

Baskerville

The Annals of the UK T_EX Users' Group

Editor: Editor: Sebastian Rahtz

Vol. 4 No. 3

ISSN 1354-5930

February 1998

Articles may be submitted via electronic mail to `baskerville@tex.ac.uk`, or on MSDOS-compatible discs, to Sebastian Rahtz, Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, to whom any correspondence concerning *Baskerville* should also be addressed.

This reprint of *Baskerville* is set in Times Roman, with Computer Modern Typewriter for literal text; the source is archived on CTAN in `usergrps/uktug`.

Back issues from the previous 12 months may be ordered from UKTUG for £2 each; earlier issues are archived on CTAN in `usergrps/uktug`.

Please send UKTUG subscriptions, and book or software orders, to Peter Abbott, 1 Eymore Close, Selly Oak, Birmingham B29 4LB. Fax/telephone: 0121 476 2159. Email enquiries about UKTUG to `uktug-enquiries@tex.ac.uk`.

Contents

I	Editorial	3
1	<i>Baskerville</i> articles needed	3
2	A pox on logos	3
3	CD-ROM 4T _E X: a T _E X workbench for DOS PCs	3
4	Macro name markup in <i>Baskerville</i> submissions	4
4.1	Colophon	4
II	EuroT _E X '94.....	5
III	The ease of including graphics in T _E X documents using 4T _E X	6
1	Introduction	6
2	T _E Xcad	6
3	emT _E X and the 'special' commands	7
4	BM2Font	9
5	Hewlett Packard Plotter files and HP2xx	9
6	POSTSCRIPT and GhostScript	10
7	The <code>figures</code> style files	10
8	4T _E X	10
IV	L ^A T _E X 2 _ε standard graphics and colour support.....	13
1	Introduction	13
2	The graphics package	13
2.1	Rotation	13
2.2	Scaling	13
2.3	Graphics inclusion	14
3	The colour package	14
3.1	Using colours	15
3.2	Defining new colours	15
V	The <code>keyval</code> package.....	17
1	Introduction	17
2	Example	17
3	The Internal Interface	18
VI	<i>Baskerville</i> Production — the horror story.....	19
VII	Book reviews.....	21
0.1	Writing & Illuminating & Lettering, by Edward Johnston	21
0.2	Modern Typography, by Robin Kinross	21

VIIIA tutorial on TrueType.....	23
1 Background	23
1.1 Type	23
1.2 Scalable fonts and digital type	23
2 True Type Fonts	24
3 So who will use TT?	25
4 Further reading	25
IX Backslash—Expansion of macros and so forth.....	26
X Malcolm’s Gleanings.....	29
1 Macsyma	29
2 As others see us	29
3 Colour	29
4 Acrobat again	30
4.1 Acrobat in Publishing	30
4.2 pdf or dvi?	32
4.3 Size isn’t important	32
5 Editor nods	32

I Editorial

1 *Baskerville* articles needed

Baskerville really needs material from its readers! Please send your interesting articles to the editor, and delight fellow T_EX users. Please note the following schedule of copy deadlines:

Issue	Submit material for publication	Submit minute notices	last- anticipated post- ing date
4.4	Aug 15	Aug 22	Sep 8
4.5	Oct 17	Oct 24	Nov 10
4.6	Dec 19	Dec 22	Jan 9

Each issue of *Baskerville* will have a special theme, although articles on any T_EX-related subject are always welcome. Contributions on the themes for the remainder of 1994 are eagerly solicited:

- ☞ *Baskerville* 4.4 will be a back to basics special issue on mathematical and tabular typesetting;
- ☞ *Baskerville* 4.5 will try and go beyond T_EX, to see what is on the horizon.
- ☞ *Baskerville* 4.6 will be about font-encoding if past history is anything to go by ...

2 A pox on logos

Does anyone else share my hatred of T_EX-related logos? When Knuth did it with T_EX, it was funny; when Lamport invented L^AT_EX, it was mildly amusing; since then, downhill all the way. These ghastly creatures often don't translate well from CM, they cause problems with indexes, running heads and so on, we can never remember the name of the macros, and they hold T_EX up to ridicule by outsiders. So T_EX can do kerning — far out. If I had my way, I'd ban them all except T_EX itself, and that goes for METAFONT too....

3 CD-ROM 4T_EX: a T_EX workbench for DOS PCs

At the end of 1993 the NTG (the Dutch language oriented T_EX user group) released the package 4T_EX. It consists of a set of 31 diskettes, on which the most important T_EX and L^AT_EX related packages are assembled in well structured modules. 4T_EX is an attempt to integrate all major T_EX related programs in a shell that shields you from the tedious and frustrating job of setting environment variables and program parameters. In 4T_EX all functions are available through simple menus, so profound knowledge of all underlying programs is no longer required (the article in this issue of *Baskerville* discusses this integration in the area of graphics).

To simplify the distribution of 4T_EX, a CD-ROM will be released in at the beginning of June. This CD-ROM should enable you to set up a 4T_EX system with minimal effort.

The CD-ROM will contain at least:

- a fully installed 4T_EX 3.10 workbench, ready for use;
- the complete 4T_EX distribution set;
- the MAPS (NTG's *Minutes and Appendices*) articles on T_EX related topics, mostly in English, of recent years in POSTSCRIPT format;
- all TEX-NL and UK-TEX mailing list discussions of recent years;
- extra documentation, such as:
 - 'The T_EXbook' and 'The METAFONTbook' in T_EX code,
 - T_EX, L^AT_EX and METAFONT introductions,
 - very extensive bibliographies on miscellaneous T_EX related topics in BIBT_EX format,

- indexes to T_EX macros and to files on CTAN (Comprehensive T_EX Archive Network);
- extra programs, such as:
 - GNUplot,
 - GhostScript,
 - dvi utilities,
 - POSTSCRIPT utilities,
 - Polish and Cyrillic T_EX packages,
 - WEB: literate programming.

The CD-ROM will include the updated documentation on 4T_EX, an A5 sized booklet of approx. 120 pages, that explains how 4T_EX works in depth. The price of the CD plus booklet is set at US \$35. It will be available to UKTUG members from mid-June from Peter Abbott; enquiries and reservations may be made to Sebastian Rahtz (spqr@ftp.tex.ac.uk).

4 Macro name markup in *Baskerville* submissions

We are sometimes asked what markup to use articles submitted to *Baskerville*. Following plain L^AT_EX ‘article’ style covers almost all needs, but additional time can be saved by using the following macros in your text, which define common names:

<code>\AmSTeX</code>	<i>AMSTeX</i>
<code>\BV</code>	<i>Baskerville</i>
<code>\BibTeX</code>	<i>BIBTeX</i>
<code>\MakeIndex</code>	<i>MakeIndex</i>
<code>\PS</code>	<i>POSTSCRIPT</i>
<code>\PicTeX</code>	<i>PiCTeX</i>
<code>\SLiTeX</code>	<i>SLTeX</i>
<code>\TUB</code>	<i>TUGboat</i>
<code>\UKTUG</code>	<i>UK T_EX Users’ Group</i>
<code>\ukt</code>	<i>UKTUG</i>

4.1 Colophon

This issue of the journal was created entirely with the test distribution of L^AT_EX 2_ε and printed on a Hewlett Packard LaserJet 4. *Baskerville* is set in ITC New Baskerville Roman and Gill Sans, with Computer Modern Typewriter for literal text. Production and distribution was undertaken in Cambridge by Robin Fairbairns and Jonathan Fine.

II EuroT_EX '94

EuroT_EX '94
Włodek Bzyl,
Department of Mathematics,
University of Gdansk,
Poland
eurotex@halina.univ.gda.pl

EuroT_EX '94 will take place at Sobieszewo on an idyllic island off the coast of Gdansk in Poland. The conference will run from Monday September 26th to Friday September 30th, and the *maximum* cost (based on two persons sharing) will not exceed \$260.00 / £175.00 / DM 450.00. Those arriving early on Monday will be able to take part in a guided tour of the old town of Gdansk, whilst Tuesday to Thursday will be jam-packed with talks and tutorials on T_EX and related topics.

All delegates will be accommodated in a single building, and for the whole week will be cut off from civilisation: no distractions, no need to leave the island: everything will be provided. For those unable to sustain the pace, quiet walks along the shore searching for amber will provide the ideal opportunity for therapeutic meditation.

Papers are solicited *now*, and early registration for the conference is advised: with its Central European location and idyllic setting, the conference is expected to attract many delegates. If you are even *thinking* of coming, then you are urged to contact the organisers: we will then add you to the mailing list and keep you posted about any changes in the schedule. Please note the following deadlines:

Confirmed registration: 1 September 1994 (cancellation charged at 50%);

Late registration: 15th September 1994 (no cancellation possible);

Bursaries: as with the TUG meeting at Aston last year, it is hoped to be able to offer financial assistance to delegates who would otherwise be unable to attend; of course, we cannot be sure at this stage that sponsors will be as generous as they were last year, but intending delegates who will need assistance in order to be able to attend should indicate in the space provided the *minimum* bursary which would allow them to be able to attend, and should give clear reasons why they are applying. All applications will be treated in the strictest confidence. Delegates who are in no need of a bursary and who are able to assist others less fortunate are urged to pledge a donation in the space provided.

Tutorials and Courses: It is intended to offer both tutorials (which will take place during the week of the conference proper), and courses (which will take place during the week following the conference); proposed topics include book design and L^AT_EX 2_ε, but no firm decisions have yet been taken on topics, durations or costs. Further details concerning this area will be circulated as soon as they are known, and space has been provided on the provisional registration form (available from the organisers) in which both to indicate an interest in tutorials/courses and to suggest possible topics.

III The ease of including graphics in T_EX documents using 4T_EX

Wietse Dol
Landbouw-Economisch Instituut (LEI-DLO)
P.O.Box 29703
2502 LS Den Haag
The Netherlands
W.Dol@LEI.Agro.nl

[Editor's note: I am grateful to Wietse Dol and Gerard van Nes (editor) for permission to reprint this article from MAPS 93.2, the journal of the Nederlandstalige T_EX Gebruikersgroep.]

1 Introduction

T_EX has been developed with the idea that it should be possible to have a T_EX implementation for every operating system (MSDOS, VMS, VM-CMS, UNIX etc.). Another feature of T_EX is that documents can be freely exchanged between operating systems (because documents are written in standard ASCII). Graphics, however, are machine dependent and the possibility of including graphics in T_EX or L^AT_EX depends on the operating system and the DVI-driver you are using. This means that including graphics in T_EX or L^AT_EX is often not an easy job.

The solution often used for including graphics is including POSTSCRIPT pictures in the document using the `\special` command. The `\special` command is ignored/passed on by the T_EX compiler but the POSTSCRIPT DVI-driver will use the `\special` command to insert the POSTSCRIPT picture at the right place and in the right size in your document. The advantage of this method is that for all operating systems there are POSTSCRIPT DVI-drivers and that POSTSCRIPT files are also written in standard ASCII, therefore you can transport text file and graphics to all operating systems. The disadvantage of this method is that you can *only* include POSTSCRIPT pictures in your document and that you need a POSTSCRIPT printer to produce output. It is not possible to use the screen previewer to view the DVI-file.

When you have a PC there are other ways to incorporate pictures in T_EX documents. Before we will discuss them, we have to know more about the different types of pictures. In principle there are two types of pictures, namely bitmap and vector pictures. A bitmap picture is a matrix with the entries corresponding to points with a colour. The dimension of the matrix specifies the height and width of the picture. Because of the fixed matrix, manipulating the picture is difficult and resizing the picture often leads to undesirable results. However, many MSDOS graphic packages produce pictures in a bitmap format. These bitmap files come in many different types, mostly as a result of different compression and colour encoding techniques. Examples of bitmap pictures are: GIF (CompuServe), TIFF, PCX (PC Paintbrush), BMP (Windows 3.x), IFF (Amiga), LBM (Amiga), IMG (Ventura), CUT (Dr Halo), and PCL (Hewlett Packard).

A *vector* picture is specified by a device-independent mathematical description and is therefore easy to manipulate/resize. However, the problem with vector pictures is that most DVI-drivers cannot handle them. Examples of vector pictures are: HPGL (Hewlett Packard Graphics Language), PS (PostScript), and EPS (Encapsulated PostScript).

In the remainder of this paper we will discuss the computer programs *TeXCad*, *Graphic WorkShop*, *HP2xx*, *BM2Font*, *PCLtoMSP*, and *GhostScript*. We will end this paper by describing how 4T_EX combines the strength of all these programs to incorporate graphics in T_EX documents.

2 T_EXcad

TeXCad is a drawing program written by G. Horn for producing drawings in L^AT_EX documents. It allows the objects available in the L^AT_EX picture environment to be drawn and edited. Its output is a sequence of L^AT_EX picture commands which can be inserted into a L^AT_EX document to generate the drawing. The advantage of this program is that the text font used in the graphic is the same as the text font used in the main text (*i.e.* Computer Modern). Using the L^AT_EX picture environment also makes it possible to compile/print/view these graphics on other operating systems.

The disadvantages are its user unfriendliness as compared to other graphical packages, and the limited set of objects of the \LaTeX picture environment.

\TeXcad is written in *Turbo Pascal V5* and runs on all PC machines. A mouse is not required but strongly recommended. \TeXcad supports the special commands of $em\TeX$ for line drawing (very useful for drawing lines at any angle) but you should not forget to include the style file `emlines2.sty` in the document style declaration of the document. Likewise, if you use *bezier* curves. The problem with these $em\TeX$ specials, however, is that they are machine dependent. For a detailed discussion how to install and use \TeXcad see Horn (1990). For an example of \TeXcad see Figure 5.

3 $em\TeX$ and the ‘special’ commands

The `\special` command is ignored completely by the \TeX compiler but is executed when running the DVI-driver. Not all DVI-drivers can execute all the `\special` commands. The `\special` command is therefore output device dependent and it is ignored when the DVI-driver does not support that specific `\special` command. The advantage of the `\special` commands is that you can use device-dependent instructions to produce output, *e.g.*, you can use a POSTSCRIPT printer to include POSTSCRIPT pictures in your \TeX document.

$em\TeX$ has some `\special` commands to include PCX, MSP (Microsoft Paint) and black and white BMP bitmap pictures. The syntax of the `\special` command is:

```
\special{em:graph [path] <bitmap file>}
```

where `[path]` is an optional path and `<bitmap file>` is an PCX, MSP, or black and white BMP bitmap file. The upper left corner of the graphic file is located at the reference point of a character. Run length encoded BMP files and 4 colour CGA-mode PCX files cannot be used. All non-white pixels of a PCX file are printed (assuming the standard palette). The width of the graphic must not exceed 32760 pixels, the height must not exceed 32766 pixels. The viewer and the printer drivers of the $em\TeX$ package will show and print the bitmap. However, it is not possible to manipulate the picture. This means that different drivers will produce different sized pictures (as a result of the resolution of the device driver and the fixed resolution of the bitmap graphic). For a detailed discussion about `\special` commands see Mattes (1992).

When you want to manipulate the picture (*e.g.* resizing the picture or colour reduction) you can for instance use the shareware program *Graphic Workshop*. *Graphic Workshop* is a program for working with bitmapped graphic files. It will handle most of the popular bitmap formats. *Graphic Workshop* is a simple, menu driven environment which will let you perform the following operations on graphic files:

- View bitmap pictures
- Convert between any two bitmap formats
- Print the bitmap on almost all popular printers
- Dither colour pictures to black and white
- Reverse the colours
- Rotate the picture
- Flip the picture
- Scale the picture to any size
- Select a part of the graphic as a new picture
- Reduce the number of colours
- Sharpen and Soften the picture
- Adjust the brightness, contrast and colour balance of the bitmaps.

When the picture has more than 32 colours and you use the $em\TeX$ special command to include a graphic an error will appear when viewing or printing the picture. Too many colours will result in too black and unfocused pictures and it is much better to use *BM2Font* to incorporate the picture into a \TeX document.

The problem with bitmap files is their fixed dimension, *i.e.* the size of the picture will depend on the DVI-driver used. Suppose the bitmap file `golfer.pcx` has dimensions 550×770 dots. Using the picture and a 300 dpi DVI-driver the picture will have a width of $550/300 = 1.83$ inch = 47mm and a height of $770/300 = 2.57$ inch = 65mm. Other DVI-drivers will result in different sizes. The bitmap file is now printed using:

```
\begin{figure}  
\begin{center}  
\setlength{\unitlength}{1mm}  
\begin{picture}(47,65)
```



Figure 1. This is an example how to use pictures in $em\TeX$



Figure 2. The use of PSFIG to insert a PS picture



Figure 3. This is the file genesis.bmp converted to \TeX fonts

```
\put(0,65){%
\special{em:graph golfer.pcx}}
\end{picture}
\end{center}
\caption{This is an example how to
        use pictures in \emtex}
\end{figure}
```

In this example we use millimeters as the unit of measurement. Normally the picture `golfer.pcx` will be printed from the upper left corner of the picture box, but it should be printed from the lower left corner, therefore we need `\put(0,65)`. The result is shown in Figure 1.

$em\TeX$ also supports a POSTSCRIPT `\special` command, but it will not view or print a POSTSCRIPT graphic file, except when you use a POSTSCRIPT DVI-driver (*e.g.*, `dvips`) to print on a POSTSCRIPT printer.

There are many macro packages that can help you insert POSTSCRIPT files in your document, *e.g.* `psfig`, a macro package written by T.J. Darrell. With the help of a POSTSCRIPT DVI driver, figures are automatically scaled and positioned on the page, and the proper amount of space is reserved. To include a POSTSCRIPT picture, include the `psfig` style at the top of your document:

```
\documentstyle[11pt,psfig]{article}
and, when you wish to include a figure, invoke the macro with, e.g. ,
\begin{figure}
\begin{center}
\psfig{figure=%
        tiger.ps,width=50mm,height=50mm}
\end{center}
\caption{The use of PSFIG to insert
        a PS picture}
\end{figure}
```

and the result is shown in Figure 2.

Note that spaces in the arguments of the macro are not allowed. For a detailed discussion of all possibilities (*e.g.*, rotation, scaling etc.) we refer to Darrell (1992) and Goossens (1993). The `psfig` macros will generate some `\special` commands to claim the correct space and size, and with a POSTSCRIPT DVI driver the picture will be printed correctly.

4 BM2Font

BM2Font is a program written by F. Sowa and is used to convert bitmap pictures to \TeX fonts. These \TeX fonts can be read by the DVI-drivers and are used to view and print pictures. *BM2Font* can convert the following bitmap pictures: PCX, GIF, BMP, IFF, LBM, TIFF, IMG, and CUT. For a detailed discussion how *BM2Font* works and all the possible parameters we refer to the manual. Note that *BM2Font* can produce several \TeX fonts (*i.e.* bitmap fonts (extension .pk) and \TeX font metric files (extension .tfm) and that the bitmap fonts are resolution dependent.

The command syntax of *BM2Font* is

```
bm2font <bitmap file> [options]
```

The result of a command like *bm2font example.pcx* is one or more font files, but also a file called *example.tex*. This file *example.tex* (written to the current directory) uses the picture fonts and defines a macro called `\setexample` (*i.e.* consisting of the the word SET and the filename EXAMPLE (without file extension)). The picture is now produced simply by giving the command `\input example.tex` and the command `\setexample` on the location where you want the picture.

We will end this section with an example. Suppose we have a BMP bitmap file *genesis.bmp* and we want to convert this bitmap to \TeX fonts for the laserprinter (300 dpi). Running *BM2Font*

```
bm2font genesis.bmp -h300 -v300 -m50 -n50
```

will result in one \TeX font metric file (*genesisa.tfm*), one bitmap font (*genesisa.pk*), and the \TeX file *genesis.tex* containing the macro `\setgenesis`. The parameters `-h` and `-v` are the horizontal and vertical resolution of the printer, the parameters `-m` and `-n` are the width and height you want the picture to be in millimeters. To produce the picture in the \TeX document we can use, *e.g.* :

```
\begin{figure}
\centerline{\input genesis
             \setgenesis}
%the file genesis.tex contains
%the macro \setgenesis
\caption{This is the file genesis.bmp converted
         to \TeX\ fonts}
\end{figure}
```

BM2Font gives excellent results and is easy to use. The only disadvantage is that you need to generate picture fonts for every printer you use. For example when you use a 300 dpi laserprinter as well as a 600 dpi POSTSCRIPT printer, *BM2Font* will give you .pk and .tfm files with the same name but for different printers. So you have to do your own book-keeping, and track which font files you need for the printer you are using at the moment.

Note: if the length of the filename of the picture file is eight characters, the last character will be omitted for the construction of the .tfm and .pk files (because of the addition of the font numbers a, b, ...). For instance *scrndump.pcx* will produce *scrnduma.tfm*, *scrnduma.pk*, and *scrndum.tex* and the macro is called `\setscrndum`.

Note: No digits are allowed in the picture filename; *e.g.* , *screen1.pcx* has to be renamed to *screeni.pcx*.

When you have a PCL bitmap file (Hewlett Packard LaserJet and DeskJet graphic output) you can convert this file to a MSP or a PCX bitmap. The conversion is done by using E. Mattes' conversion program *pcltomsp*. After conversion you can use the `\special` command to include the graphic (see above) or the program *BM2Font* to generate \TeX fonts. The syntax is easy, *e.g.*

```
pcltomsp -qop graph.lj graph
```

converts the PCL file *graph.lj* into the PCX file *graph.pcx*, and does not display warnings or the program title.

5 Hewlett Packard Plotter files and HP2xx

Some graphics programs produce graphic files in the *HPGL* format (Hewlett Packard Graphic Language). These are vector pictures specifically made for Hewlett Packard plotters. Because we can only use POSTSCRIPT and certain bitmap pictures in \TeX documents we need to convert such files. The extensions often used for HPGL files are .hpp, .plt, and .hpg and are produced by, *e.g.* , the programs *Matlab*, *Gauss*, and *Harvard Graphics*. To convert HPGL plotter files we use the program *HP2xx*.

HP2xx is a freeware program from H. Werntges and is used to print, view and convert HPGL plotfiles. We refer to the documentation for a detailed discussion of all the possibilities (*e.g.* , rotation, picture size, pencolour, magnification etc.). $\mathcal{A}\TeX$ uses *HP2xx* to convert HPGL files to *PCX* bitmap files and *EPS* (Encapsulated PostScript) files. *HP2xx*

uses no environment variables, it reads and writes the files from the current directory. *HP2xx* supports the 800×600 super VGA modes (e.g., the Tseng ET4000 and the Trident SVGA). *HP2xx* is easy to use and produces excellent quality. You can convert HPGL pictures to MF (METAFONT format), CAD (to be used with *T_EXcad*), EM (*emT_EX* specials), EPIC (the Enhanced Picture style), IMG-, PBM-, PCL- and PCX-bitmaps and EPS PostScript pictures.

Suppose we have a file `test.hpg`. We can convert this file to a 300 dpi PCX bitmap file `test.pcx` with height 100 mm (width is automatically calculated) using the command

```
hp2xx -mpcx -d300 -h100 -f test.pcx test.hpg
```

Instead of converting the picture to a PCX bitmap we can also convert it to an Encapsulated PostScript file (use `-meps` and `-f test.eps` instead of `-mpcx` and `-f test.pcx`). After conversion we can proceed as discussed above.

As an example we show a Lotus picture that is printed as a HPGL file and then converted to a 300 dpi PCX bitmap and a EPS picture using the syntax described above. After conversion we used the style file `figures.sty` (see below) to print this picture (Figure 4).

6 POSTSCRIPT and GhostScript

If you want to view, print and manipulate POSTSCRIPT files and you do not have POSTSCRIPT printer (or commercial software), we suggest to use the freeware program *GhostScript* from Aladdin Enterprises. Using *GhostScript* you can view and print `.ps` and `.eps` POSTSCRIPT files on any screen and any printer. *GhostScript* also supports the Tseng ET4000 and the Trident graphics card for viewing in super VGA mode. You can resize the picture to any length and width. You can also calculate the BoundingBox and convert the POSTSCRIPT picture to a PCX bitmap. This PCX bitmap can be used with the DVI drivers of *emT_EX* to view and print the picture in *T_EX* documents without using a POSTSCRIPT DVI-driver. For a detailed discussion how *GhostScript* works we refer to the *GhostScript* documentation. Note that *GhostScript* is freeware, is regularly updated and gives good results. The only disadvantage is perhaps that it is not very user friendly.

7 The figures style files

The style file `figures` is a modification of the `psfig` style file and is used in *4T_EX*. It combines the possibility to print/view PCX pictures with the *emT_EX* special commands, and to print POSTSCRIPT files as with `psfig`. By default (or when using the command `\pcx`) `figures` will try to use PCX picture files. When not found or when using the command `\postscript`, `figures` will look for POSTSCRIPT files (`.eps` and `.ps`). This makes it possible to view/print pictures using the *emT_EX* special commands and to print the pictures in a *T_EX* document on any printer. To include a picture include the `figures` style option at the top of your document:

```
\documentstyle[11pt,figures]{article}
```

and, when you wish to include the figure (`example.pcx` or `example.eps`), call the macro like this:

```
\putfigure{%
  figure=example,width=2in,height=3in}
```

Note that the extension of the picture file is not specified. All commands defined in the style file `psfig` are also available (see e.g. Goossens (1993)). Some names of the macros of `psfig` are changed, e.g. `\psfig` is changed in `\putfigure` and some extra macros are added, e.g. `\pcx` (use PCX files), `\figurefull` (the same as `\psfull`); `\figuredraft` (the same as `\psdraft`).

8 4T_EX

In the preceding sections we have discussed several ways to incorporate graphics in *T_EX* documents. These sections are summarized by a flow diagram (see Figure 5).

The main problem of a user who wants to include a picture is that he/she needs to know which program to use and which parameters and commands one needs before one gets reasonable output. The *4T_EX* workbench is developed to shield you from these dirty bits. The aim of *4T_EX* is a simple menu based interface that lets the user choose between all available *T_EX* related programs and give some help wherever needed.

The *GRAPHICS* utility of *4T_EX* helps you incorporating pictures. All the programs discussed above are used, but a user does not need to remember all the (program specific) parameters. Simply choosing from an options list one can specify the parameters. Often this is not necessary because most parameters are set automatically (e.g. the printer resolution). Often the user only needs to specify the size of the picture and then *convert* the picture. Converting means

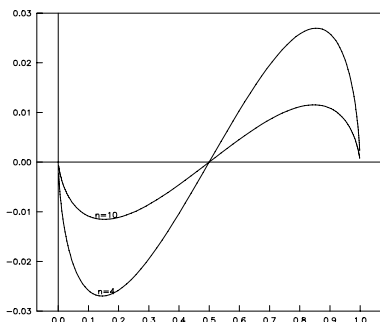


Figure 4. A HPGL picture converted to PCX or EPS

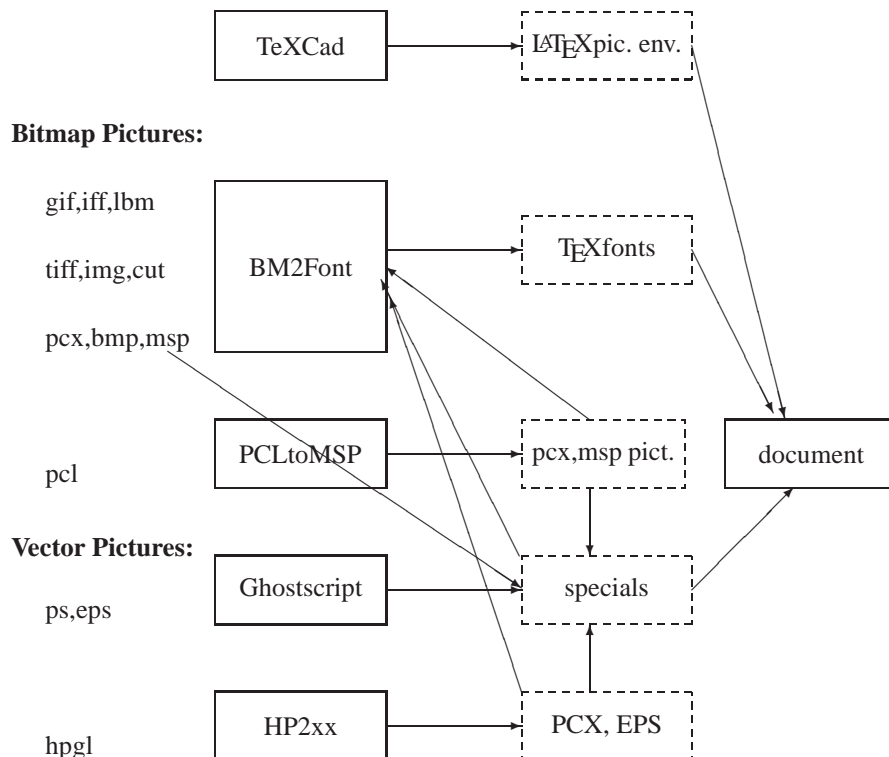


Figure 5. How to insert graphics (example of $\mathcal{A}\TeX$ cad)

that the correct programs are called and that the result is a \TeX file that can be used to insert the picture in your document. When for instance we convert the file `acad.hpgl`, $\mathcal{A}\TeX$ will end the conversion by telling the user that the picture may be inserted simply by adding the style file `figures.sty` and using the statement:

```

\begin{figure}
  \centerline{\input ACAD}
  \caption{your title}
\end{figure}
    
```

In this way a uniform approach to inserting pictures is reached, *i.e.* it does not matter if you use *HP2xx*, *BM2Font*, *PCLtoMSP*, or *GhostScript*. After conversion only one simple input statement is enough.

$\mathcal{A}\TeX$ also makes it possible to view and manipulate the picture. Viewing is also possible in super VGA modes. For instance a POSTSCRIPT picture can be rotated, a BoundingBox can be calculated and $\mathcal{A}\TeX$ magnifies the picture so that it will fit the specified size exactly. The same holds for HPGL pictures. Bitmap pictures can be manipulated using *Graphic WorkShop*. $\mathcal{A}\TeX$, however, uses the shareware program `cshow` to view bitmaps.

All the necessary bookkeeping is done by $\mathcal{A}\TeX$. For instance the fonts (`*.pk` and `*.tfm`) are stored in the correct directories and the conversion file is stored on the current working directory.

The conversions of pictures is done in such a way that it allows you to view and print the \TeX document with the pictures on any screen or printer. When you want to use a POSTSCRIPT printer you simple need to include the command `\postscript` in the document. The conclusion therefore must be that with $\mathcal{A}\TeX$ the inclusion of graphics has become an easy job.

References

- Darrell, T. (1992), *psfig 1.8 users guide*, available from public file-servers.
 Goossens, M. (1993), "PostScript en \LaTeX , de komplementariteit in de praktijk", *MAPS Nederlandstalige \TeX Gebruikersgroep*, **93.1**, 101–113.

Horn, G. (1990), *T_EXcad Version 2.8*, available from public file-servers.

Mattes, E. (1992), *DVIDRV 1.4s manual*, available from public file-servers.

IV L^AT_EX 2_ε standard graphics and colour support

David Carlisle
carlisle@cs.man.ac.uk
Sebastian Rahtz
spqr@ftp.tex.ac.uk

1 Introduction

With the release of L^AT_EX 2_ε, graphics file inclusion, rotation, scaling and colour macros are now a standard part of the system (though implemented as two separate packages). These are, of course, dependent on the abilities of the driver in use, as they are all implemented by using `\special` commands; it is hoped that until (if ever?) a common `\special` format is agreed upon, the standardized interface will make life easier for authors. This article *briefly* describes the facilities of the new packages, but for the full glory readers are advised to book for the July training meeting which covers this area.

Both these packages are now available on CTAN¹ in `macros/latex2e/packages/graphics`, but users are warned that documentation is incomplete, and that the *internal* interface is still open to change. Making the packages available is simple; if we start our document:

```
\documentclass[dvips]{article}
\usepackage{graphics}
\usepackage{color}
```

we load both packages, and the global option ‘dvips’ is passed to them, indicating which driver we are using (most common drivers are supported).

2 The graphics package



2.1 Rotation

Any T_EX box can be rotated with the command

```
\rotatebox{angle}{text}
```

where *angle* degrees is measured anti-clockwise. Normally the rotation is about the left-hand end of the baseline of *text*, but more complex examples are possible. A simple example of rotation is


```
I like \rotatebox{45}{cats} but
I cannot bear \rotatebox{-45}{dogs}
```

which produces: I like  but I cannot bear ; note that the right amount of space is left for the rotated material.

2.2 Scaling

A box can be resized in two ways, by scale or by specifying the desired size. The first is achieved with

```
\scalebox{h-scale}[v-scale]{text}
```

If *v-scale* is omitted, the vertical scale factor is the same as the horizontal one. Thus `\scalebox{2}[.5]{Cats}` produces .

Scaling to size is done with

```
\resizebox*{h-length}{v-length}{text}
```

¹Subscribers to the Mac or DOS disk packages from UKTUG will receive copies during June with the first full release of L^AT_EX 2_ε.

which resizes *text* so that the width is *h-length*. If ! (exclamation mark) is used as either length argument, the other argument is used to determine a scale factor that is used in both directions. Normally *v-length* refers to the height of the box, but in the starred form, it refers to the ‘height + depth’. As normal in L^AT_EX 2_ε, box length arguments, `\height`, `\width`, `\totalheight`, `\depth` may be used to refer to the original size of the box. The cats can be resized again using `\resizebox{1in}{0.2in}{Cats}` to produce **Cats**.

The user should be aware that these scaling operations are done by the *driver*, and it is likely that bitmap fonts will be scaled, instead of a new size being selected from scratch.

2.3 Graphics inclusion

The basic command to include a graphics file is:

```
\includegraphics*[llx,lly][urx,ury]{file}
```

If * is present, then the graphic is ‘clipped’ to the size specified. If * is omitted, then any part of the graphic that is outside the specified ‘bounding box’ will over-print the surrounding text.

If the optional arguments are omitted, then the size of the graphic will be determined by reading the graphic file itself, if possible. If `[urx,ury]` is present, then it should specify the coordinates of the top right corner of the image, as a pair of T_EX dimensions. If the units are omitted they default to bp. So `[1in,1in]` and `[72,72]` are equivalent. If only one optional argument appears, the lower left corner of the image is assumed to be at `[0,0]`. Otherwise `[llx,lly]` may be used to specify the coordinates of this point.

The package works by examining the suffix of the file name and looking that up in a rule-table, which tells it whether or not the file can be read for a size, how to do so, and what driver-specific macro to call for the type of file. This allows extensible support for whatever bitmap or vector graphic file types the driver can work with. The interested reader should consult the documentation for the interface to this system.

A combination of features in the graphics package allows us to rotate a scaled portion of a figure:

```
\rotatebox{45}{%
  \resizebox{1.5cm}{4cm}{%
    \includegraphics*%
      [100,100][500,600]{%
        golfer.ps}%
    }%
}
```



which produces

Those users familiar with the (*e*)*psfig* package can use an extended form of the graphics package (provisionally entitled `graphicx`) which offers a full ‘key=value’ interface to all the commands described above, using the generic `keyval` parser by David Carlisle described elsewhere in this issue of *Baskerville*. A small wrapper package (`epsfig`) provides an exact emulation of `psfig` (and Rokicki’s `epsf` macros) for those with existing documents marked up in this way.

3 The colour package

The L^AT_EX 2_ε colour support offers a variety of facilities:

- colouring text;
- colouring box backgrounds;
- setting the page colour;
- defining new colour names

3.1 Using colours

There are two text colouring commands, which work in the same way as the normal font-changing macros. The first one is a *command*:

```
\textcolor{<colourname>}{<text>}
```

This takes an argument enclosed in brackets and writes it in the selected colour. This can be used for local colour changes, since it restores the original colour state when it is completed, *e.g.*

This will be in black

```
\textcolor{Blue}{This text will be in blue}
```

and this reverts to black

The second colour macro is a *declaration*:

```
\color{<colourname>}
```

This colour macro takes only one argument and simply sets a new colour at this point, *e.g.*

```
\color{red} All the following text  
will be red.
```

```
\color{black} Set the text colour  
to black again.
```

The colour declaration does of course respect normal \TeX grouping; if we write

We start in black, but now

```
{\color{red} all text  
is in red, {\color{green} but this  
should be in green} and this  
should be back in red.}
```

And we finish in black

we will see²

We start in black, but now **all text is in red, but this should be in green and this should be back in red**. And we finish in black

The *background* of a normal LR \TeX box can also be coloured:

```
\colorbox{<colourname>}{<text>}
```

This takes the same argument forms as `\textcolor`, but the colour specifies the background colour of the box.

There is an extended form:

```
\fcolorbox{<colourname>}{<colourname>}{<text>}
```

This has an extra *colourname* argument, and puts a frame of the first colour around a box with a background specified by the second colour.

The line width and the offset of the frame from the text are controlled by the standard `\fboxsep` and `\fboxrule` lengths.

\TeX does not have internal support for colour attributes of text, and \TeX ‘grouping’ across pages, floats, footnotes etc will not always yield the expected results. However, $\LaTeX 2_{\epsilon}$ has extended support to cope with most situations, and it is hoped that more driver support will make this even better.

3.2 Defining new colours

The colour names ‘white’, ‘black’, ‘red’, ‘green’, ‘blue’, ‘cyan’, ‘magenta’ and ‘yellow’ are predefined by all driver files. New colour names can be defined with:

```
\definecolor{<name>}{<model>}{<spec>}
```

where *spec* is usually a list of comma-separated numbers needed by the *model*. Typically, drivers can cope with the models *gray*, *rgb* and *cmyk* (although the system is extensible), allowing, *e.g.* :

```
\definecolor{lightgrey}{gray}{.25}  
\definecolor{cornflowerblue}{rgb}{.39,.58,.93}  
\definecolor{GreenYellow}{cmyk}{0.15,0,0.69,0}
```

²The examples of colour like this will be set using gray scales.

It is also possible to use the `\textcolor` and `\color` macros with an explicit colour model and specifications, to avoid the overhead of defining new colors.

One of the important concepts inherited from James Hafner's `colordvi` macros is the allowance for a layer of colour 'names' above the actual specification given to the printer; Hafner worked out a set of 68 CMYK colours which correspond to a common set of Crayola crayons; these are predefined in the header files used by *dvips*, and the user calls them *by name*, allowing for tuning of the header files for a particular printer without changing the source. This system is provided by the $\text{\LaTeX}2_{\epsilon}$ colour package for those drivers which support it, and its use is strongly recommended.

V The keyval package

David Carlisle
carlisle@cs.man.ac.uk

1 Introduction

This article describes a $\LaTeX 2\epsilon$ package implementing a system of defining and using sets of parameters, which are activated using the syntax $\langle key \rangle = \langle value \rangle$. It is distributed with the $\LaTeX 2\epsilon$ standard graphics / color packages which are available on the CTAN archives in `macros/latex2e/packages/graphics`. It is important to stress that although the $\LaTeX 3$ project intends to support interfaces of this kind, the present package is *not* in any sense a preview of $\LaTeX 3$.

When a ‘keyval’ system is set up in a \LaTeX package, a set of macros is defined for each keyword that is needed; this is called whenever the parameter appears in a parameter list. For instance, if the set `dpc` is to have the keyword `scale`, then the author might write:

```
\define@key{dpc}{scale}{scale ({\tt\string#1})}
```

The first argument of `\define@key` is the name of the set of keywords being used, the second is the keyword, and the third is the macros to call. These will be given as `#1` the $\langle value \rangle$ specified by the user.

Normally it is an error to omit the ‘ $\langle value \rangle$ ’; however if an optional $\langle value \rangle$ is supplied when the keyword is defined, then just the keyword need be supplied. Thus, after

```
\define@key{dpc}{clip}[true]{...}
```

the user can type ‘`clip = true`’ or ‘`clip = false`’ or just ‘`clip`’, which is the same as ‘`clip = true`’

To use these keywords, we call `\setkeys` with a comma separated list of settings, each of the form $\langle key \rangle = \langle value \rangle$, or just $\langle key \rangle$. Any white space around the ‘=’ and ‘,’ is ignored.

As the $\langle key \rangle$ is passed as a macro argument, if it consists entirely of a `{ }` group, the outer braces are stripped off. Thus `,key=foo`, and `,key={foo}`, are equivalent. This fact enables one to ‘hide’ any commas or equals signs that must appear in the value. *i.e.* in `foo={1,2,3}`, `bar=4`, `foo` gets the value `1,2,3` — the comma after `1` does not terminate the keyval pair, as it is ‘hidden’ by the braces.

Empty entries, with nothing between the commas, are silently ignored. This means that it is not an error to have a comma after the last term, or before the first.

2 Example

We may extend the examples above to give a ‘fake’ graphics inclusion macro, with a syntax similar to that used in the `psfig` macros.

`\dpcgraphics` has one optional argument which is passed through `\setkeys`, and one mandatory argument, the filename. It actually just typesets its arguments, for demonstration.

The declared keys are: `scale`, `height`, `width`, `bb`, and `clip`. Except for the last, they must all be given a value if used.

Note how in the following, any white space around = or , is ignored, as are the ‘empty’ arguments caused by extra commas. Note also that each macro receives *exactly* the tokens that you specify as arguments, no premature expansion is done.

```
\newcommand{\dpcgraphics}[2][ ]{%  
  {\setkeys{dpc}{#1}INPUT: #2}}
```

```
\define@key{dpc}{height}{%  
  height ({\tt#1})\}\}  
\define@key{dpc}{width}{%  
  width ({\tt#1})\}\}
```

reprinted from *Baskerville*

Volume 4, Number 3

```

\define@key{dpc}{bb}{%
  bounding box ({\tt#1})\}
\define@key{dpc}{scale}{%
  scale ({\tt\string#1\relax})\}
\define@key{dpc}{clip}[true]{%
  clip ({\tt\string#1\relax})\}
\def\scalemacro{9}
\dpcgraphics
[ height =4in, ,
  width = 3in,
  scale = \scalemacro,
  bb = 20 20 300 400 ,
  clip,
] {aaa}
height (4in)
width (3in)
scale (\scalemacro)
bounding box (20 20 300 400)
clip (true)
INPUT: aaa

```

3 The Internal Interface

A declaration of the form:

```
\define@key{family}{key}{...}
```

defines a macro `\KV@prefix@key` with one argument. When used in a keyval list, the macro receives the value as its argument.

A declaration of the form:

```
\define@key{family}{key}[default]{...}
```

defines a macro `\KV@family@key` as above; however, it also defines the macro `\KV@family@key@default` as a macro with no arguments, and definition

```
\KV@family@key{default}.
```

Thus if macros are defined using `\define@key`, the use of a key with no value `... ,foo,...` is always equivalent to the use of the key with some value, `... ,foo=default,...`. However, a package writer may wish that the ‘default’ behaviour for some key is not directly equivalent to using that key with a value (in particular, as pointed out to me by Timothy Van Zandt, you may wish to omit error checking on the default value as you know it is correct.) In these cases one simply needs to define the two macros `\KV@family@key` and `\KV@family@key@default` directly using `\def` (or `\newcommand`). I do not supply a user interface for this type of definition, but it is supported in the sense that I will try to ensure that any future upgrades of this package do not break styles making use of these ‘low level’ definitions.

VI Baskerville Production — the horror story

Robin Fairbairns
Computer Laboratory
University of Cambridge
robin.fairbairns@cl.cam.ac.uk

or — never mind the quality, feel the hsize

I've been part of the *Baskerville* production team for three issues now; you may have noticed my name in Sebastian's 'Colophon'. What I want to discuss in this article is the production business as I (and Jonathan Fine) see it. Sebastian's story is entirely another thing: he's the one that does the *real* work — we simply proof-read and get what he turns out onto paper.

Two things prompted me to produce this article: the first was an actual technical production achievement, which I thought might be of use as a 'technical tip' (there's a standing action on all of the UK T_EX Users' Group Committee to produce such article for *Baskerville*). The second prompt came in a mail message from Allan Reese, who said "[...], it seems to me that there has been a substantial reduction in the physical quality [of *Baskerville* production]"; I hope to shed some light on the effect that Allan observed.

From proof to paper

Sebastian works on *Baskerville* wherever he happens to be at the time; his domestic arrangements are such that this often means 'at home'. When an issue is nearing completion, he (e)mails the committee to say that a proof POSTSCRIPT file is ready for inspection. Those of us who have an account on the CTAN archive `ftp.tex.ac.uk` can then log in and retrieve the file that he's placed in his private filespace there (one of Sebastian's other hats is that of CTAN maintainer).

The POSTSCRIPT file is typically in the range 1.5–2MB long; pulling several copies (as I have to do) is only really practical politics because I have access to plenty of free disc space. More significantly, the file takes anything up to an hour to print (all the printers I can use are connected by serial line); so I don't *print* more often than is absolutely necessary. (Most of the bulk of the file is made up of downloaded copies of the fonts: early on in the present regime, Sebastian produced an issue as seven four-page files, the largest of which was 1.7MB, and took nearly an hour to print on its own!)

Once I've printed a copy, there's a rush on to proof-read. This is something I tend to be good at (I've a lot of experience of proof-reading ISO standards, a peculiarly nit-picking way of spending one's time). I (and others) mail lists of perceived problems back to Sebastian, and we iterate until all seems right. Because everything happens asynchronously, this iteration can be a long process.

Eventually, a week or two later, I have a revised file to print. I duly go ahead, using the 'service' HP LaserJet 4 here. While it's a damned good printer, I have had unsatisfactory prints out of it — among other things, the toner seems always to run out just before I need to do a production run!

At this stage, Jonathan Fine takes the results of my efforts for copying. My laboratory has a very respectable print room, with a Xerox Docutech copier, but we don't use it because the cost is nearly twice what we pay the copy shop (part of the reason is that the price break at the copy shop is at a smaller number of copies than that in my laboratory's print room). To prepare the matter for copying, Jonathan has to paste the A4 pages from my printer onto A3. This game ('imposition') almost deserves an article in its own right. It prepares *Baskerville* for presentation as you see it, stapled down the middle of (typically) six sheets of A3 paper, and is something the Laboratory's Docutech would do electronically, but the copy shop can't do.

In fact (in my opinion), the weakest part of the production chain is the copy shop; they are providing as cheap a service as they can manage, the result of which is that we occasionally suffer mis-stapled copies, copies with pages missing, and so on. (They are, at least, happy to put such problems right.)

All that remains to do is to stuff and then post the envelopes. This isn't a trivial matter, but its 'technicalities' are hardly significant to a group of people whose common interest is typesetting: I'll say no more.

reprinted from Baskerville

Volume 4, Number 3

What could we do better...?

Baskerville's early (somewhat irregular) issues were produced at Aston University on a phototypesetter, and then reproduced onto high-quality paper. This gave the impression of high quality³, but it bore a cost that we could probably not meet on the punishing publication rate of *Baskerville* at present.

We could print onto better-quality paper: "reprographic quality" paper for use in laser printers is available: its use would presumably improve the reproduction of what we've printed.

I could find a more lightly loaded (and hence more consistent quality) LaserJet 4 (or other 600dpi or better printer); this may or may not be possible — if anyone in the Cambridge area has such a machine to offer at the right price⁴, we'll gratefully accept.

We could improve the imposition process and finally, we could go to a better copy shop.

... *and why don't we?*

I'm always willing to consider a better printer; but since I know of none, I can't use it.

A copy shop with tighter quality controls would cost us more, and would probably (as a direct result of the controls) take longer.

The view of the committee is that *Baskerville*'s quality is (just about) adequate as it is. We are producing the newsletter of a (not very large) user group: the imperative is that we get the thing out as often as possible — our members more often comment about the content of *Baskerville* than about its presentation.

What will we do?

We are going to try reprographic quality paper: since the amount of paper used for *printing* is small, the incremental cost per issue won't be serious. On a similar basis, Jonathan has bought some 'professional' spray gum for use in the imposition process (it'll be much quicker than using PrittSticks, and the results ought to be better).

Unless (or in any case until) we find somewhere better, we will try to improve our relationship with the copy shop. When they make mistakes, they lose money by having to redo work for us: we expect they'll be happy to work with us to improve their service to us (thus, incidentally, improving their profits).

The technical nugget

Stuck on the wall above our LaserJet 4 are a couple of examples of misprinting (one is a print of the University crest with a spike sticking out several inches to the North-East). Notes on the examples suggest that the solution is "to use another printer" (it's evidence of a firmware bug in the POSTSCRIPT interpreter). In the last edition of *Baskerville* (volume 4, issue 2), some text at the bottom of page 20 'jumped' from near the beginning of column one to just before the beginning of column two; the effect was to obscure the line in both columns by overprinting.

Changing printers wasn't actually an option for me, since I was committed to using a LaserJet 4, and I know of none which don't have the bug. Some time ago there was a discussion on Usenet group `comp.text.tex` about just this problem, and I retained a suggestion from Hans Visser (of the Technical University of Delft) on how to solve the problem within `dvips`. But I don't run `dvips` for *Baskerville* — Sebastian does, and there's a delay introduced by any interaction with Sebastian; so I decided to try and solve the problem myself.

Visser's solution was to edit the definition of the abbreviated command 'w' in `texps.pro`, one of the files `dvips` inserts into the output of any file it produces. The change he suggested was to change the original `/w{0 rmoveto}B` to `/w{10 5 rmoveto 10 sub -5 rmoveto}B` (the command B is defined earlier in the file to mean `bind def`).

My problem was caused by POSTSCRIPT that said `(er)s(ville)` (part of *Baskerville* as a font name). Remembering Visser's instructions, I traced the definition of `s` to mean `/s{show 3 w}B`; wanting a 'quick fix', I decided simply to change the offending instance of the `w` command. My third shot, `(er)show 1.5 100 rmoveto 1.499 -100 rmoveto(ville)` finally did the job.

I wouldn't recommend the game I played (locating stuff in POSTSCRIPT generated by `dvips`, and then correcting it) to anyone; but it *did* work. I would be interested to learn if anyone else has encountered this same firmware bug in LaserJet 4s (I'm told it's been eliminated from current production machines), and if there are alternatives to Visser's solution. In the meantime, I'm asking Sebastian to change *his* `dvips.pro`..

³Though there are those who complained that those versions of *Baskerville* were 'under-inked'

⁴Very cheap!

VII Book reviews

Malcolm Clark

m.clark@warwick.ac.uk

0.1 *Writing & Illuminating & Lettering*, by Edward Johnston

This is an interesting book, not least since it is said to have been continuously in print since 1906. My edition is from 1946 (published by Pitman), but I note that it is now available from A&C Black. In essence this is an account of Calligraphy. What has this to do with printing, you may ask? A number of typographers would argue that calligraphic skills are a necessary prerequisite for the adequate understanding of type. But more than this, I have a soft spot for Johnston. He taught at the Central School of Arts and Crafts, alongside Eric Gill (who had been his pupil, and with whom he later shared rooms). Johnston went on to create the typeface which was used by London Underground⁵, and in a digital form still is used by them. Gill of course created *Gill Sans* (among others), which bears some similarities (well, it's a sans serif) to *Johnston*. The similarity was great enough to lead one correspondent on `comp.fonts` to consider *Johnston* a derivative of *Gill Sans*. Neither are derivatives of the other, but stemmed from some common beliefs and skills. The really interesting thing is that Johnston was a calligrapher, while Gill was a stone cutter. The requirements of the two crafts are rather different – and different again from type. *Johnston* was intended as a ‘signing’ face: it was not intended for continuous text (unlike *Gill Sans*). I have however seen the modern variant (*New Johnston*) which was redesigned by Banks&Miles used for continuous text. It works up to a point. Of course other signing fonts have become used for continuous text – most notably Helvetica (designed for signs at Swiss airports).

Obviously the book is primarily about hand lettering: Johnston was writing to encapsulate the craft, and to encourage others to develop their skills. It is therefore a rather detailed manual which leads you through all the practical aspects of the craft. There is an underlying theme, which may be made explicit from this quote: ‘The first general virtue of lettering is *readableness*, the second, *fitness* for a given Use.’ Almost everything else stems from this belief. Given our concern with typography it is arresting to read that ‘to arrange letters well requires no great art, but it requires a working knowledge of letter-forms and of the reasonable methods of grouping those forms’. So there.

Given the title, which included ‘illuminating’ it is hardly surprising that Johnston includes the use of colour. Traditionally, rubrication (*i.e.* ‘adding of ‘Red, or other coloured letters... to a MS. or Book’) was used for title pages, for prefaces and notes, for headings of columns and pages, for initials, marking stanzas, colophons, and so on. Johnston seems to have had an especial liking for gold, either matt or burnished. This is one feature which is unlikely to transfer to electronic publications. Elsewhere he notes ‘use a limited number of pure, bright colours’.

It may also be interesting that Johnston is quite happy to ‘letter-space’ in some circumstances, although the example he gives is upper case and he makes no obvious linking to letter spacing lower case.

It is an interesting read: it is very practical and straightforward. He advises, recommends, but seldom pontificates. He makes it all so reasonable that I might just sign up for a calligraphy class next term.

Copies may be obtained from Typobooks, FREEPOST, Colchester, Essex CO3 4JH, for a mere £14.99. This new edition has the advantage that the plates have been re-originated. Those in my copy lack definition.

0.2 *Modern Typography*, by Robin Kinross

This is a rather fine volume. It discusses the history of typography since ‘Modern’ times. That is, since about 1700. This is essentially the same meaning for ‘modern’ as is found in ‘Computer Modern’. Kinross chooses this date since it is around then that there emerges a ‘readiness to articulate knowledge and consciousness’ (of printing). The book will not tell you how to create typography. What it discusses are the influences which created the movements in typography. It does so in a considered and careful way which was dismissed as ‘dry’ by one `comp.fonts` correspondent; on the other hand, I found it erudite and entertaining.

Kinross describes it as ‘typographic history’, a field he criticizes in general for its absence of historical skills, superficial notions of design, and ritual statements of admiration or distaste. He also notes that much previous work along these lines (citing Stanley Morison) has made the assumption that all typography is book typography. Adopting

⁵at this point I have to say ‘Frank Pick’. Frank Pick was the driving force behind the corporate identity of London Underground, all those years ago – a corporate identity which has, up to now, managed to survive.

this iconoclastic approach, he gives short shrift to, for example, Baskerville, Bodoni, the Bauhaus, the more precious private presses and ‘fine printing’.

As he wends his way from Moxton to the present day, he throws some light on one of the ironies of rationality: the standardisation of the point system predated metrication by only a few years, and a proposal to create a metric point (from Firmin Didot) failed to be accepted. It is also to France that the origin of paper sizes in the ratio $1:\sqrt{2}$ belongs, although this did not achieve prominence until it was adopted in Germany early this century. Kinross has little time for the argument that greater mechanization in printing led to a decline in standards. On the contrary, he argues that ‘standards of press-work only improved with powered printing’. We are taken through a well-documented account of printing in England from the times of William Morris, and similarly shown the influences which crossed the Atlantic, as well as the more or less autonomous developments taking place in the United States. In the early years of this century, German typography was a force to be reckoned with. Within it were debates which hardly affected the english-speaking world, notably that of the place of ‘blackletter’. Kinross summarises this crudely (his own term) as ‘to prefer roman over blackletter was to be modern... to prefer sanserif roman letters to those with serifs was to be more modern still’. His background references and discussion shows how much more complex the question was, placing the debate firmly in its political and social context.

It is perhaps with the ‘new typography’ that Kinross is at his most excited. This ‘movement’ is most clearly given form by Jan Tschichold’s *Die neue Typographie* published in 1928. Kinross links back to a statement by Lissitzky in 1923 which might be felt to be prophetic: ‘The printed surface transcends space and time. The printed surface, the infinity of books, must be transcended. The electro-library.’ Make of that what we will. Perhaps the strongest principle of this new typography was that it was related to purpose; this therefore helped determine the details of the designed object, as well as placing it in its wider context. It is intriguing that one consequence of this new typography was ‘the cult of spiral binding’, as well as a tendency towards unjustified setting – ‘more proper to machine composition’. In the midst of this Kinross deals with the Bauhaus, pointing out that there were never any fully-fledged typographers there. Later he goes on to suggest that the reputation of the Bauhaus rested for many years on the exhibition about it held in New York in 1938, accompanied by the book accompanying the show. In a sense the Bauhaus reinterpreted itself, at a time when it could be seen as another symbol of hope crushed under the heel of the oppressive dictator.

After the war came the rise or identification of ‘Swiss typography’. One element of this was the application of ‘the grid’ (possibly partly a result of the need to accommodate the three official Swiss languages – they could be fairly conveniently given three columns on the grid: another consequence was the tendency to produce square formats). One interesting feature of Kinross’s book is the extent to which he reveals the internal tensions within the typographic world: tensions which are seldom evident from the outside.

Developments do not end here, and in his final chapter Kinross does touch on digital processes, even going so far as to spend some time on METAFONT, and more significantly, on PostScript. Post-modernism does appear, but curiously, its practitioners are not named: I had rather expected to see something about Spiekermann and Brody, or at least Emigre, but although clearly quite aware of them, he remains relatively silent. However, an example of Spiekermann’s work does appear in the illustrations. The thirty or so illustrations enlarge and expand the text, and often represent items otherwise hard to find (for example a German book of 1935 where blackletter text is printed in a surprisingly ‘modern’ form).

The form of the book is itself interesting. Each page has a wide left margin and is set ragged right. I was surprised how comfortable I found this. The wide margin is used for ‘footnotes’. There is very little use of type differentiation to denote special features. In fact, each section starts with a heading in the body type, but offset by 20 pt or so (and a bigskip). Running heads are in small capitals. I think I object to the running heads being set to the right on each page, since they include the page number. I really don’t find page numbers next to the binding edge very useful. Footnotes are in a slightly smaller face, while chapter titles are slightly larger than the body type. Apart from that, the only variation is in the occasional use of italic. This is nicely minimalist. It echoes Richard Southall’s short article many years ago in *TUGboat* 5(2), ‘*First principles of typographic design for document production*’. It will come as no great surprise that Kinross also came through Reading University’s Department of Typography.

This is a book I will recommend. I felt much better informed from having read it, with a better understanding of the movements in typography, and a better feeling for their limitations. Although it does concentrate mainly on ‘bookish’ matters to the exclusion of the world of magazines, newspapers and journals, it does provide useful insights and provoking conclusions. It is published by Hyphen Press at about £15.

VIII A tutorial on TrueType

Ian D Chivers
udaa260@bay.cc.kcl.ac.uk

1 Background

For a better understanding of what True Type has to offer it is useful to look a little closer at fonts and typography.

1.1 Type

It is important to realise that the design of fonts is part science, part art and part magic. When we look at the highest quality fonts we find they are designed to look good at a particular point size. A particularly good recent example is *Telephone Book*, the font used in your BT telephone directories, and much credit must be given to John Miles of Banks and Miles for what has been achieved. The font is small, yet very legible when printed on relatively poor quality paper, and the directory is now 4 column rather than 3 column. This represents a considerable saving on trees!

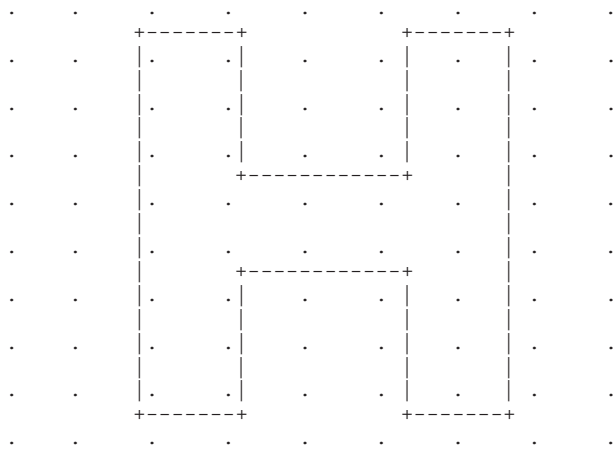
1.2 Scalable fonts and digital type

Let us now look at scalable fonts and digital type. What the computer offers with the possibility of scalable fonts seems to go against accepted practice. The process is

1. take the font outline and scale to the correct size;
2. fill the outline with pixels, *i.e.* generate a raster bitmap;
3. transfer to the display device

There are two main problems here. Firstly a scaled font does not look good at all points sizes — remember that fonts are designed to look good at specific points sizes. Secondly rasterisation has inherent problems that are particularly acute with low resolution devices where for example we have to map a character onto a small grid of dots or points. For more information on the general problems of rasterisation one should look at a book on computer graphics, and there is one given in the bibliography.

The following illustrates one of the problems,



where we are mapping an upper case h onto a low resolution device. The right hand upright is only one pixel wide, compared to two for the left. The human eye is surprisingly good at noticing things like this. To see this more realistically Paintbrush offers the possibility of zooming in on some text, and one can see the actual pixels and how the characters are made up. It is quite illuminating.

Adobe were the first company to achieve widespread success with scalable fonts with POSTSCRIPT. They managed to do this by firstly adding special rasterisation algorithms to the interpreter to overcome the problems of small point sizes and low resolution devices, and secondly adding hints to type 1 fonts to further massage the characters to enhance their appearance.

2 True Type Fonts

One of the heavily marketed features of Windows 3.1 was the addition of True Type fonts to Windows. As someone who has been involved in electronic publishing for over ten years I had a high degree of scepticism about Windows and True Type.

There are a number of advantages that True Type should bring and some of these are:

1. For all Windows applications the screen display should match the printer display, if the printer supports bit mapped mode, POSTSCRIPT or TrueType.
2. The disappearance of jagged screen fonts, with no need to build each font at each point size that you wanted, as the fonts are scalable. This was the case with earlier versions of Windows, and one that will be familiar to most people involved in the dtp field, not just under Windows.
3. In a university environment with staff and students using public access systems with Windows installed it means that they can take their document from one system to another and always have access to the following fonts

Times New Roman
 Arial
 Courier New
 Symbol
 Wingdings

and print them on a range of printers and get the same results, as the rasterisation takes place in the pc, rather than the printer.

If Word for Windows is installed then the following additional fonts are also available

American-Uncial-Normal
 Fences
 MT Extra
 News Gothic

4. The ability to send documents to bureaus without breaking the copyright on the font. With True Type you can embed a font in the document. The fonts are encrypted, but are easily decrypted for screen display and printing.

Well, does it all work? In the main, yes. Though as with most things that Microsoft are involved with there are some minor problems.

The statements below are based on a system based on a 33 MHz 80486 DX PC, with 8 Mb memory and a 1024 × 768 colour display.

Firstly let us look at the rescaling of text as we alter the window size or actually alter the point size of the text. The on screen display is pretty good in most applications though I was surprised at the variation between them. Discussions with someone involved in Windows developments indicates that there is more than one way to achieve what should be the same end result.

I had Adobe Type Manager installed and found little noticeable difference between them. ATM handles the POSTSCRIPT fonts, and Windows handles the TT fonts.

There were differences in what packages running under Windows offered:

Word For Windows 2.1	4 to 127 points, 0.5 point increments
Excel 4.0	1 to 409, 1 point intervals
Write	4 to 127, 1 point intervals
Designer 3.1	1 to 144, 1 point increments
Ventura 4.1.1	1 to 254 in 0.5 points
Corel Draw	0.7 to 2160.0, 0.1 point intervals

The sizes of the output files when printing surprised me initially. The following figures are for one A4 page that contains a table with each character in the Windows character set.

Times Roman, POSTSCRIPT printer, using internal printer font	173,958b
TT Times New Roman, POSTSCRIPT printer, download bitmap, 1270 dpi	5,648,944b
TT Times New Roman, POSTSCRIPT printer, download bitmap, 300 dpi	509,841b
TT Times New Roman, Epson printer, download bitmap, 180×360 dpi	599,905b

If we look at one of the graphing packages or spreadsheets then we would have figures similar to those below:

POSTSCRIPT printer, A4 page, 11 × 8 inch	990000b
image area, 300 × 300 dpi	
Epson 360 × 180 dpi, A4 page, 11 × 8 inch	712800b
image area, 360 × 180 dpi	

For a 20 page document involving text and graphics therefore we could easily have a file size of $20 \times 600000b$, or about 11.5 Mb.

Printing this kind of document is therefore slow on dot matrix printers, and not wonderfully fast on POSTSCRIPT printers.

This obviously has some fairly important ramifications in a university environment where increasingly people are moving to Windows and Windows based applications. Good business for the printer manufacturers!

TT works well with non-latin fonts, which did again surprise me. With Arabic, for example, there is a separate screen font for menus, dialog boxes etc, as the subtle angles in Arabic writing cannot be scaled cleanly at normal screen resolutions (72 dpi — ega or 96 dpi — vga). However, when printed it looks simplistic. The second Arabic font does look quite good printed and does a reasonable job of the cursive Arabic style.

3 So who will use TT?

For people with dot matrix printers, and early HP Laser Jets (and there are a lot about) they have for the first time access to a print quality normally associated with POSTSCRIPT devices.

For people involved in document interchange the fact that you can legally move documents and the fonts around is a strong plus. How many people reading this article will have been given a POSTSCRIPT file to print and had problems with missing fonts?

For people with access to local POSTSCRIPT facilities and typesetters at ULCC and Oxford then I can't easily see TT displacing POSTSCRIPT.

An examination of the major players in the typesetting industry shows that POSTSCRIPT is the de facto industry standard. Xerox offer for example a complete publishing solution with their DocuTech aimed at the professional printer, and that is based on 600 dpi scanners and POSTSCRIPT.

4 Further reading

For those with Internet access there are a number of sources of information on fonts and TT in particular. Microsoft make available a mass of material at `ftp.uu.net` in the `/vendor/microsoft/TrueType-Info` directory. Some of the files held with descriptions are:

- `ttspec1.zip`, `ttspec2.zip`, `ttspec3.zip`: The complete TT specification. About 400 pages if my memory serves correctly.
- `tt-talk.zip`: Contains three papers, one on digital type, one on linear versus non linear scaling and the third on the Lucida family of fonts.

There is also a very good FAQ on fonts that looks more generally on fonts with a good coverage of a variety of hardware and software platforms. Good start for a general overview of what is available.

IX Backslash—Expansion of macros and so forth

Jonathan Fine

J.Fine@uk.ac.cam.pmms

It is usual, in programming languages which admit compilation (such as *C*, *BASIC* and *Pascal*) for there to be a rigid and inviolable separation between code and data. It is possible for an interpreted *BASIC* program to write a program source file which is then loaded and run, but such is rather bad form. The same separation generally applies to *Smalltalk*, which is probably the most sophisticated of the interpreted languages. (My knowledge of *LISP* is limited. May its supporters please note that my endorsement of *Smalltalk* is, for the purposes of this column, a personal opinion only).

T_EX, however, has no inbuilt distinction between code and data. As far as it is concerned, all is just one long sequence of varying types of tokens. This will be made clearer later. It is not as if there is one stream from which instructions are drawn, and another from which data is drawn. It is usual for compiled programming languages to have a “*GOTO*” mechanism (usually implicit within loop and conditional constructs, and also subroutine and function calls) that allows forward and backward jumps within the code stream, which is in fact more like a heap of tiny sequences of instructions linked by random access pointers.

Why am I saying all this? Most beginners expect *T_EX* to behave like other programming languages. Up to a point it does, particularly if all one wishes to do is write a simple replacement text macro, or set the values of some registers or parameters. But when it come to reading data from within a macro it definitely does not, and here beginners generally become unstuck, in the sense of losing their grip and running off the rails. In another sense, of course, they become stuck. You pays your money, you takes your choice.

I know that I had these problems six years ago when I started with *T_EX*. While the *T_EXbook* explained to me how *T_EX* behaved, it did not give examples to clearly dispel my wrong prejudices. (If you have prejudices or habits, may they be beneficial.) Hence this article. Most people have some experience of writing a program, even if only a humble batch file for use with *MS-DOS*.

It is a simplification, which does no harm for the purpose of this article, to imagine the input stream to *T_EX* being one enormous long list of tokens. Change of category codes, `\input` and `\endinput` commands, and also the `\openin` and `\read` commands do not fundamentally alter this point of view.

If a format file or some macros have previously been loaded (and such usually has been) then some of these tokens will be macros (or more exactly will have macro meaning when executed) and will thus influence the subsequent operation of *T_EX*.

It is now time to announce the fundamental law on the expansion of *T_EX* macros. Suppose a *T_EX* macro in the input stream (usually but not necessarily at the very head of the stream) is expanded. The effect of this expansion is to alter or edit the input stream, in a very specific manner. This is explained on [203] (this means page 203 of *The T_EXbook*). Once the parameter text, if any, has been read, and the replacement text, if any, has been put in its place, the expansion of the macro is at an end. It is done, over, finished, and no more. However, for the purposes of error reporting *T_EX* keeps a note of how the replacement text came to arise. We will see the use of this later. This information however in no way affects subsequent error-free execution. As far as *T_EX* is concerned, it is just as if it had been presented at this stage with the given amended input stream. Processing by *T_EX* now continue with the current state and the new stream of tokens.

Here is an example. Plain *T_EX* defines

```
\def\centerline #1{\line{\hss #1\hss}}
```

and so the expansion of

```
\centerline{<Title>}
```

is

```
\line{\hss <Title>\hss}
```

and that's it. This is the end of the expansion of the `\centerline` macro. It so happens that `\line` is also a macro

```
\def\line{\hbox to \hsize}
```

and so we obtain

reprinted from Baskerville

Volume 4, Number 3

```
\hbox to \hsize{\hss <Title>\hss}
```

as a subsequent stage from the `\centerline` command. The token `\hbox` refers to a primitive TeX command, which is now executed. Note that if there were control sequences in the `<Title>`, then they will not be executed until TeX is processing the contents of the `\hbox`.

If there is a misspelt control sequence with the `<Title>`, TeX will produce one of its famous multiline error messages, saying that within the expansion of `\centerline` there was an expansion of `\line`, within which there was an expansion of the misspelt control sequence. But because misspelt and thus, presumably unknown, the expansion is to produce an error message. Knuth has new users run through precisely this situation [33]. Did you follow his advice and typeset the story about R. J. Drofnats? I confess that I did not.

The expansion of a macro results in a change in the input stream of tokens. Let us use the word ‘performance’ to mean the end and final result of the expansion and execution of the macro and the tokens contained within, and perhaps their performance also. The expansion of `\centerline` is as above. The execution is to set text in a horizontal box of width `\hsize` and centered. Beginners may be frightened by the line of code

```
\setbox 0=\centerline{Title}
```

but experts will know that this is in fact legitimate, and for why.

Let us now move on to loops. I know that such things are avoided by all except those with tendencies to ovine larceny (I’m struggling to fill the white space at the end of the article) but just suppose we wish to read a sequence of letters and—oh horror—put a small space between each and the next.

There are many ways to do this (letter space, not steal sheep). Without a context there is no right or wrong, although the more bizarre solutions are more amusing and instructive of human psychology than useful. Without further ado, let’s have some examples.

My favourite is admirable in its simplicity. Here it is.

```
\def \spaceit #1{#1\littlespace\spaceit}
```

We assume that `\littlespace` will produce a small space, say by a kern.

Let’s see it in operation. The performance of

```
\spaceit Baskerville
```

begins with the expansion of `\spaceit`

```
B\littlespace \spaceit askerville
```

and then the `B` and `\littlespace` are performed (*i.e.* typeset and added to the current horizontal list), leaving

```
\spaceit askerville
```

which now proceeds as before. This is called “tail recursion” by computer scientists [219]. It is an elegant way of repeating a story (Groan).

All things, even `Baskerville`, will come to an end. We need to find a way of persuading `\spaceit` to stop. One way to do this is to space a sentinel and the end of `Baskerville`, for which `\spaceit` can test with each iteration. I will show how to do this next month.

Testing for the sentinel takes time. In some situations it is better to take a more active approach. Let us look at this. We want

```
\endspaceit
```

to break the `\spaceit` loop, so that

```
\spaceit Baskerville\endspaceit
```

will insert all those `\littlespaces`.

The penultimate expansion of `\spaceit` is

```
\spaceit e\endspaceit
e \littlespace \spaceit \endspaceit
```

and once the ‘e’ and the `\littlespace` have been done we have

```
\spaceit \endspaceit
\endspaceit \littlespace \spaceit
```

and now we go for a dirty trick. With the definition

```
\def \endspaceit \littlespace \spaceit {}
```

the expansion of the previous line is

```
% empty
```

which is just what we want. There we are, a loop without use on any of the control primitives. (It is worth noting that the so called *expansion* of a macro might be *smaller* than its arguments, or even zero.

Finally, solutions and exercises.

Solution 3. *Two tokens have the same meaning. When does the substitution of one for the other make a difference?* For definiteness suppose that we

```
\let \RELAX \relax
```

and then replace some occurrence of `\relax` by `\RELAX`. I know that this example is unlikely, but it serves to express the solution to the problem. It will make a difference in the following situations. Firstly,

```
\string \relax
```

and secondly any assignment such as

```
\let \relax \something
\def \relax { ... }
```

and finally

```
\def \macro { ... \relax ... }
```

should an `\if` or `\meaning` be subsequently applied to `\macro`, and as far as I know, that's it.

Solution 4. *What operational difference is there between*

```
\def\aaa{aaaaaaaa}
\def\xyz{aaaaaaaa}
```

and

```
\def\aaa{aaaaaaaa}
\let\xyz\aaa
```

if any at all was the problem. Macros need memory for their storage, and [383] tells us how much. The second variant will require less main memory (and make for quicker `\ifx` tests I presume) than the first. This is because the `\let` command [206–7] sets the meaning of the first argument (`\xyz`) to be whatever the current meaning of the second (`\aaa`) is. \TeX stores meanings in its memory. The `\let` command sets the meaning pointer for `\xyz` to be equal to (and so point to the same meaning as) the meaning pointer for `\aaa`. Moreover, if the code above itself appears in a macro, this macro will require less storage *and* execute quicker when the second variant is used.

Exercise 5. This comes from the excellent *Around the Bend* puzzle column run by Michael Downes of the American Mathematical Society (email mkd@math.ams.org). The problem is to write a macro which will trim the leading and trailing spaces from user supplied text, such as the parameter text to `\centerline` or `\section`.

Exercise 6. When unexpandable commands are inserted between the letters of a word the kerning and ligatures are lost [19, Exercise 5.1]. Compare 'WAW' to 'WAW'. The second has had `\relax` commands inserted between the letters. Clearly, high class letter spacing (should there be such a thing) will respect the kerning information in the original font. For ligatures it is not so clear, and certainly harder. The problem is to deal with this kerned letterspacing problem. And while you're at it, how do we deal with the trailing `\littlespace` that `\spaceit` will leave at the end of Baskerville.

X Malcolm's Gleanings

Malcolm Clark
m.clark@warwick.ac.uk

1 Macsyma

A brochure for Macsyma arrived the other day. On the back page of this multi-colour leaflet was the statement that 'Macsyma's math expressions look just like those in textbooks'. I hear some of you already 'that's hardly surprising since Macsyma can output in $\text{T}_{\text{E}}\text{X}$ format'. Well, yes it can, but what the advert was extolling was its ability to use MS-Write ('which comes with MS-Windows') to create 'screen displays of large expressions', at which it 'excels'. By this time you will have worked out that I wasn't impressed by the example they give. If I make a list of the infelicities that were displayed you'll think I was making it up. If Macsyma thinks that textbooks look like this it is clear that standards of literacy, mathematics and attention to detail have declined irredeemably. I may have to retire to Tunbridge Wells.

2 As others see us

In the production notes accompanying the Acrobat in Publishing booklet, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ (or Latex) is described as a 'mark up text processing package'.

Rosemary Bailey pointed out to me that in a report entitled 'The Scientific, Technical and Medical Information System in the UK', prepared on behalf of The Royal Society, the British Library and The Association of Learned and Professional Society Publishers, $\text{T}_{\text{E}}\text{X}$ is defined as 'A mark-up language, similar to SGML, compiler and output software, first developed by the American Mathematical Society (AMS), for complex mathematical papers. The files contain standard ASCII characters, and can therefore be transmitted over simple computer networks. Has the ability to cope with all mathematical symbols and can provide high-quality output.'

Rather similarly, Allan Reese remarked in the UKTUG electronic digest that when his letter to the Daily Telegraph was published, ' $\text{T}_{\text{E}}\text{X}$ ' had been changed to 'a text based publishing system'.

3 Colour

I can't really see why there is all this fuss about colour. None of the publishers I have spoken to show much enthusiasm for $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$'s (sort of) new ability to allow us to place colour on the page. There are few enough STM (Scientific, Technical and Medical – the nomenclature for most $\text{T}_{\text{E}}\text{X}$ -friendly publishers like Elsevier, Wiley, etc) books with half tones let alone full colour. Colour is expensive. In my own institution the cost difference between a colour page and a monochrome one is a factor of 12 or so. Fine for the odd page, but hardly enticing. It is true that colour has its place: that place is usually a full colour photograph or two in, for example, medical books, or the dust jacket or cover of a book. Few journals routinely print colour, and when they do, they often levy an additional charge.

So why are people so excited? There are possibly a couple of reasons, and they have nothing to do with paper. The first explanation has been with us for a long time. $\text{S}_{\text{L}}\text{T}_{\text{E}}\text{X}$ has always given the capability of creating slides in colour, although I have never knowingly witnessed slides created in colour by $\text{S}_{\text{L}}\text{T}_{\text{E}}\text{X}$. But given current technologies, colour slides make some sort of sense in two presentational forms: the traditional overhead slide; and through one of these panels which allows the contents of a screen to be projected. In the first case we still have the hassle of printing, but in the second we have something potentially quite useful. Arguably the screen is the cheapest colour system around. It is quite expensive in capital terms, and its resolution is not great (always a worry with Computer Modern which is deficient at 300 dpi, let alone 72–80 dpi), but at least the colour rendition is 'what you see is what you get'. Once you go from screen colours to printed colours you are in trouble, and things seldom come out as convincingly. This is a well known problem, although there are few well known solutions. But stick to the screen and you have something quite convincing. This was brought home to me the other day when I obtained the notes for an Acrobat seminar in paper and electronic form. The paper form was in black, grey and white. The electronic form was in colour. Here was an instant example of added value to the electronic form – at no real additional cost. What fascinates me most is that

this is essentially subversive. Recall that T_EX's clarion call was to produce masterpieces of the publishing art: it seems clear to me that publishing here implied publishing on paper: what we may have is a reason to stick to electronic form because the paper form is less rich – and indeed, can never be as content-full.

One of my reasons for querying the usefulness of colour was the arrival recently on my desk of *Cahiers GUTenberg*, the journal of the French (speaking) T_EX group. This long-awaited volume (lateness seems endemic in T_EX based productions) is a colour issue, and goes into many of the details surrounding PSTricks, Seminar, and colour in general. While I appreciate that if you talk about colour, you really have to use it as well, I find the appearance of the volume similar to the early days of DTP, when the naive had just discovered fonts.

4 Acrobat again

4.1 Acrobat in Publishing

By now you will have realised that I think that Acrobat is a good thing. By and large my enthusiasm was vindicated at the Acrobat in Publishing seminar held in London on May 16th at the Society of Chemical Industry in Belgravia, London. It was there that the penny dropped that there are advantages to including colour. The meeting took the typical form of a number of lectures touching on various aspects of Acrobat technology and a few hands on demonstrations. In addition we were given a 40-page hand-out of the talks, and a Mac or MS-DOS floppy with the text in Acrobat format. One of the things which the meeting lacked, although it was targeted 'for people in the publishing and printing industries' and was claimed to 'explain what Adobe Acrobat technology is', was much explanation of the components of Acrobat. The notes did explain, but it might have saved much confusion if an initial talk had just explained some of the buzz-words and jargon which was about to be unleashed.

The first speaker (a replacement from the published programme) was from UK Mail International (part of the group who publish the Mail, the Mail on Sunday and the Evening Standard). To my surprise they turn out to be very committed to electronic publishing (I really must stop underestimating the right's ability to utilise technology: Conservative they may be; conservative they are not). One proposal was that we may expect to see compilations of back numbers of newspapers on CD-ROM in Smith's (in Acrobat format). The less charitable whispered that they couldn't imagine wanting back editions of the Mail, but the point here was that for hardly any additional cost, the newspaper proprietors have another product. Almost equally, one can go from that and perhaps have the latest edition transmitted to you electronically. They were talking in terms of a typical issue of the Evening Standard taking about 10 minutes to download over a 64kbps ISDN link. They also suggested that there could be added value by including video clips (for example of a winning goal), which are clearly not possible in the 'standard' version. What was never really suggested was that I might just want specific stories, so that I might make up my very own newspaper (this is a suggestion which has been around for some years), or that the format that I might wish to read on my screen might be different from the one that the sub-editors had determined. At least it will be marginally easier reading a tabloid on screen than a broadsheet (but no more easier to print).

Rosie Altoft from John Wiley discussed some of their experiences with Acrobat. Wiley used Acrobat in its beta development days. As you should know, Wiley has been using T_EX for many, many years, and is arguably one of the most electronically aware of our national STM publishers. Part of Wiley's experience with Acrobat derives from their association with the CAJUN project, which itself is part of the fruit of the journal EP-odd (Electronic Publishing: origination, dissemination and design⁶) which they publish. CAJUN and EP-odd will recur in this report. I was not especially clear how much of an advantage Acrobat was to Wiley's. Since they have been dealing with electronic submission for years, any advantages seemed rather incremental rather than revolutionary. The most exciting thing she suggested was the ability to cut down the amount of time and reworking involved in changes to cover design – chiefly through the addition of the 'sticky notes' (or Postit notes) feature. But she did note that their New York office had been involved in an experimental scheme to allow college lecturers to create their own course material by selecting chapters from a wider range of books, and having them printed up into the course book, again through Acrobat. This embodies selection and print on demand (or, at least, very short run printing) – things which are definitely in the pipeline.

Philip Smith of Nottingham University Computer Science Department, *Using Fonts in Acrobat* added to my sum total of Acrobat knowledge in a number of ways. Basically he was describing how Acrobat handled fonts, but not harping on about the Multiple Master Technology, which he rather took as read. At present Acrobat can handle both POSTSCRIPT Type 1 and Type 3 fonts, and TrueType fonts. In passing, in an earlier edition I suggested that Minion and

⁶Those wondering why the name EP-odd was chosen might reflect that the Electronic Publishing conference has a tendency to take place every two years, in even numbered years: a sort of EP-even.

Myriad had been renamed Adobe Sans and Adobe Serif. This appears not to be the case. I have at least one document which has Minion, Sans and Serif. Philip cleared up one point which had been worrying me: how does Acrobat handle non-Latin fonts? Basically it embeds. Embedded fonts are those which, for some reason, Acrobat decides to include with the document, so that rendition is possible. How does it decide? It will not embed the 'standard 14': these are Times, Helvetica and Courier in their four variations, plus Symbol and Zapf Dingbats: it will embed fonts which do not use the Latin (ISO Latin 1) character set: it will always embed Type 3 fonts; all others will be approximated through Multiple Masters. Almost. You can force the 'real' fonts to be embedded if you use Distiller (one of the Acrobat suite). You will appreciate that there is a legal issue lurking in here. Can I legally distribute an Acrobat file which contains an embedded font which has been licensed to me, but which may not have been licensed to the recipient? Firstly, all Type 1 fonts in the Adobe Type Library (which may include those licensed from ITC, Linotype and Monotype) may be 'freely' embedded in Acrobat files (that's a bit woolly to me); secondly, Adobe considers its encryption to be good enough to prevent the unscrupulous from extracting the fonts and using them for other purposes. But beyond that you do run the risk of violating copyright law. The other piece of key information which Philip gave was how to obtain the information about which fonts Acrobat is using for a document. Hold down Shift+Ctrl (on Windows), or Shift+Option (on a Mac) while selecting the *Document Info* item from the *File* menu. A slight catch is that this is a running total: you either have to view the whole document page by page first, or do a search for a word which doesn't exist (that forces processing of each page). Another catch is that if you are viewing your second or third document, their fonts will also be listed.

After lunch, Ian Chivers of Kings College London discussed the use of Acrobat with Ventura. He was concerned with its use in an academic institution, and particularly for the production of large multi-author documents. He noted that Acrobat (in common with other Windows products) was resource hungry, requiring something of the order of a 33 MHz 80486DX with 8 Mbyte of memory for serious work. A Distiller run on a 40 page document took 4 hours on a 20 MHz 80386SX with 5 Mbyte of memory. I was interested to see that Ventura was taking Acrobat quite seriously to the extent of providing the hooks to generate 'bookmarks' for tables of contents and indexes, which Acrobat can subsequently use.

Leon Harrison (again of Nottingham's Computer Science Department) described CAJUN. This acronym stands for *CD-ROM Acrobat Journals Using Networks*⁷. It is in fact a collaborative venture with John Wiley & Sons and Chapman & Hall. A couple of interesting features emerged in this talk. The major product is EP-odd on CD-ROM. EP-odd is archived in L^AT_EX. The text remained constant in the archive, but the macros evolved. However, they were not themselves archived, and when it came to rerun the articles, discrepancies became apparent. The extra value which Acrobat form can add includes links between documents (or to the table of contents, etc) which can be embedded in the L^AT_EX macros. There are some difficulties, since forward references require that they know exactly what point they are to refer to (a common enough problem in L^AT_EX, solved through the .aux file, but requiring some more subtle maneuvering in the POSTSCRIPT which will become Acrobat, apparently). Line art had been redrawn and discarded, requiring some scanning in from page proofs. One of the other journals in the project is *Collaborative Computing*, which is re-keyed into 3B2, although authors may submit in L^AT_EX (given the algorithmic similarity between 3B2⁸ and T_EX this is a bit sad).

The last paper was from the urbane Conrad Taylor, who discussed some of the design issues which were highlighted by Acrobat. He made a number of points on displaying documents which apply quite widely: you can seldom display the whole page and read the body type (I can, because I have an A4 screen and *Textures*, but not all the world is blessed in this way – however, Conrad was actually talking about newspaper formats here, and only would-be newsletters use A4 format, so he's right), and therefore have to zoom and scroll. This becomes tedious. If the document has colour you need 24-bit colour support. Rendering and redrawing can be time consuming, especially for graduated tints and complex vector mapped graphics. If it is indeed a newspaper, the large size prohibits hard copy at the size for which the pages were designed. The diagrams have a level of detail appropriate for litho printing, but not for the screen. He made the observation that it would be more effective to reformat a newspaper before distributing it in Acrobat format, to take account of some of these difficulties. Conrad went on to give an example of designing for paper and the screen. Admittedly, the example he provided will come as no great surprise to (L^A)T_EX people, but it is interesting to see how far his typesetting tool, FrameMaker, has come. Basically he employed some generic and was able to take the same marked up document to produce a screen oriented version and a paper-oriented version. What made this interesting

⁷The acronym CAJUN had been established when Acrobat was actually called Carousel, hence Carousel Assisted Journals Using Networks, but what's in a name?

⁸A Santa Barbara beer to the first person to give me the correct explanation for this name; reveler collects, of course.

was that he did the conversion live, and that Frame supports similar tools to Ventura to allow the implanting of useful links to support table of contents and other navigation aids. Of course, when I say ‘tables of contents’, I don’t just mean that they exist, I also mean that they are electronically linked to the sections to which they refer. It was most agreeable to see Conrad defending and promoting the use of generic markup.

All in all, a most useful meeting, attended by close to 100 people. The venue was good, with excellent facilities both for the social end of the meeting and the presentations. The group might usefully consider using this location, if we can fill it!

4.2 pdf or dvi?

It was at this Acrobat meeting that I started to wonder if Leslie Lamport’s notion some years ago that \TeX should produce POSTSCRIPT rather than dvi was not correct. I had always rejected this notion, partly on the grounds that POSTSCRIPT was a proprietary system, that many non-POSTSCRIPT printers were out there, and that POSTSCRIPT screen previewers were few and far between. Well, POSTSCRIPT is hardly proprietary any more: there are so many clones, and the details have all been published; there are still lots of non-POSTSCRIPT printers, but the availability of GhostScript for all the main platforms (Mac, Unix and pc/Windows) means that this is not a complete barrier. Similarly, the use of GhostView allows POSTSCRIPT to be viewed on the screen (invaluable for those pesky EPS inclusions). The use of GhostScript and GhostView does involve an extra step, but could impose a degree of standardisation which could save much effort. I would argue that the \LaTeX 2 ϵ support for graphics is almost exclusively for POSTSCRIPT graphics, acknowledging the pre-eminence of this system for serious work.

But selecting POSTSCRIPT as the ‘ultimate’ output format does not go far enough. It should be Acrobat (or more correctly, portable document format, pdf). (\LaTeX) should produce pdf. Adobe already produces Acrobat viewers for Mac, DOS and Windows. At present they make a small charge, but I’m fairly confident that they will soon be part of the operating system, or given away with so many applications that we can assume their ubiquity. I was given a Seybold CD-ROM in Acrobat format with an Acrobat reader for Windows at the seminar. I already have one for the Mac which I was given at the launch of Acrobat in London last year. 5D Solutions is producing a freeware Acrobat reader for Unix. One of the advantages of the Acrobat reader is that it will allow you to print to POSTSCRIPT and non-POSTSCRIPT printers – and if the document has been created by Distiller, that means that your embedded EPS will also be printed out. In other words, we have a POSTSCRIPT interpreter in software (just like GhostView and GhostScript). I’ve already commented on Acrobat’s font substitution. Acrobat supports a hypertext framework (pdfmark) which allows navigation through the document. As yet it does not support intra-document links, but that may come in time.

If the NTS (New Typesetting System) project has any imagination, it will see beyond the narrow confines of creating a system to create even finer masterpieces of the publishing art and will eagerly embrace the technologies present here to create a system for practical examples of the *electronic* publishing craft. The opportunities are there. We can only hope their minds are not yet closed.

4.3 Size isn’t important

What is the difference in size between .tex .dvi, .pdf and .ps files? I compared only one file, a draft of the one which contains this column. There are a number of things to watch. A .pdf may contain embedded files, which will obviously make it larger. I used Blue Sky’s Type 1 Computer Modern in my preparation. In theory it should not have been embedded, and my checks indicate it was not (I viewed it on another platform which does not have these fonts – in fact, which does not have \TeX on it). The .pdf figure is from using pdfWriter, not Distiller. I would expect Distiller to produce slightly more compact code. The .ps figure is from dvips on a Unix box. POSTSCRIPT is a notoriously difficult beast to tie down, since what you are probably measuring has more to do with optimisation decisions made by the drivers’ authors.

file	bytes
.tex	32682
.dvi	46728
.pdf	103248
.ps	115412

5 Editor nods

Neither our revered and esteemed editor, nor John Bowsher, need lose sleep that the \TeX logo is restricted by Knuthian fiat to Computer Modern. As long ago as 1986 (*TUGboat*, 7(2), p.101) Knuth had recognised that the kerning and lowering amount for the logo were font specific, even within CM. He went on to say ‘the plain \TeX macros are specifically oriented to Computer Modern fonts. Other typefaces call for variations in the backspacing, in order to

preserve the logo's general flavor'. He then goes on to note that he has typeset the logo in a variant of Times Roman for his *Computer Journal* paper 'and the standard `\TeX` macro worked fine.'

This seems to suggest that (a) Knuth had long ago realised the problem, and (b) he does not feel that the T_EX logo should be restricted to CM (sigh).