of Manitoba, Winnipeg, Canada. For information, visit http://www.documentengineering.org/.

Sep Association Typographique Internationale (ATypI) annual conference, Brighton, UK. For information, visit http://www.atypi.org/.

*Status as of 25 November 2006*

For additional information on TUG-sponsored events listed here, contact the TUG office (+1 503 223-9994, fax: +1 206 203-3960, e-mail: office@tug.org). For events sponsored by other organizations, please use the contact address provided.

An updated version of this calendar is online at http://www.tug.org/calendar/.

Additional type-related events are listed in the Typophile calendar, at http://www.icalx.com/html/typophile/month.php?cal=Typophile.

# Invitation to EuroTEX 2007

EuroTEX 2007, April 28th until May 2nd, 2007, will be organized jointly by CSTUG, the Czechoslovak TEX Users Group and GUST, the Polish TEX Users Group, at Bachotek, near Brodnica, in the north-east of Poland. This is the place where the annual GUST BachoTEX conferences are organized yearly since 1992. EuroTEX 2007 will also be the XV BachoTEX, hence you might find it referred to as the EuroBachoTEX 2007 conference.

We are trying to make it easier for you to get there: if justified by the number of pasengers, two EuroTEX busses might be organized: one coming from the south, from the Czech Republic or perhaps even from Hungary, and one from the west, from Holland. For details watch the conference site.

The preliminary motto of the conference is

*TEX: Paths to the Future*

It seems to be justified by the recent developments which address the current shortcomings of our beloved system. These developments will be presented during the conference and should make it very interesting. In no particular order, they are:

- new pdfTEX release,
- METAPOST v. 1.1,
- a batch of new font familes from the new project called TEX Gyre,
- a working version of Omega 2,
- LuaTEX,
- XeTEX, and
- many more.

Watch the conference site:

`http://www.gust.org.pl/BachoTeX/EuroBachoTeX2007`

where you also can find a Google Maps pointer to the conference location. The site is going to be updated as new information becomes available.

Please consider to contribute papers to make EuroBachoTEX 2007 even more interesting. Dates for abstracts and paper submissions will be published soon at the conference web page.

And, of course: put this event into your calendar and then come and join the TEXies from around Europe and the world. You can not afford to miss it!

Jerzy Ludwichowski
(for the Organizing Committee)

(These reports were written for *The PracTEX Journal* issue 2006-4; we thank the authors and editors for allowing their inclusion here also. The online version at http://tug.org/pracjourn also includes photos, which we regretfully had to omit. *Ed.*)

## The 3rd Annual GuIT (Italian TEX Users) Meeting

— Onofrio de Bari, GuIT vice president

- The GuIT06 site, with additional information and photos: http://www.guit.sssup.it/GuITmeeting/2006/2006.en.php.
- A podcast of the conference, provided by Kaveh Bazargan of River Valley Technologies: http://web.mac.com/kaveh1000/iWeb/GUIT_2006/Podcast/Podcast.html.

The third annual meeting of GuIT, the Italian TEX User Group, was held on 21st October 2006 in Pisa, Italy. Because of all the activities since GuIT was founded in 2004, we are pleased to see ourselves rapidly reaching the level of interest that other European national user groups receive in their countries. The founding of GuIT with a charter, the publication of our journal, *Ars TEXnica*, and the annual meetings are the result of very intense but gratifying work. In particular, we are very happy to receive many requests from Italian users to join GuIT and help with our efforts.

The annual meeting is always a special day, giving GuIT members the chance to abandon their "net anonymity" and to meet each other in person to discuss LATEX, its use and its future, both in Italy and internationally. The talks at this year's meeting covered a wide range of topics, and were chosen to appeal to all levels of users.

The conference started with an introductory speech by Lance Carnes, editor of *The PracTEX Journal*, about PracTEX and what it means for regular TEX users.

Next was Enrico Gregorio who spoke about category codes; in particular, he described an application he developed for entering math expressions by redefining LATEX category codes and commands.

Gustavo Cevolani introduced a review of several methods to create booklets using LATEX packages. Some methods were custom-developed for a specific purpose, while others were adaptations of existing methods to obtain the desired result.

A paper on LATEX tables by Lapo Mori, who could not attend in person, was presented by Maurizio Himmelman. He gave a detailed description of the problems that can arise when creating tables, and solutions supplied by numerous packages to solve these issues.

Kaveh Bazargan, of River Valley Technologies, gave a talk about mimicking the vertical grid spacing common in traditional typesetting with LATEX. This was a very interesting subject, presented with "eye-candy" slides which caught everyone's attention.

The morning session ended with a talk by Jean-Michel Hufflen who talked about MlBIBTEX, a reimplementation of BIBTEX. The goal of this project is to improve BIBTEX to work better in a multilingual environment.

In the afternoon it was time for me to give my talk, about GNU Emacs. I tried my best to make people aware of basic GNU Emacs editing features for LATEX, and continued by analyzing features of sophisticated TEX and LATEX environments such as AUCTeX and preview-latex.

Salvatore Palma presented the use of LATEX to produce interactive mathematics tests for high schools. His results are impressive, and this subject will be developed further in the future.

The final three talks were about critical editions using LATEX. In the first talk Jeronimo Leàl presented a course given to university students about critical editions. The next two presentations were about the Maurolico Project, oriented to critical editions of the works of the Italian mathematician, Francisco Maurolico. Pier Daniele Napolitani, head of the project, and Massimiliano Dominici supplied a general introduction to MauroTEX (the language built for the project) and a review of its features and developments. Roberta Tucci described her experience using MauroTEX for the critical edition of a single mathematical work.

Lastly, GuIT president Maurizio Himmelmann closed the meeting with some brief remarks and thanked the speakers and organizers.

The level of attendance at the meeting was extremely satisfying. During the day about eighty people attended the conference. Thirty attendees were from outside Tuscany, who came by car, train or plane to attend the event; among them were Claudio Beccari (one of the first TEX users in Italy), Gianluca Gorni, and many others who will forgive me if I don't mention their names here.

As vice president of GuIT, I am very pleased to say that the scientific level of this year's meeting was extremely high. Most of all, speaking for all GuIT members, I want to say thanks to Lance Carnes, editor of *The PracTEX Journal* and Kaveh Bazargan of River Valley Technologies. Their contributions to the meeting were invaluable, not only for their presentations but also for their personal interest in our evolving user group.

I cannot omit, of course, to say thanks to the Sant' Anna School of Advanced Studies who provided the meeting facilities, and in particular to Prof. Giulio Bottazzi of the Laboratory of Economics and Management, without whose support this meeting would not have been possible.

Last but not least, as is often said in such cases, a big "thank you" to the GuIT staff and to the attendees. We have received and still receive emails of congratulation which encourage us greatly in what we are trying to do for the Italian and international TEX community. Our hope is to make GuIT an organization which will improve and grow in coming years.

### UKTUG sponsors day of LaTeX

— Charles Goldie, UKTUG Committee Member

- LaTeX Workshop slides and handouts: `http://uk.tug.org/events/workshops/living-and-working-with-latex`

The UK TeX Users' Group (UKTUG) held a workshop "Living and Working with LaTeX" on October 20, 2006. This was something of a renewal event for the Group, which in recent years has found it hard to find volunteer energy enough to mount any major event. The breakthrough came with the realisation that it would make sense to outsource the practical aspects of the day, leaving the volunteer organisers to concentrate on where knowledge of the TeX world was needed, namely deciding on a theme and recruiting presenters relevant to it. The outcome was a meeting run as far as practical matters were concerned by the London Mathematical Society, at its new conference facility in central London.

The workshop focussed on using LaTeX to write technical documents, theses, books and articles. A major theme was using LaTeX better, for example to make it easier to collaborate and to re-use and revise documents. Centrepieces of the day were three major presentations:

- Peter Flynn: "Sorry, Professor, the dog ate my thesis: how to expect the unexpected when using LaTeX",
- Nicola Talbot: "Writing a thesis in LaTeX: hints, tips and advice",
- Jonathan Fine: "Avoiding problems, solving problems, asking for help".

About 55 attended, essentially filling the room used for the plenary sessions. Groups on particular topics formed naturally in breakout sessions in between the presentations. Participants seemed to be a mixture of relative newcomers and experienced TeX-setters. Most found that what they gained from the workshop was commensurate with their input: asking a question or making a point forces you to formulate your thoughts clearly; then back come reactions, help, collateral information and the identification of others with similar concerns.

Presentations and background materials are on the web under `http://uk.tug.org/events/`. UKTUG expresses its heartfelt thanks to the three presenters for their contribution to the success of the day. The Group was encouraged by the outcome, and hopes to mount a further event or events on a similar model in the future.

## Institutional Members

Aalborg University, Department of Mathematical Sciences, *Aalborg*, *Denmark*

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*

Marquette University, Department of Mathematics, Statistics and Computer Science, *Milwaukee*, *Wisconsin*

Masaryk University, Faculty of Informatics, *Brno*, *Czechoslovakia*

Moravian College, Department of Mathematics and Computer Science, *Bethlehem*, *Pennsylvania*

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*

United States Environmental Protection Agency, *Narragansett*, *Rhode Island*

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*

Vanderbilt University, *Nashville*, *Tennessee*

# TeX Users Group Membership Form 2006

## Promoting the use of TeX throughout the world.

*mailing address:*
P. O. Box 2311
Portland, OR 97208-2311 USA

*shipping address:*
1466 NW Naito PKWY, Suite 3141
Portland, OR 97209-2820 USA

| | |
|---|---|
| *phone:* | +1 503-223-9994 |
| *fax:* | +1 206-203-3960 |
| *email:* | `office@tug.org` |
| *web:* | `http://www.tug.org` |

| | |
|---|---|
| *President* | Karl Berry |
| *Vice-President* | Kaja Christiansen |
| *Treasurer* | David Walden |
| *Secretary* | Susan DeMeritt |
| *Executive Director* | Robin Laakso |

TUG membership rates are listed below. Please check the appropriate boxes and mail the completed form with payment (in US dollars) to the mailing address at left. If paying by credit/debit card, you may alternatively fax the form to the number at left or join online at `http://tug.org/join.html`. The web page also provides more information than we have room for here.

**Status** (check one) □ New member □ Renewing member

**Automatic membership renewal** in future years □
Using the same payment information; just contact office to cancel.

| | Rate | Amount |
|---|---|---|
| □ **Early bird membership** for 2006<br>After May 31, dues are $85. | $75 | ______ |
| □ **Special membership** for 2006<br>You may join at this special rate ($55 after May 31) if you are a senior (62+), student, new graduate, or from a country with a modest economy. Please circle accordingly. | $45 | ______ |
| □ **Subscription** for 2006 (non-voting) | $95 | ______ |
| □ **Institutional membership** for 2006<br>Includes up to eight individual memberships. | $500 | ______ |
| □ **Don't ship any physical benefits** (*TUGboat*, software)<br>*TUGboat* and software distributions are available electronically. | deduct $20 | ______ |
| □ **Send last year's TeX Collection 2005** right away<br>*Instead of* this year's TeX Collection 2006. | n/a | |
| □ **Send CTAN 2006 on CD** (shipped on DVD to everyone) | $15 | ______ |
| **Purchase last year's materials** | | |
| □ *TUGboat* volume for 2005 (3 issues) | $20 | ______ |
| □ TeX Collection 2005<br>2 CD's & 1 DVD with proTeXt, MacTeX, TeX Live, CTAN. | $20 | ______ |
| □ CTAN 2005 CD-ROMs | $15 | ______ |
| **Voluntary donations** | | |
| □ General TUG contribution | | ______ |
| □ Bursary Fund contribution | | ______ |
| □ TeX Development Fund contribution | | ______ |
| □ CTAN contribution | | ______ |
| □ LaTeX 3 contribution | | ______ |
| | **Total $** | ______ |

*Tax deduction:* The membership fee less $35 is generally deductible, at least in the US.
*Multi-year orders:* To join for more than one year at this year's rate, just multiply.

**Payment** (check one) □ Payment enclosed □ Visa/MasterCard/AmEx

Account Number: ______________________ Exp. date: ________

Signature: ______________________

**Privacy:** TUG uses your personal information only to send products, publications, notices, and (for voting members) official ballots. TUG does not sell or otherwise provide its membership list to anyone.

Electronic notices will generally reach you much earlier than printed ones. However, you may choose not to receive any email from TUG, if you prefer.

□ Do not send me any TUG notices via email.

Name ______________________

Department ______________________

Institution ______________________

Address ______________________

______________________

City ______________ State/Province ______________

Postal code ______________ Country ______________

Email address ______________________

Phone ______________ Fax ______________

Position ______________ Affiliation ______________

# TEX Consultants

The information here comes from the consultants themselves. We do not include information we know to be false, but we cannot check out any of the information; we are transmitting it to you as it was given to us and do not promise it is correct. Also, this is not an official endorsement of the people listed here. We provide this list to enable you to contact service providers and decide for yourself whether to hire one.

TUG also provides an online list of consultants at `http://tug.org/consultants.html`. If you'd like to be listed, please fill out the form at `https://www.tug.org/consultants/listing.html` or email us at `consult-admin@tug.org`. To place a larger ad in *TUGboat*, please see `http://tug.org/TUGboat/advertising.html`.

**Kinch, Richard J.**
7890 Pebble Beach Ct
Lake Worth, FL 33467
561-966-8400
Email: `kinch (at) truetex.com`
Publishes TrueTEX, a commercial implementation of TEX and LATEX. Custom development for TEX-related software and fonts.

**Martinez, Mercè Aicart**
C/Tarragona 102 $4^o$ $2^a$
08015 Barcelona, Spain
+34 932267827
Email: `m.aicart (at) menta.net`
Web: `www.edilatex.com/index_eng.html`
We provide, at reasonable low cost, TEX and LATEX typesetting services to authors or publishers world-wide. We have been in business since the beginning of 1990.

**MCR Inc.**
731 Beta Drive #G
Mayfield Village, OH 44143
(440) 484-3010; fax: (440) 484-3020
Email: `sales (at) mcr-inc.com`
Web: `www.mcr-inc.com`
Contract typesetting/printing services.

**MicroPress Inc.**
68-30 Harrow Street
Forest Hills, NY 11375
+1 718-575-1816; fax: +1 718-575-8038
Email: `support (at) micropress-inc.com`
Web: `www.micropress-inc.com`
Makers of VTEX, fully integrated TEX system running on Windows. VTEX system is capable of one-pass output of PDF, PS, SVG and HTML; VTEX IDE includes Visual Tools for writing equations, function plots and other enhancements and a large number of fonts, many not available elsewhere. Makers of many new and unique mathematical font families for use with TEX, see `http://www.micropress-inc.com/fonts`. Makers of microIMP, a fully WYSIWYG LATEX-based Word Processor. microIMP supports TEX and AMSTEX math, lists, tables, slides, trees, graphics inclusion, many languages and much else — all without any need for knowing TEX commands, see `http://www.microimp.com`. See our web page for other products and services. Serving TEX users since 1989.

**Peter, Steve**
310 Hana Road
Edison, NJ 08817
+1 (732) 287-5392
Email: `speter (at) dandy.net`
Specializing in foreign language, linguistic, and technical typesetting using TEX, LATEX, and ConTEXt, 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. I have extensive experience in editing, proofreading, and writing documentation. I also tweak and design fonts. I have an MA in Linguistics from Harvard University and live in the New York metro area.

**Veytsman, Boris**
2239 Double Eagle Ct.
Reston, VA 20191
(703) 860-0013
Email: `borisv (at) lk.net`
Web: `http://users.lk.net/~borisv`
TEX/LATEX consulting. Integration with databases, full automated document preparation systems, conversions and more.

**TUG**BOAT Volume 27 (2006), No. 2 TUG 2006 Conference Proceedings

## Table of Contents (ordered by difficulty)

### Introductory

### Intermediate

### Intermediate Plus

### Advanced

### Contents of other TeX journals

### Reports and notices