Documentation of MacEQ2\TeX, DVILW, and Hcopy2PS

Mårtensson, Bengt

1987

CODEN: LUTFD2/(TFRT-7352)/1–050/(1987)

Documentation of MacEQ2TEX, DVILW, and Hcopy2PS

Bengt Mårtensson

Department of Automatic Control
Lund Institute of Technology
March 1987

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
March 23, 1987

Document Number
CODEN: LUTFD2/(TFRT-7352)/1–050/(1987)

Author(s)
Bengt Mårtensson (appendix by Trevor Darell)

Supervisor

Sponsoring organisation

Title and subtitle
Documentation of MacEQ2TEX, DVILW, and Hcopy2PS

Abstract

This report documents the programs MacEQ2TEX, DVILW, and Hcopy2PS. The report consists of five different parts:

1. MacEQ2TEX. This is a program that translates a Macsyma log file containing typesetting code for the Unix typesetting program Troff/EQN into TEX-code.

2. DVILW. This program translates a TEX DVI-file into PostScript, for printing e.g. on an Apple LaserWriter. This paper is a general users guide.

3. DVILW—Wizards guide. More specialized information on DVILW is collected here.

4. Hcopy2PS. This program translates Hcopy-meta hardcopies, generated e.g. from Simnon, into PostScript, that can be printed on e.g. an Apple LaserWriter, or included in TEX-documents processed by DVILW.

5. "Incorporating PostScript and Macintosh figures in TEX". This paper, written by Trevor Darell, is included as an appendix. It documents the macro package psfig/TEX, which facilitates inclusion of PostScript figures in TEX. This macro package has been adapted to DVILW.

This report supercedes and replaces the reports TFRT-7334 (first half), and TFRT-7335.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
50

Recipient's notes

Security classification

MacEQ2TEX

Macsyma log file to TEX filter

Bengt Mårtensson, October 15, 1986
Last Revised March 1, 1987

This paper documents the program MacEQ2TEX that translates a Macsyma log file, containing typeset commands for the Unix typesetting program Troff/EQN, into TEX code.

1. Introduction

Macsyma can generate typesetting code for the Unix typesetting program Troff/EQN by setting the global variable `typeset` to `true`. In this case, the *d*-lines will be output to the screen and to the log file as EQN-code. MacEQ2TEX is a program for translating a Macsyma log file, containing these typesetting commands, into TEXcode. It will turn `alpha` into α, `alfa` into *alfa**, do correct square roots, fractions, and matrices, and recognize function such as "sin" to be typeset in roman. An example is given in Section 4. For the format chosen by the present macros, the reader is referred to this example.

It is believed that the program should be possible to use without any knowledge of TEX, except for how to call the program and print out the DVI-file. However, the reader is assumed to have an elementary knowledge of Macsyma and TEX.

Both the Troff/EQN code and the generated TEX code are to be considered as "*dumb*", i.e. might contain not to smart line-breaks, layout etc, etc. Therefore, there are two in principle completely different ways of using this program: It can be used to make your log file more beautiful and easy to read; and secondly, it can be used as a decent first iteration for high-quality typesetting. The second iteration you do yourself with your favorite text editor and TEX. To automatically generate "*smart*" type-setting code for mathematical formulas I consider a task for a smaller expert system. The purpose with this program is to generate "*dumb*" code (the `/index` and `/subscript` qualifiers are an exception from this principle, though). Therefore requests to incorporate new features will most likely be treated cold-hearted. Furthermore, "smartification" is probably better done in using super-editor, such as GNU Emacs.

This paper is compatible with the version of MacEQ2TEX that is dated March 1, 1987. The program consists of approximately 1000 lines of Pascal, and runs under VAX/VMS version 4.x.

By default, MacEQ2TEX generates Plain TEX code. If the `/regler` qualifier is selected, MacEQ2TEX generates TEX-code according to the standard used at the Department

* Note that this is text italics, not math italics.

of Automatic Control, namely where the escape-character of TEX is replaced by the exclamation point "!". Furthermore, begin-group and end-group ("{" and "}" in Plain TEX) are replaced by "<" and ">".

2. Function

Basic Operation

The program is run by the command `maceq2tex`[/⟨*options*⟩] *file_name* where *file_name* is the name of the Macsyma log file. Default file-type is `log`. The file name can also be omitted, in which case the default file name is `macsyma.log`. By default, MacEQ2TEX creates a TEXfile with the name *file_name*`.tex`. If the `tex`-option is selected, the command "`tex` *file_name*" is given after completion.

If the `macrofile` or `tex` qualifier is given, MacEQ2TEX will insert a macro definition file, by default `tex$inputs:macsymac.tex` in the output file. It contains macro definitions necessary for TEX to understand the commands generated by MacEQ2TEX.

Qualifiers

Next the different qualifiers will be described. They can be abbreviated as long as the abbreviations are unique.

`/plain` *(Default)* `/regler`

These qualifier determine if PlainTEX-code will be generated, or if the escape character and the begin- and endgroup symbols will be replaced according to the convention above.

`/include` *(Default)* `/noinclude` `/tex`

The `include` qualifier will include the macro file `tex$inputs:macsymac.tex` into the TEXfile as described above. Also the TEX command "`\bye`" will we written at the end of the file. The `/noinclude` qualifier will inhibit this, which is more suitable for generating files for inclusion in documents. The `tex` qualifier will send the generated TEX-file to TEX after completion. The `/tex` qualifier will imply the `/include` qualifier.

`/macrofile=`*file_name*

This makes MacEQ2TEX insert another macro file than the default.

`/index`

This qualifier will convert $a10$ to a_{10} etc. The precise rule is as follows: If a variable consists of letters, followed by a digit and possibly some extra characters, the conversion will take place. Also parameters with names such as "$\%r1$" will be converted.

`/subscript`

This qualifier will convert ki to k_i etc. If a variable consists of exactly two letters, the conversion will take place. This might be desirable in some situations.

`/outfile=`*file_name*

This directs the output to the file *file_name*, instead of the default file, described above. The default file type is `tex`.

Conflicting options are allowed, in which case the rightmost of the conflicting qualifier takes effect. E.g. `/noinclude/include` is equivalent to `/include`. This makes it possible for you to change defaults by defining e.g. `mactex == "''maceq2tex/noinclude'"`.

Qualifiers Not to be Used by Normal Users
There are also some qualifiers that are not to be used by the normal user. They exist for debugging purposes or historical reasons, and might disappear in coming versions.

`/debug`
The `debug` qualifiers will open a log file with the name `debug.log` and will write in it, first the complete conversion list it knows of, then the outcome of every call to `ReadToken` and `TransformToken`. (This behavior might change in the future.)

3. Hints, Discussion, Bugs, Problems, and Possible Improvements

As described in the introduction, there are in principle two different uses of this program. Essentially only the latter one, namely to produce high-quality type-set formulas will be discussed here.

The EQN code looks faily weird on the screen when you are running Macsyma interactively. Therefore, it might be a good idea to first run Macsyma the usual way, then to open the log file, turn on the typesetting and then "playback".

The general idea is to in the generated TEX-file write some general macro call, which the user can (re-) define according to his or her needs or tastes. "Standard" macros are given in Section 5, which will serve as a guide for writing new.

Fairly often, Macsyma's EQN-code generation fails to break an expression (see the example in the next section), and asks you to try to break it yourself with a text editor. Most often, this is simpler to do *before* MacEQ2TEX, in the EQN-code, than in the TEX-code. The reason for this is that MacEQ2TEX will balance occurrences of all left- and right parentheses, braces etc. by inserting the corresponding "`\left.`" and "`\right.`". To introduce a breakpoint, simply write

```
.EN
.EQ
```

on two separate lines of the log file at the place of the desired break.

A particularly "dumb" feature of the generated code is that it uses "`$$`" between all displayed lines in the *d*-lines. This will make them to widely spaced apart if there is more than on line of display in a *d*-line. High-quality type-setting code should instead use the TEX-command `\displaylines` (or equivalent).

MacEQ2TEX typesets names longer than one letter in italics, not math italics. This is done with the TEX macro call `\name`, which is included in the macro file. It also puts a thinspace on both sides of the name. This macro can be redefined according to personal taste.

4. An Example

The following example was run with the `/index` and `/subscript` qualifiers. Note in particular the failure of breaking the long list on line (d7), the overfull `hbox`'es, and the

awfully bad breaks in line (d9). Cf. the comments made above.

```
(c3) a+aa+aaa;
```

(d3) $$aaa + a_a + a$$

```
(c4) x^2+a1*x+a2;
```

(d4) $$x^2 + a_1 x + a_2$$

```
(c5) solve(%,x);
```

(d5) $$\left[x = -\frac{\sqrt{(a_1^2 - 4a_2)} + a_1}{2}, x = \frac{\sqrt{(a_1^2 - 4a_2)} - a_1}{2}\right]$$

```
(c6) x^3+a*x^2+b*x+c;
```

(d6) $$x^3 + ax^2 + bx + c$$

```
(c7) solve(%,x);
 Breakup of expression failed.
 You may try to break it yourself
```

(d7)

$$\left[x = \left(-\frac{\sqrt{3}i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{(27c^2 + (4a^3 - 18ab)\,c + 4b^3 - a^2b^2)}}{6\sqrt{3}} - \frac{27c - 9ab + 2a^3}{54}\right)^{\frac{1}{3}} + \frac{}{9\left(\frac{\sqrt{(27c^2-}}{}\right.}\right.$$

```
(c8) part(1,%);
(c9) part(d7,1);
```

(d9) $$x = \left(-\frac{\sqrt{3}i}{2} - \frac{1}{2}\right)\exp\left(\frac{\sqrt{(27c^2 + (4a^3 - 18ab)\,c + 4b^3 - a^2b^2)}}{6\sqrt{3}}\right.$$

$$\left. - \frac{27c - 9ab + 2a^3}{54}\right),$$

$$\frac{1}{3}\Bigg)$$

$$+\frac{\left(\frac{\sqrt{3}i}{2}-\frac{1}{2}\right)\left(a^2-3b\right)}{9\left(\frac{\sqrt{\left(27c^2+\left(4a^3-18ab\right)c+4b^3-a^2b^2\right)}}{6\sqrt{3}}-\frac{27c-9ab+2a^3}{54}\right)^{\frac{1}{3}}}$$

$$-\frac{a}{3}$$

```
(c10) a:matrix([cos(phi),-sin(phi),0],[sin(phi),cos(phi),0],[0,0,1]);
```

$$\text{(d10)}\quad \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
(c11) c:subst(psi,phi,%);
```

$$\text{(d11)}\quad \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
(c12) b:matrix([1,0,0],[0,cos(theta),-sin(theta)],[0,sin(theta),cos(theta)]);
```

$$\text{(d12)}\quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

```
(c13) a . b . c;
```

(d13)

$$\begin{bmatrix} \cos(\varphi)\cos(\psi)-\sin(\varphi)\sin(\psi)\cos(\theta) & -\sin(\varphi)\cos(\psi)\cos(\theta)-\cos(\varphi)\sin(\psi) & \sin(\varphi)\sin(\theta) \\ \cos(\varphi)\sin(\psi)\cos(\theta)+\sin(\varphi)\cos(\psi) & \cos(\varphi)\cos(\psi)\cos(\theta)-\sin(\varphi)\sin(\psi) & -\cos(\varphi)\sin(\theta \\ \sin(\psi)\sin(\theta) & \cos(\psi)\sin(\theta) & \cos(\theta) \end{bmatrix}$$

5. MACSYMAC.TEX

This is the file `macsymac.tex`:

```
% MACSYMAC --- macros for MacEq2TeX
% Bengt Martensson 86-10-06
% LastEditDate: "Sun Mar  1 12:11:39 1987"
{\obeyspaces\gdef {\ }}
\def\beginMACSYMAlog
{\begingroup
%%%  \def\\{{\char"21\relax}}
  \def\^{{\char"5E\relax}}
  \def\~{{\char"7E\relax}}
  \def\abs{\mathop{\rm abs}\nolimits}
  \let\epsilon=\varepsilon
  \let\rho=\varrho
  \let\phi=\varphi
  \def\csc{\mathop{\rm csc}\nolimits}
  \def\laplace{{\cal L}}
  \def\bold ##1 {\hbox{ {\bf ##1} }}
  \let\oldsqrt=\sqrt
  \def\sqrt ##1 {\oldsqrt{##1}}
  \def\sup ##1 {^{##1}}
  \def\sub ##1 {_{##1}}
  \def\name ##1{\,\hbox{{\it ##1}}\,}
  \begingroup
  \parskip=0pt\parindent=0pt
  \obeylines\obeyspaces%
  \tt}
\def\endMACSYMAlog {\endgroup\endgroup}
\def\beginMacsymaEQ #1%
{\endgroup
  \bgroup
  \def\dotheequationnumber{\leqno{\rm #1}}
  $$}
\def\endMacsymaEQ
{\dotheequationnumber
  $$
  \egroup
  \begingroup
  \parskip=0pt\parindent=0pt
  \obeylines\obeyspaces%
  \tt}
```

6. Revision History

October 20, BM

Vocabulary increased by `cdot` and `abs`. Fixed bug occuring when a double quote (") is not preceded by a space.

November 6, BM

Fixed minor bugs. Increased maximum token length to 35. Increased vocabulary with standard mathematical functions. Fixed bug when "}" is not preceded by a space. Proper handling of `bold`. Changed the handling of names longer than one letter to be typeset in italics, not math italics. Stops gracefully on defect MACSYMA files. `\sqrt` changed.

December 10, BM

Increased vocabulary by ... (transformed to `\cdots`). Changed `sup, sub, from, to`. Macsymac: Definition of `\sup` and `\sub`, `\let\epsilon=\varepsilon` etc. Breaks non-typeset lines after position 70 and after spaces, tabs or commas. Handles spurious eqn/troff commands without quitting.

February 18, 1987

Rewrote token handling. Introduced the qualifiers `/plain`, `/leif` (later renamed to `/regler`), and `/macrofile`. Changed breaking of long, non-typeset lines from column 70 to column 90. Some minor errors corrected. Clean-up.

March 1, 1987

Changed "`/leif`" to "`/regler`". Converted this document to Plain TEX.

Acknowledgement

The command decoding is stolen from Leif Andersson. I am very thankful for being able to use this.

DVILW

DVI to PostScript filter adapted to LaserWriter

Bengt Mårtensson, 860903
Last revised 870312 (BM)

This paper documents the program DVILW that translates a TEX DVI-file to the page-description language PostScript, adapted to the Apple LaserWriter, and optionally prints it. The handling of TEX's `\special`-command is fully documented. This paper is compatible with the version of DVILW which is dated March 12, 1987.

1. Introduction

DVILW is a program for translating TEX DVI-files to the page description language PostScript, adapted to the Apple LaserWriter, and optionally prints it. It is a fairly powerful postprocesser. It allows full access to all fonts resident in the LaserWriter. There are several options for altering the output. The implementation of the `\special`-command in TEX allows full access to PostScript, as well as inclusion of illustrations etc. DVILW asks the user interactively for replacements of missing fonts and `\special` files in the same way as TEX does.

There is an accompanying document "DVILW—Wizard's guide" not meant for ordinary users. It describes installation, font handling, and a more technical description of the PostScript code generated, in particular the `\special`-command.

The ancestor of DVILW was the program DVILGP. The development of DVILW from DVILGP was done by Leif Andersson and Bengt Mårtensson. It is approximately 2000 lines of Pascal, and runs under VAX/VMS version 4.x.

2. Function

Basic Operation

The program is run by the command `dvilw`[/⟨*options*⟩] *file_name* where *file_name* is the name of the DVI-file. Default file-type is `dvi`. By default, DVILW creates a PostScript file with the name *file_name.ps*. If the `print`-option is selected (which is default), the file is printed on the LaserWriter.

Qualifiers

Next the different qualifiers will be described. They can be abbreviated as long as the abbreviations are unique. Conflicting options are allowed, in which case the rightmost of the conflicting qualifier takes effect. E.g. `/noprint/print` is equivalent to `/print`. This makes it possible for you to change defaults by defining e.g. `dvips == "''dvilw'/noprint"`.

`/print` *(Default)* `/noprint`

The `/print` option prints and deletes the PostScript file after completion.

`/portrait` *(Default)* `/landscape`
This selects portrait/landscape orientation of the page.

`/inquire` *(Default)* `/noinquire` `/nospecial`
These qualifiers govern the treatment of the `\special`-command. The `/nospecial` qualifier makes DVILW ignore the `\special`-command. The `/inquire` qualifier makes DVILW inquire for a new file name when a requested `special` file is not found. An empty file name (i.e. just `return`) will make DVILW ignore the file. If the `/noinquire` qualifier is given DVILW will ignore not found `special`-files. (E.g. for running DVILW in batch mode.) The exact handling of the `\special`-command is documented in the next section.

`/bottomspecial` *(Default)* `/topspecial`
Determines the alignment of inserts with `\special`. See the next section.

`/manualfeed`
This qualifier turns on the manual feeding on the LaserWriter. This can be used e.g for feeding letter-head paper, cardboard, and transparancies.

`/copies=`*number_of_copies*
This makes every page be printed *number_of_copies* number of times. The job will come out unordered, but this is much faster than sending multiple jobs. Requests to make less than one copy are ignored.

`/xoffset=`*x_offset* `/yoffset=`*y_offset*
These qualifiers allows the user to specify the x- and y-offset of the page. Unit is mm. The offsets specified with this command are added to any `\hoffset` or `\voffset` values that may have been given in TEX. If all these offsets equal 0, then the distances between the upper left corner of the document and the edges of the paper are both 25 mm. The qualifiers work in the same way both in portrait- and landscape mode. Needless to say, physical limitations restrict the meaningful values.

`/rmagnification=`*magnification* `/amagnification=`*magnification*
These qualifiers modify the magnification of the document. `/amagnification` overrides the magnification, while `/rmagnification` multiplies it. The parameter is divided by 1000, e.g. `amagnification=1200` means a magnification by a factor 1.2. `/amagnification` and `/rmagnification` are considered as conflicting in the sense described above, i.e. the last specified overrides the previous one(s).

`/psmagnification=`*magnification*
These qualifiers modify the magnification of the document by specifying a different scale in the Current Transformation Matrix (CTM) in PostScript. The parameter is divided by 1000, e.g. `/psmagnification=1200` means a magnification by a factor 1.2. If the document contains pixel-fonts, this will most likely not give satisfactory results.

`/terse` *(Default)* `/quiet` `/verbose`
Determines how much is written on the screen. `/terse` writes a short message for each page successfully processed, `/quiet` writes error messages only, and `/verbose` writes page numbers and also the names of the referenced font files.

`/outfile=`*file_name*
This directs the output to the file *file_name*, instead of the default file, described above. The default file type is `ps`. Note that the command `dvilw/outfile=a` *someone_elses_dvifile* will give a possibility for printing a DVI-file residing on a directory on which you do not have write privileges.

`/pages=`*page_range_specs*
This option will produce only the pages in the *page_range_specs*. A *page_range_specs* consists of one or several page-ranges, separated by commas. A page-range is either a number or two numbers separated by colon (`:`). Examples are: a) `5` b) `-7:0` c) `-8:-5,1:5,15,-1:0`. The page-numbers are the actual page numbers printed on the pages by TEX, not the consecutive number in the file. These are not necessarily the same. Note that negative page numbers are allowed. (These will be printed as lower case roman numerals by the PlainTEX-command `\folio`.)

3. Handling of `\special`

The `\special{`*file_name*`}` command in TEX makes DVILW include the file *file_name* in the output file. Several file names can be given, separated by commas, spaces, or pluses. Default file type is `pro`. If *file_name* is not found, then `ps$inputs:`*file_name* is sought for. (Note that `ps$inputs` can be a search list.) Also user-supplied file names (by the `/inquire` qualifier) are sought for in this fashion.

Strings of PostScript code can be inserted by enclosing them in double quotes (`"`). Some simple examples are given in the next subsection. Note that the argument of `\special` is expanded. TEX recognizes the comment sign (`%`) in the argument, but not e.g. dollar signs (`$`).

The options `/bottomspecial` and `/topspecial` determine the alignment. By default (`/bottomspecial`) the lower left corner of the inserted page is placed where the dot presently is located. `/topspecial` exists mainly for compatibility with older versions of DVILW and it is recommended that it should not be used for new documents. The alignment for `/topspecial` is done so that (roughly) the upper left corner of the inserted "page" is placed where the dot presently is located.

More details are given in the accompanying document "DVILW—Wizard's guide".

psfig/TEX

There is a very nice public domain TEX macro package "psfig/TEX" by Trevor Darell. It facilitates the inclusion of arbitrary PostScript figures in TEX documents. Figures are automatically scaled and positioned on the page, and the proper amount of space reserved. For a full description, see [Darell]. It has been adapted to DVILW by this author. Everything described in [Darell] has been successfully converted. There is no need for "`\psfiginit`" described in [Darell]. More details are given in "DVILW—Wizards guide".

Translation, Scaling, and Rotation of PostScript Figures

The possibility to include PostScript code within the argument to the `\special` statement allows access to all the powerful facilities of PostScript. Next some simple examples are given, intended to introduce the reader to simple translations, scalings and rotations.

The line

```
\special{"0.5 0.5 scale",figure.ps}
```

will insert the PostScript illustration in the file `figure.ps` scaled by a factor of 0.5. The line

```
\special{"0.5 1.5 scale",figure.ps}
```

will insert the PostScript illustration in the file `figure.ps` scaled by a factor of 0.5 in the x-direction and 1.5 in the y-direction. The line

```
\special{"100 -20 translate",figure.ps}
```

command will insert the PostScript figure translated 100 units in the x-direction and -20 units in the y-direction. The units are by default bigpoints. (It holds that 1 inch = 25.4 mm = 72 bigpoints.) By the line

```
\special{"-90 rotate",figure.ps}
```

the figure will be inserted after rotating $-90°$ (in the counter-clockwise direction) around the current point. For the final, slightly more advanced example,

```
\special{"10 20 translate -1 2 scale 45 rotate",fig1.ps,fig2.ps}
```

The command will first translate 10 bigpoints in the x-direction and 20 in the y-direction, then scale the x-direction by -1 and the y-direction by 2 (i.e. the picture will come out with the x-axis reversed). Then, after rotating 45°, the figures in `fig1.ps` and `fig2.ps` are inserted. (Just a reminder: translations, scalings, and rotations do *not* commute.)

More details are given in "DVILW—Wizard' Guide". For a full description of the language PostScript, see [Adobe]. A short introduction is given in [Darell].

4. Handling of Fonts

The LaserWriter Plus contains the fonts listed in Figure 1. The fonts marked with (*) are present also in the LaserWriter without "Plus". Your LaserWriter may have more (or less) fonts.

The names listed are the PostScript names. (The case of the letters are significant in PostScript.) The TEXname is formed from these by replacing all letters by lower case and deleting all hyphens ("-"). DVILW recognizes the TEXname for these fonts and creates the appropriate PostScript code.

The following example describes the use:

```
\font\Palatino=palatinoroman scaled 1234
\Palatino
When you see this, you can possibly figure out why Knuth
dedicated his MetaFont book to H.\ Z.
```

```
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Courier (*)
Courier-Bold (*)
Courier-BoldOblique (*)
Courier-Oblique (*)
Helvetica (*)
Helvetica-Bold (*)
Helvetica-BoldOblique (*)
Helvetica-Narrow
Helvetica-Narrow-Bold
Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique
Helvetica-Oblique (*)
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
NewCenturySchlbk-Italic
NewCenturySchlbk-Roman
Palatino-Bold
Palatino-BoldItalic
Palatino-Italic
Palatino-Roman
Symbol (*)
Times-Bold (*)
Times-BoldItalic (*)
Times-Italic (*)
Times-Roman (*)
ZapfChancery-MediumItalic
ZapfDingbats
```

Figure 1. List of PostScript names for LaserWriter Fonts.

will typeset the sentence in 12.34 point Palatino-Roman. Note that all sizes are available, not just a predefined set.

A more detailed description, including implementation details, how to make `.tfm` files, creating new fonts etc. is given in "DVILW—Wizard's guide".

Missing Fonts
When loading a font that is not resident in the LaserWriter, DVILW searches for the largest size equal to or less than the desired size. If there is a discrepancy of more than 1%, a warning will be issued on the terminal.

When a font is not found, the user is inquired interactively for a replacement.

6. Problems and Hints

Inclusion of Macintosh Documents
Inclusion of Macintosh documents are possible. First generate PostScript code describing the document by the following procedure: Start just as if printing the document on the LaserWriter. Then, immediately after clicking "OK", hold the key cloverleaf-F pressed until the text "Generating PostScript file" is echoed. The Macintosh now generates a text-file named "PostScript" on the present system disk. (There are no provisions for giving it another name, so it will overwrite a previous file with the same name.) Then transfer the PostScript file to the host.

When printing the document containing the Macintosh illustration, the LaserWriter must have the Macintosh dictionary downloaded. The simplest way of achieving this is to make sure that a Macintosh job has been sent to the LaserWriter the ordinary way since last power up.

The alignment of the inclusion is done so that the lower left corner of the page corresponds to the TEX point. Inserting a MacDraw illustration should look like `\special{"MacDraw",` *file_name*}, and correspondingly for MacDraft and MacWrite. (This will only affect alignment.)

It should be noticed that this is a use of the Macintosh that is not supported by Apple. The official dictionary at the time of this writing has version number 40, corresponding to LaserPrep 3.1, and there is no reason to believe that it will survive to the end of time.

DVILW works fine with Macintosh PostScript files created for this prolog. The use of older files is strictly disencouraged. Some more comments are given in "DVILW—Wizards Guide".

Large Pages
Pages larger than the usual printing area can be output by printing them repeatedly with different values of `/xoffset` and/or `/yoffset`, and then cut and glued together afterwards. Example: To produce a 300 mm long page you just write it in TEX as if you could print 300 mm long pages. Then print it with the commands `dvilw` *file* and `dvilw/yoffset=-100` *file*. Cut and glue it afterwards.

The LaserWriter Barfs
The major problem presently is not the program, it is the LaserWriter. The size of its virtual memory is simply not adequate for printing complex TEX-document with down-loaded fonts. I have frequently exhausted the virtual memory only by 2–3 pages! Hopefully, there will soon be available more virtual memory. Until then, you have to process the files just a few pages a time. The amount is dependent on the number of fonts you use, and how large they are. Note that if no Macintosh job has been sent to the LaserWriter since power up, the available virtual memory will increase.

References

ADOBE SYSTEMS INCORPORATED (1985): *PostScript Language Reference Manual*, Addison-Wesley, Reading, MA.

DARELL, TREVOR (1987) Incorporating PostScript and Macintosh figures in TEX.

KNUTH, D. E. (1984): *The TEXbook*, Addison-Wesley, Reading, MA.

DVILW—Wizard's Guide

Bengt Mårtensson, March 12, 1987

Introduction

This paper supplements the documentation of the program DVILW. More specialized information is collected here.

Installation

Installation of DVILW: All users should have the following definitions (apart from the usual TEX defintions).

```
$ dvilw == "$[directory]dvilw"
$ lwprint == @[directory]lwprint
$ define tex$afm directory
$ define ps$inputs directory
$ define laser_writer terminal
```

To install the program: Modify the command procedure `lwprint.com` appropriately.

```
$ pascal dvilw
$ link dvilw
$ macro newldriver
$ link newldriver
```

(Or possibly use an existing `.exe` file.) Copy all `.tfm`-files to `tex$fonts`. Copy all `.afm`-files to `tex$afm`. Copy `tex.pro`, `texencodefont.pro`, `dvilw_fonts.dat` to `ps$inputs`.

These files are described on other places in this document.

New tfm and afm files can be generated from DOPL. This program contains its own instructions. (For wizards only.)

More on Some Qualifiers

There are also some qualifiers that are not to be used by the normal user. They exist for debugging purposes or historical reasons, and might disappear in coming versions.

`/errorsonly` `/terse` *(Default)* `/verbose` `/debug`
These qualifiers govern how much debugging information is written on the screen. `/errorsonly` is synonymous with `quiet`. The use of this is less than with other dvi-interpreting programs, since the PostScript file is a human readable text file, which can be read and modified using any text editor.

For unknown reasons, the `/landscape` mode does not handle the pixel fonts satisfactorily. The raster does not exactly fit in. As a quick "fix" to this `/landscape` forces `psmagnification` to be 998. As usual, this can be overridden by another `/psmagnification` specification later on the command line. These qualifier does not exactly commute...

The Generated PostScript Code

The generated PostScript code is roughly "conforming", see [Adobe, Appendix C]. I.e. it contains machine readable comments a printing program can use. For the PostScript interpreter to understand the code generated by DVILW the code has to be preceeded by a prolog-file. This should be present under the name `ps$inputs:tex.pro`. It is based on, but not identical to the prolog file used by DVI2PS. The PostScript definitions necessary for psfig/TEX, [Darell] has been included in the prolog.

Inclusion in Other Documents

In order to allow inclusion in other documents, global changes such as scaling, translation and rotation, there must not be any PostScript command in the file that forces any changes of the graphical state in terms of absolute quantities, only by modifying the old state. Due to what I consider a flaw in PostScript, (no named graphic savesets, and the implicit `grestore` performed by `restore`) this has requirement has been fulfilled only "almost". There are only two PostScript command in the generated PostScript code that is not compatible with this requirement: `initgraphics` and `showpage`, so only these have to be redefined in order to allow inclusion or global changes. Example: writing

```
{
\catcode`{=12\catcode`}=12
\special{@"/oldinitgraphics { initgraphics } bind def
          /initgraphics { oldinitgraphics 15 rotate } def"}
}
```

in the last page of your document will make the entire document, including all `\special`'s, come out rotated 15 degrees counter-clockwise.

`special`-*stuff*

The alignment of `/topspecial` is done so that (roughly) the upper left corner of the inserted "page" is placed where the dot presently is located. More precisely, the point where the dot is located corresponds exactly to the point with coordinates (0,820) in the default PostScript coordinate system.

The PostScript operators `showpage` and `erasepage` are disabled (e.g. redefined to do nothing) in a `\special`. Therefore, plots etc which print by itself should be possible to use without modification.

Ordinarily, the current PostScript status is restored after the execution of the `\special` command. However, if the first character in the argument to `special` is '`@`', then this is not done, and the global changes can be done. The primary use of this is to include prologue-files needed for several inserts. Example: `\special{@plotdict}` will insert `⟨ps$inputs:⟩plotdict.pro` so that all subsequent `\special` files will have access to the definitions in `plotdict`. (Note, however, that the pages are sent to the LaserWriter in opposite order. Therefore, a global insert will have effect on previous pages, but not on the following.)

A `\special` command is either local or global (i.e. starting with a leading "`@`"). Local and global effects are not allowed in the same `\special`.

Next a detailed description of the implementation is given. The reader is assumed to be familiar with PostScript. When `\special` is encountered, first a saveset is created. After pushing the coordinates of the current points on the stack, `initgraphics` is performed, which restores the graphics state to default. If landscape-mode is in effect, the coordinates are rotated and translated accordingly. Then the old coordinates of the current point are used in order to translate the coordinate system according to the description above. Then the file and/or the PostScript code is inserted. After this, the saveset is used in order to restore the graphics state as it was before `\special` was handled.

For global `\special`'s—i.e. when the argument starts with "`@`", as described above—the file and/or Postscript code is not embedded in `save/restore` pairs. Thus it will impact futute save-sets. No `initgraphics` is performed, and no redefinition of `print-page` and `erasepage` takes place.

Macintosh-generated PostScript

If you are using Macintosh generated for older Macintosh prologs than the present one, with version number 40 (this is the `/av 40 def` line), you are asking for trouble. The author only has experience of version 13, and has seen version 36. The most important difference is that the command `cp` (mnemonic for "close page") handles the stack differently. For printing Macintosh figures with version 13, one of two methods can be used: Either include the Macintosh prolog in the same `\special` as the figure that is going to use it, or use a global `\special`, and modify `tex.pro` by removing the line

```
/cp { pop pop pm restore } bind def % Remove if using version 13
```

When printing the file on the LaserWriter, it is necessary make sure that no Macintosh job have been sent to the LaserWriter since power-up.

You have to do this to print page 4 of [Darell].

Handling of Fonts

DVILW handles the usual TEXfonts as `pxl`-files. These are supposed to have file names such as `tex$pxl:[cmr10]cmr10.1500pxl`. There is no provision for alternative pixel areas in the present version.

The LaserWriter Plus contains the fonts listed in Figure 1 in the user's guide for DVILW. The fonts marked with `(*)` are present also in the LaserWriter without "Plus". The names listed are the PostScript names. Note that the case of the letters are significant in PostScript. The list has been obtained by inquiring the LaserWriter directly, so it is guaranteed free of typos. The TEXname is formed from this by replacing all letters by lower case and deleting all hyphens ("-"). DVILW recognizes the TEXname for these fonts and creates the appropriate PostScript code. The necessary information for this is read from a file with the name `ps$inputs:dvilw_fonts.dat`. A line in this file has the following format: It starts with the (PostScript) name of the font. Then the description follows of what is required to define the font, e.g. dictionaries and PostScript code. This should define a font with the same name as the TEX name (except that case is not significant in the TEX names). See the examples below. The description is in exactly the same format as the one used in the special command. In

this way, more esoteric fonts, such as reencoded, outlined, underlined, shaded, etc... can be created without recompilation. See below. Also note that all information has to be on one line only, and no line-continuation is allowed. The program keeps track of the loaded dictionaries, so that no dictionary will be loaded more than once.

Reencoding of Fonts

This subsection will describe the reencoding of the fonts. This is stricly speaking not a part of DVILW but concerns its use, and the use of the resident fonts from TEX.

The encoding scheme used by Plain TEX, described in The TEXbook, Appendix F, differs substantially from the Ascii code. (Ever wrote "<" outside of math mode?) The PostScript operators described below, together with the accompanying program DOPL, which creates `tfm`-files, tries to get as close to this as possible. See Figure 1. The selection of the encoding was done by this author. The goal was to get as close to the PlainTEX coding, and the original coding, where the first goal was given total priority over the second. Therefore, there are duplications, i.e. the same character sometimes appears on more than one place in the table.

More precisely (octal notation will be used, even though I am hexadecimal person normally): Positions 000–012, which in Plain TEX are filled by uppercase Greek letters, are empty. Of the ligatures, of which the Plain TEX knows of "ff", "fi", "fl", "ffi", and "ffl" on positions 013–017, only "fi" and "fl" are present. Fixed pitch fonts, such as Courier, does not have ligatures. (Note that this does not present any problem at all for TEX, since it reads the ligature information from the `tfm`-file.) "j", the dotless "j" in position 021, is not existent in the PostScript fonts, and replaced by "j". (Is the dotless "j" really used for anything? There is not a single example in the TEXbook e.g...) On position 040 Plain TEX has a weird little bar, which as far as I am aware of, only is used to construct "ł" and "Ł". This has (of course?) no counterpart in the PostScript fonts. I have put the PostScript character for "ł" at position 040 instead. All the rest of the positions up to 177 are the same as for Plain TEX.

The accented characters not encoded in the standard PostScript encoding, I have encoded in positions 200–237, 321–340, 344–347, and 354–357. The characters which in the original PostScript encoding resided in a place where something else has been put, have been encoded to the positions 360, 362–364, 366–367, and 374–377. All positions mentioned in this paragraph are left undefined in the original PostScript encoding.

In the generated `pl` and `afm` files, I have defined the character on position 240 to be an invisible character, named `void`. This gives the possiblity to write that character just to get the position defined, without writing anything on the paper. See the subsection on TEX interface.

As opposed to the standard PostScript encoding, all characters in a font are encoded in the encoding described above. (Can this possibly slow down the execution speed? The LaserWriter is not exactly known for its speed...)

The PostScript Operators

The PostScript dictionary file `texencodefont.pro` is normally, but not necessarily, used to define a new font. It contains definitions of the PostScript operators `TeX-Encode-Font`, `Scale-Encode-Font`, and `Outline-Encode-Font`. A call to `TeX-Encode-Font`

should look like

```
Helvetica texencodefont,"/Helvetica /$Helvetica TeX-Encode-Font"
```

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
'000								
'010					fi	fl		
'020	ı	j	`	´	ˇ	˘	¯	°
'030	¸	ß	æ	œ	ø	Æ	Œ	Ø
'040	ł	!	”	#	$	%	&	’
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	¡	=	¿	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	“	]	^	˙
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	˝	~	¨
'200	Á	Â	Ä	À	Å	Ã	Ç	É
'210	Ê	Ë	È	Ó	Ô	Ö	Ò	Õ
'220	á	â	ä	à	å	ã	ç	é
'230	ê	ë	è	ó	ô	ö	ò	õ
'240		¡	¢	£	⁄	¥	*f*	§
'250	¤	'	“	«	‹	›	fi	fl
'260		–	†	‡	·		¶	•
'270	‚	„	”	»	…	‰		¿
'300		`	´	^	~	¯	˘	˙
'310	¨		°	¸		˝	˛	ˇ
'320	—	Í	Î	Ï	Ì	Ú	Û	Ü
'330	Ù	í	î	ï	ì	ú	û	ü
'340	ù	Æ		ª	Ñ	Š	Ÿ	Ž
'350	Ł	Ø	Œ	º	ñ	š	ÿ	ž
'360	"	æ	<	>	\	ı	^	_
'370	ł	ø	œ	ß	|	}	~	{

Figure 1. Font table for reencoded font (Times-Roman).

where `Helvetica` is the PostScript name of the original font and `$Helvetica` becomes the PostScript name of the reencoded font. The PostScript name of a reencoded font is always the same as the TEXname of that font preceeded by a “$”, except that the TEXnames are case-insensitive.

A call to `Scale-Encode-Font` looks like

```
LeftPalatino texencodefont,"/Palatino-Roman /$LeftPalatino
1 0 -2 3 Scale-Encode-Font"
```

This command will define a font with the PostScript name `$LeftPalatino` derived by reencoding `Palatino-Roman` and the applying the linear transformation—remember that computer graphics people for reasons unknown to this author prefers to write a linear transformation on $\mathbb{R}^2$ as multiplying a row vector from the right—given by

$$(x \quad y) \mapsto (x \quad y) \begin{pmatrix} 1 & 0 \\ -2 & 3 \end{pmatrix}$$

(Since you are reading a wizard's guide you can surely figure out why it is called "LeftPalatino", right?)

Finally, there is the PostScript operator `Outline-Encode-Font` which will generate an outlined font. Example:

```
OutHelvetica texencodefont,"/Helvetica-BoldOblique /$OutHelvetica
1 0 0 2.5 20 Outline-Encode-Font"
```

This generates an outlined font with the PostScript name `$OutHelvetica` scaled by the matrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 2.5 \end{pmatrix}.$$

The thickness of the lines will be 20 in the characters coordinate system which "always" means that the k-height is set to 1000. (Since it was an italic font and we rescaled the x-direction by 1 and the y-direction by 2.5, it will come out very lightly slanted, approximately 5°.)

The width of the fonts described in `dvilw_fonts` are read from afm-files residing on the directory `tex$afm`.

To generate new tfm-files requires the program DOPL, described below, and the program TFtoPL, which is distributed together with TEX. DOPL is a program for generating .pl-files (for TEX) and new .afm-files (for DVILW) corresponding to reencoded LaserWriter fonts. The reencoding is done for maximal compatibility with PlainTEX.

The program DOPL is written just to do its job with the afm-files its creator know of for the moment. However, it should be easy to modify. (It is increadibly slow to run—more than 2 CPU minutes for a font.)

When entering an old font name, say `helvetica`, DOPL reads the AFM file `adobe$afm:helvetica.afm`. It is possible to give the generated files a new name, say `newhelvetica`. DOPL assumes a design size of 10bpt for the font, unless given a non-trivial scaling matrix T. The matrix T rescales the font as described above. Also compare the PostScript reference manual Chapter 5. The matrix is entered as four real numbers. Example: [1 0 0.5 2] will rescale the x-direction by 1, the y-direction by 2, and give the font an italic angle of $\arctan(0.5/2)$ (provided that its original italic angle was 0.) DOPL presently does not know what to do when the $(1,2)$-element of the matix is nonzero.

Kerning information in the file `dopl$kerning:newhelvetica.kern` will be inserted verbatim in the pl-file. No error occurs if this file is not found. The file should be in pl-format, with `designunits` = 1. (This is the format TFtoPL generates.) Warning: This possibly overrides some previous entries in the ligature table. Observe the warnings from PLtoTF!

When DOPL is finished with its work it issues the command, (in the example) `@doplcom newhelvetica`. This is intended to run a command procedure. Probably you would like this to run PLtoTF, move things around, etc...

TEX Interface to the Characters

As indicated before, the intention was to achieve maximal compatibility with Plain TEX, not to show that my decisions are better than everyone else's. There are just two things you can do with CMR10 (say) but not (without modification in the TEX macros that is) with the LaserWriter's resident fonts, and that is the Polish suppressed "l"s (`\l` and `\L`, which prints as "ł" and "Ł" respectively), and the dotless-j `\j` i.e. "ȷ". However, there are many things that can be made better than in Plain TEX together with the Computer Modern fonts, in particular the accented characters.

Appendix B presents a TEX macro that defines TEX names for these accented characters. Everywhere when it does not conflict with a macro name in Plain TEX, the PostScript names have been used. In the few cases that there have been a conflict, "PS" have been prepended to the TEX name. It is of course still possible to write `\"a` to get "ä", but `\adieresis` will most likely give a better result, since in the latter case the font designer decided where the dots should go on that particular character, not a simple algorithm designed to work decently with all fonts.*

The file will also redefine some Plain TEX macros in order to better take advantage of this. It also redefines `\l` and `\L`. For the sake of completeness, all characters are given a name.

Finally, the feature/bug of Plain TEX, namely that it does not necessarily update the position before inserting the `\special` code, is fixed by a redefinition of `\special`, which makes TEX print the character 240 (octal) in the Times-Roman font before inserting the `\special` code. This character is invisible, has height, width and depth all equal to 0, so it will only result in the point being defined. (This "trick" can of course also be used for other purposes.) The file in Appendix B also contain some additional comments.

Printing of the PostScript File on the LaserWriter

When this paper and the accompanying user's guide say that DVILW "prints the PostScript file", this really means that it issues the command `@lwprint` *file* `/delete`. This calls the command procedure `lwprint.com` which sends the file to the LaserWriter and deletes it afterwards. It is also possible for the user to call `@lwprint` the usual way

* Knuth writes on page 54 in The TEXbook: "Plain TEX works well enough when accents are infrequent, but the conventions of this chapter are by no means recommended for large-scale applications of TEX to other languages."

from VMS. In that case, the file will be deleted if and only if the `/delete` qualifier is present.

Hints, Tips, and Problems

Proof Mode

A simple proof mode, where DVILW checks that it can handle all referenced fonts is obtained by running DVILW with the qualifier `/page=12345` (where `12345` represents a page number you are absolutely sure is not present in the file). Then DVILW just reads the preamble and opens all font files in the document. A small garbage file is created. If this file is sent to the LaserWriter, no pages will be printed.

What is the Best Implementation of `\special`?

In [Knuth] the command `\special` is defined only to include its arguments in the DVI-file as information to the DVI-handling program. There is also the requirement that the DVI-handler is not allowed to modify the position of the dot during handling of the `\special` command.

Other Future Improvements

The bad behavior in landscape-mode is surely an implementation bug in the LaserWriter and not a bug in DVILW, but anyhow it would be nice to circumvent it. Can it be that the LaserWriter makes a small error when it computes sine and cosine of 270°??

It would be nice to allow several input files. Possibly several pixel-areas?

The program should read the parameters for the PostScript fonts from `tfm`-files instead of from afm-files. This should reduce the number of files you have to have resident on the system. Also, this should eliminate the risk of the information in these files being inconsistent.

Some optimization will reduce the size of the generated PostScript file by 50–20%. ¿¿Transfer some Huffman-coded variant instead?? (The generated PostScriptfile consists to ≈ 30% of spaces.) Possibly some improvement of the execution speed is possible, especially since the program DVILGP runs considerably faster. Possibility to force global `\special` inserts to the beginning of the file?? Provisions for reading `pk`-files. Possibly this information should be transferrred to the LaserWriter instead of the bit-patters corresponding to `pxl`-files.

Appendix. Revision History

86-09-07 (BM) Treatment of special changed to allow multiple files and quoted PostScript strings.

86-09-13 (BM) Changed the initial coordinate transformation. The old one forced absolute numbers into the current transformation matrix (CTM) in PostScript, thereby making it impossible to relocate, scale, rotate, or translate a document in its entirety. Special-handling defined in terms of PostScript coordinates (0,820).

86-10-02 (LA) Corrected error in `setRule` and `setChar`.

86-10-02 (BM) Cleaned up PostScript coordinate transformations. Allowed global `\special`'s. Disabling of `showpage` and `erasepage` in handling of `special`. Search

for \special-files on ps$inputs. The /inquire qualifier introduced. Changed the old name of /special to /noinquire. Introduced warning for substituting pixel-sizes. Allowed use of resident fonts, except for loading of parameters from .tfm-files. Replacement of not found pxlfonts. Handling of multiple page ranges improved. /amagnification and /rmagnification implemented. The prologue modified.

86-10-12 (BM) Small bug in CopyTheString corrected.

86-11-17 (LA) Reads the pxl-files from the logical name tex$pxl instead.

86-12-10 (BM) Change in initialization and in prolog in order to allow insertion of Macintosh generated PostScript, from LaserPrep 3.1. @MacSetUp included in prolog, stripping ctrl-d from special files (in Includespecial), Including @MacSetUp in @beginspecial.

87-01-06 (BM) Fully implemented the handling of the PostScript fonts by reading the afm-files, and the file dvilw_fonts. Introduced /psmagnification.

87-02-06 (BM) Inquire for missing fonts, case insensitive fontnames, only one PostScript definition per resident font.

87-02-04 (LA) Changed meaning and default of x- and y-offset.

87-02-11 (BM/LA) Introduced /topspecial and /bottomspecial. Fix (but still not satisfactory) for /landscape (alters psmagnification).

87-02-28 (BM) Small change of the echoing on the screen: "!special" replaced by \special, and "Outputting PostScript file" only when verbose or debug.

87-03-09 (BM) Modified the prolog (@bop0 and @end). Included the psfig/TEX stuff in the prolog.

87-03-09 (LA) Changed the command to print the PostScript file.

87-03-12 (BM) Modified prolog: added MacDraw etc, changed some def's to bind def for execution speed. Fixed bug when the position is not properly updated before a \special command.

Appendix B. The Macro File LWDEFS

```
%%% LWDEFS
%%%
%%% Some definitions which will better take advantage of
%%% the (reencoded) LaserWriter's fonts.
%%%
%%% Bengt Martensson March 1, 1987
%%%
%%% LastEditDate "Wed Mar 11 21:06:49 1987"
%%%
\def\LWdefs
{%%% Redefinitions from Plain
%
\chardef\AA='204                        % On these, we can do better
\chardef\aa='224                        % than Plain
\chardef\L='350
\chardef\l='370
%
```

```
\chardef\dag='262                       % Plain gets all of these
\chardef\ddag='263                      % from the math character set.
\chardef\S='247                         % We can do better.
\chardef\P='262
%
%%% Readymade accented letters (use e.g. \Adieresis instead of \"A)
\chardef\Aacute='200
\chardef\Acircumflex='201
\chardef\Adieresis='202
\chardef\Agrave='203
\chardef\Aring='204
\chardef\Atilde='205
\chardef\Ccedilla='206
\chardef\Eacute='207
\chardef\Ecircumflex='210
\chardef\Edieresis='211
\chardef\Egrave='212
\chardef\Oacute='213
\chardef\Ocircumflex='214
\chardef\Odieresis='215
\chardef\Ograve='216
\chardef\Otilde='217
\chardef\aacute='220
\chardef\acircumflex='221
\chardef\adieresis='222
\chardef\agrave='223
\chardef\aring='224
\chardef\atilde='225
\chardef\ccedilla='226
\chardef\eacute='227
\chardef\ecircumflex='230
\chardef\edieresis='231
\chardef\egrave='232
\chardef\oacute='233
\chardef\ocircumflex='234
\chardef\odieresis='235
\chardef\ograve='236
\chardef\otilde='237
\chardef\Iacute='321
\chardef\Icircumflex='322
\chardef\Idieresis='323
\chardef\Igrave='324
\chardef\Uacute='325
\chardef\Ucircumflex='326
\chardef\Udieresis='327
\chardef\Ugrave='330
\chardef\iacute='331
\chardef\icircumflex='332
\chardef\idieresis='333
\chardef\igrave='334
\chardef\uacute='335
\chardef\ucircumflex='336
\chardef\udieresis='337
\chardef\ugrave='340
\chardef\Ntilde='344
\chardef\Scaron='345
\chardef\Ydieresis='346
\chardef\Zcaron='347
\chardef\ntilde='354
\chardef\scaron='355
\chardef\ydieresis='356
\chardef\zcaron='357
```

```
%
%%% More characters, first some standard ASCII characters
\chardef\quotedbl='360
\chardef\less='362
\chardef\greater='363
\chardef\PSbackslash='364              % Called 'backslash' in PostScipt
\chardef\asciicircum='366
\chardef\underscore='367
\chardef\PSbar='374                    % Called 'bar' in PostScript
\chardef\braceright='375
\chardef\asciitilde='376
\chardef\braceleft='377
%%%
\chardef\void='240                     % Void character, prints nothing
%
\chardef\cent='242
\chardef\sterling='243
\chardef\fraction='244
\chardef\yen='245
\chardef\florin='246
\chardef\currency='250
\chardef\quotesingle='251
\chardef\guillemotleft='253
\chardef\guilsinglleft='254
\chardef\guilsinglright='255
\chardef\guillemotleft='253
\chardef\periodcentered='264
\chardef\PSbullet='267                 % Called 'bullet' in PostScript
\chardef\quotesinglbase='270
\chardef\quotedblbase='271
\chardef\guillemotright='273
\chardef\ellipsis='274
\chardef\perthousand='275
}
```

Hcopy2PS

Hcopy Meta to PostScript filter

Bengt Mårtensson, March 11, 1987

This paper documents the program Hcopy2PS that translates a Hcopy meta file to the page-description language PostScript, for printing e.g. on the Apple LaserWriter. A CTRL-C version named CC2PS is also presented.

1. Introduction

Hcopy meta is a simple device independent format for describing hard-copy from plotting programs. It was defined by Tomas Schönthal at the Department of Automatic Control, Lund Institute of Technology, for the programs Simnon, Idpac, Synpac etc. Hcopy2PS is a program for translating hcopy meta files to the page description language PostScript [Adobe], to be printed on a PostScript device such as the Apple LaserWriter, or to be included in other documents. It is a powerful program, not just a "dumb postprocessor". It contains several options for altering the plot by changing the appearance of the lines, the size, the characters etc. Of course, all these effects can be obtained by modifying PostScript code with a standard text editor. However, this program will make it much simpler, and possible for the user not familiar with PostScript (the reference manual is over 300 pages). It can also be used as "dumb postprocessor", requiring no particular knowledge of the user, but the program is intended to be able to produce figures of the finest quality with lots of knobs to turn in order to satisfy special requirements. The author believes that it is a good idea to implement the Hcopy meta format in new programs, thereby allowing manipulation with the full power of Hcopy2PS.

The present paper is a reference manual, not a primer. It is not a hard program to handle, even though this paper seems frightenly long.

In the next section, the use of the program and the different options and qualifiers are presented. Note that in order to describe all qualifiers there, the text necessarily has to contain a few forward references. Section 3 presents two examples. Section 4 describes in detail customizing and modifying the plots in order to satisfy special requirements and tastes. Some hints etc. are collected in Section 5. In the last section, a CTRL-C version named CC2PS is presented. (CTRL-C is a program for matrix computation etc. [CTRL-C]). CC2PS acts on CTRL-C's pen files. Appendix A presents the standard dictionary. Appendix B contains the definition of the Hcopy Meta format. Permanently down-loading PostScript dictionaries is discussed in Appendix C.

This paper is compatible with the version of Hcopy2PS and CC2PS that is dated March 11, 1987. It is my intention that this paper should be updated when the programs are updated. They are approximately 800 lines long Pascal programs, and run under VAX/VMS version 4.x.

2. Function

Basic Operation

The program is run by the command `hcopy2ps`[/⟨*options*⟩] *file_name* where *file_name* is the name of the hcopy meta file. Default file-type is `p`. The file name can also be omitted, in which case the default file name is `meta.p`. By default, Hcopy2PS creates a PostScript file with the name *file_name.plo*. If the `print`-option is selected (which is default), the file is printed.

Qualifiers

Next the different qualifiers will be described. They can be abbreviated as long as the abbreviations are unique.

`/print` *(Default)* `/noprint`
The `/print` option prints the PostScript file.

`/portrait` *(Default)* `/landscape`
This selects portrait or landscape orientation of the page.

`/manualfeed`
This qualifier turns on the manual feeding on the Apple LaserWriter. This can be used e.g for feeding letter-head paper, cardboard, and transparencies. This is, in contrast to the rest ot the commands, device dependent for the LaserWriter, and will probably not be meaningful on other PostScript devices.

`/xoffset=`*x_offset* `/yoffset=`*y_offset*
These qualifiers allows the user to specify the x- and y-offset of the page, i.e. the distance between the lower left hand corner and the edges of the paper. (Really, the translation of the default origin for the device.) Unit is mm. Default is 20 mm for both x- and y-offset. The qualifiers work in the same way both in portrait- and landscape mode. Needless to say, physical limitations restrict the meaningful values. Note that if a paper format other than A4 is used, the x- and y-offset will probably be off by a constant value in landscape mode.

`/magnification=`*magnification* `/height=`*height*
These qualifiers modify the magnification of the document. *magnification* is divided by 1000, e.g. `magnification=1200` means a magnification by a factor 1.2. *height* is the height of the whole plot-able area on paper, corresponding to the whole span of y-coordinates. These commands modify all dimensions in the image evenly, including line widths, text, and digits. These two qualifiers are just two different ways of expressing exactly the same thing. *magnification* = 1000 (which is default) corresponds to *height* = 124 (mm).

`/xstretch=`*x_stretch* `/ystretch=`*y_stretch*
These qualifiers "stretches" the x- and the y-dimensions respectively. As above, *x_stretch* and *y_stretch* are diveded by 1000. Line widths and font sizes are not affected.

`/outfile=`*file_name*
This directs the output to the file *file_name*, instead of the default output file, described

above. The default file type is `plo`. Note that the command `hcopy2ps/outfile=a` *someone_elses_hcopy_metafile* will give a possibility for printing a hcopy meta-file residing on a directory on which you do not have write privileges.

`/dvilw`
This qualifier is equivalent to `/xoffset=10/yoffset=10/magnification=800/noprint`. It is believed to be reasonable when including the plot in TEX using the DVI-handling program DVILW [Mårtensson 1986b] (which was used for this report).

`/noprolog` `/prolog=`*other_prolog_file*
The `/noprolog` qualifier inhibits the prolog file to be included in the PostScript file. The `/prolog` qualifier allows you to use another prolog than the standard one, if you for example want a non-standard pen-pattern. See Section 4.

`/parameters` *(Default)* `/noparameters`
If `/parameters` is given, Hcopy2PS writes all its computed parameters, determined by the options described in this section, after the prolog-file. Therefore, if the prolog file assigns values to these parameters, they will be overridden by the parameters Hcopy2PS writes. `/noparameters` will inhibit this, which requires the prolog to contain definitions of all these. See Section 4.

`/alone` *(Default)* `/include`
The `/alone` qualifier will make the file to print by itself by calling the PostScript operator `/showpage` at the end of the file. `include` prohibits this, and is meant for plots intended for inclusion in other documents. (Note, however, that the TEX DVI-handling program DVILW [Mårtensson 1986b] disables `showpage` in the `special` command and therefore makes it harmless to have a `showpage` too much in the file.)

`/convertexps`
This qualifier makes Hcopy2PS convert expressions such as 3.E6 to $3 \cdot 10^6$.

`/horizontalfont="`*font_spec*`"` `/verticalfont="`*font_spec*`"` `/digitfont="`*font_spec*`"`

These qualifiers allows the user to use other fonts for the horizontal and vertical text, and for the digits. Note that the double quotes (") are mandatory. *font_spec* is of the form *font_name:font_size*, where *font_name* is the name of any font for the moment known to the PostScript device. A *font_size* of 0 is allowed, which will suppress all text in that font. This makes a very convenient way of getting rid of the sometimes annoying date and "`hcopy meta`" line by defining "`horizontalfont="a:0"`".

The LaserWriter Plus contains the resident fonts listed in Figure 1. The fonts marked with (*) are present also in the LaserWriter without "Plus". Note that the case of the letters is significant. (The list has been obtained by inquiring the LaserWriter directly, so it is guaranteed to be free of typos.)

font_size is a size of the font measured in bigpoints (bpt). It holds that 1 bpt = 1/72 inch. The defaults fonts are 12 point Helvetica for horizontal- and vertical font, and 12 point Symbol for digit-font. When the plot is rescaled by `/magnification=` or (equivalently) `/height=`, the defaults are rescaled accordingly.

Hcopy2PS considers a string of horizontal text as a digit string if it starts with a digit and the rest consists only of digits and the characters "." and "E". This is to allow both hyphens ("-") and minus-signs ("−") in the same plot. The only font resident in the LaserWriter that contains proper minus-signs instead of hyphens is "Symbol".

```
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Courier (*)
Courier-Bold (*)
Courier-BoldOblique (*)
Courier-Oblique (*)
Helvetica (*)
Helvetica-Bold (*)
Helvetica-BoldOblique (*)
Helvetica-Narrow
Helvetica-Narrow-Bold
Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique
Helvetica-Oblique (*)
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
NewCenturySchlbk-Italic
NewCenturySchlbk-Roman
Palatino-Bold
Palatino-BoldItalic
Palatino-Italic
Palatino-Roman
Symbol (*)
Times-Bold (*)
Times-BoldItalic (*)
Times-Italic (*)
Times-Roman (*)
ZapfChancery-MediumItalic
ZapfDingbats
```

Figure 1. List of PostScript names for LaserWriter Fonts.

Conflicting options are allowed, in which case the rightmost of the conflicting qualifier takes effect. E.g. `/noprint/print` is equivalent to `/print`. This makes it possible for you to change defaults by defining e.g. `metaps == "''hcopy2ps/noprint'"`.

3. Examples

The installation here of the very nice public-domain program GNUPLOT has a Hcopy Meta driver, written by this author. Hcopy2PS has generated Figure 2 from one such file.

As a further example we show the Simnon plot on page 17 in [Åström]. This has been produced by the command

```
hcopy2ps/horizontal="ZapfChancery-MediumItalic:15"
/ver="AvantGarde-Book:12"/dvilw vdpol
```

(The different scaling chosen on the axes is due to changes in the Simnon version.)

4. Prolog Files and Customizing

The PostScript file is a human readable text file, allowing inspection and modification with a standard text editor. It is "conforming", e.g. containing machine readable comments as described in [Adobe, Appendix C].

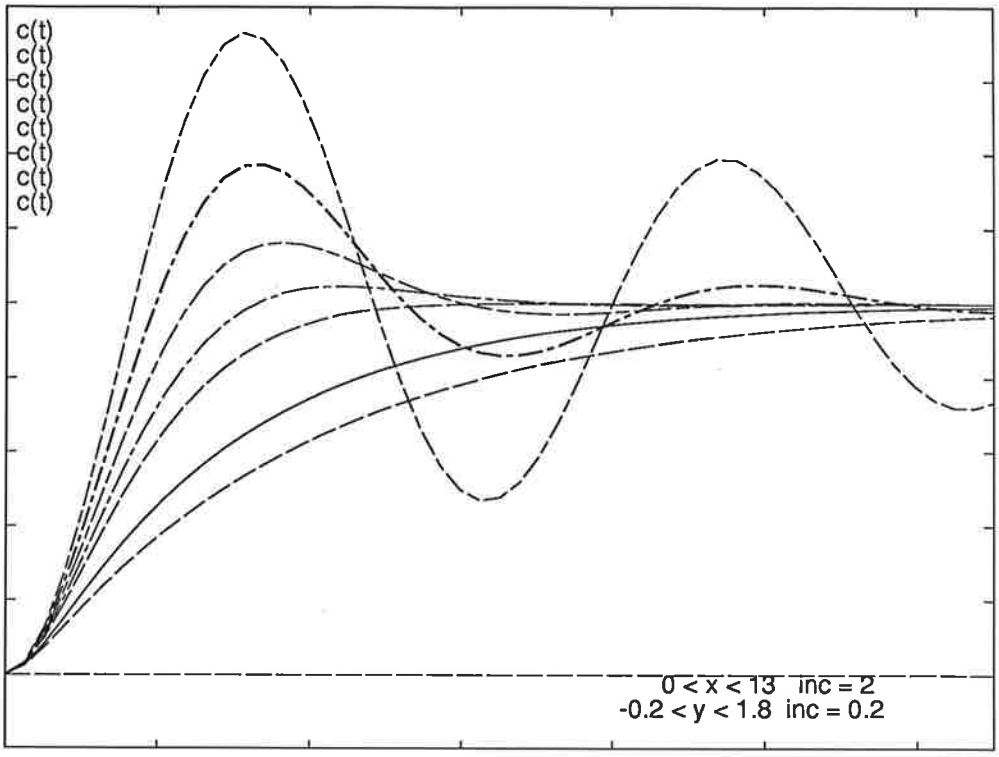


Figure 2. A figure by GNUPLOT.

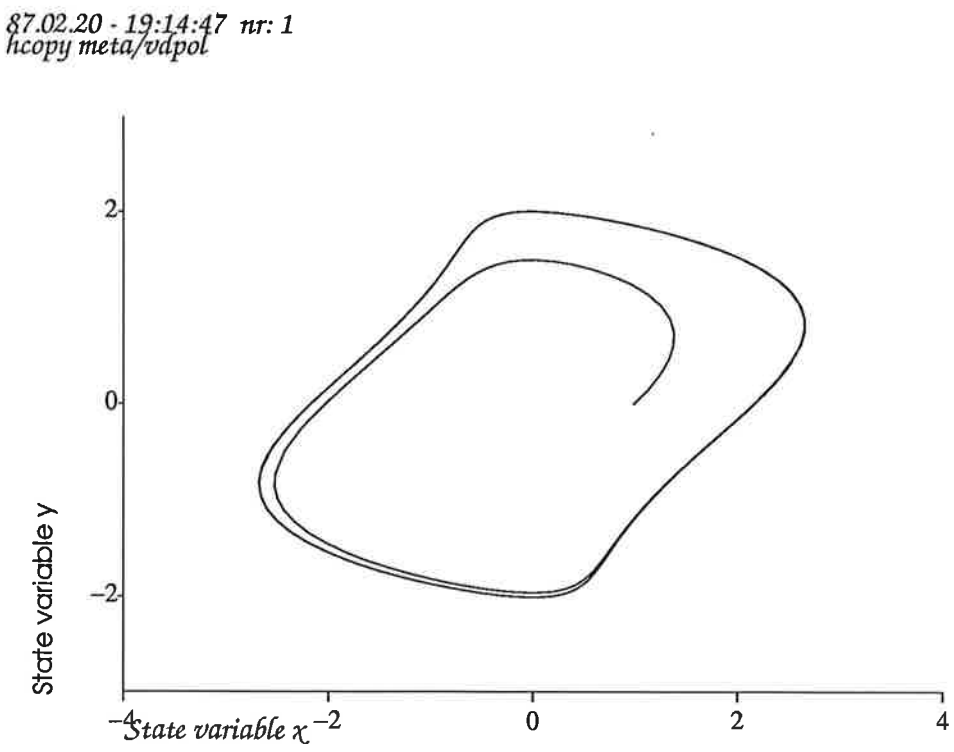


Figure 3. The figure on page 17 in [Åström].

For the PostScript interpreter (e.g. in the LaserWriter) to understand the commands in the plot-file, the plot has to be preceded by a short prolog file, containing definitions of the commands used by the plot. Unless told otherwise by different qualifiers (described in Section 2) the prolog file `ps$inputs:plotdict.pro` will be prepended to the PostScript file.

The qualified user can use his own prolog with the `/prolog=` qualifier, or by specifying `/noprolog` and then insert another prolog file, e.g. when the plot is included in another document. By specifying `/noparameters`, Hcopy2PS does not include any of its parameters in the generated file; therefore the prolog file must in this case contain a specification of them.

The Prolog

Next we describe the standard prolog, listed in Appendix B. To fully understand the description, working knowledge of PostScript is of course required, but hopefully not in order to e.g. modify the line widths. Standard stack-manipulation notation will be used for describing the call of the operators.

The first two non-comment lines defines a new dictionary and starts to use it. Then the two parameters `digit-x-offset` and `digit-y-offset` are defined. These are two constants, in the plot's coordinate system, which will offset everything that is typeset with the digit-font. These parameters allow fiddling with the position of all the digits simultaneously.

Then the array `pen-width-array` of 8 elements is defined. The uppermost entry is the width (in bigpoints) of pen number 1 and so on. The array `pen-cap-array` follows next. This affects the line caps, i.e. how the lines end. Every entry is either 1, 2, or 3. "0" represents a "butt cap": the stroke is squared off at the endpoint of the path; there is no projection beyond the end of the path. "1" stands for a round cap: a semicircular arc with diameter equal to the line width is drawn around the endpoint and filled in. Finally, "3" is a "projecting square cap": the stroke continues beyond the endpoint of the path for a distance equal to the half of the line width an is squared off.

`pen-dash-array` is declared next. The declaration in Appendix B means the following: For pen 1, make it solid. For pen 2, first draw 100 units "black", then 25 units "white". Then repeat. For pen 3, first draw 100 units "black", then 25 units "white", then 50 units "black", then 25 units "white". Then repeat. And so on. The units are the plot's original coordinates.

Then follows the definition of the operator `set-pen` (*pen* $\mapsto$), implementing the "pen"-shift. This assigns values to the linewidth, the dashing pattern, and the line cap according to the arrays described above. The line-join (see [Adobe, p. 218]) is set to 1, meaning "round join", which seems to be the only reasonable when drawing e.g. graphs of smooth functions.

The operator `hcopy-end` ($\mapsto$) follows next. It calls the operator `showpage` if the variable `print-page-when-done` is `true`.

`hcopy-initialize` ($\mapsto$) sets up the basic graphics state, and defines the coordinate system. This is done so that all plotting will be in the plot's native coordinate system.

For this, the prolog uses the two variables `scaling` and `default-scaling`. Hcopy2PS computes `scaling` to be the size of one unit in the plot's coordinate system, expressed in bigpoints (= 1/72 Inch). `default-scaling` is given the value `scaling` would have if the plot is not rescaled by `/magnification` or (equivalently) `/height`. `true-h-font` ($\mapsto$) is defined to be an operator that when called sets up the appropriate font, appropriately scaled, for horizontal text. Also `true-v-font` ($\mapsto$) and `true-d-font` ($\mapsto$) are defined for vertical text and for digit-string respectively. `true-d-exp-font` ($\mapsto$) is defined as the same font as for writing the digits, but 70% of the size. This is meant for exponents.

The operator `d` (*string* x y $\mapsto$) is defined to print the string *string*, which is assumed to be a string describing a number, at position (x+`digit-x-offset`, y+`digit-y-offset`).

The operator `d-exp` (*string exp-string* x y $\mapsto$) is similar to `d`, but differs since *exp-string* is treated as an exponent, meant for printing numbers as $3 \cdot 10^6$. The latter is achieved by the call `(3) (6) x y d-exp`. The line `(\32710) show` prints "·10" in Symbol, and (probably) "10" with another font.

The operator `h` (*string* x y $\mapsto$) is defined to print the string *string* as horizontal text, starting at the position (x, y).

The operator `v` (*string* x y $\mapsto$) is defined to print the string *string* as vertical text, starting at the position (x, y).

The operator `s` (*pen* $\mapsto$) calls `set-pen` and stokes the current path.

Finally, `l` (x y $\mapsto$), `m` (x y $\mapsto$), and `n` ($\mapsto$) are defined as abbreviations for `lineto`, `moveto`, and `newpath` respectively.

Parameters in prolog file

In order to be run with the `/noparameters` qualifier, the prolog has to contain definitions of the 12 parameters Hcopy2PS ordinarily writes to the file. This can look like

```
/x-offset 10 def
/y-offset 10 def
/scaling   0.0899 def
/default-scaling   0.1127 def
/h-font /ZapfChancery-MediumItalic def
/h-font-size 15 def
/v-font /AvantGarde-Book def
/v-font-size 12 def
/d-font /Symbol def
/d-font-size 12 def
/print-page-when-done true def
/landscape false def
```

(The parameters are exactly the ones used for Figure 3.) These have all either been described abover, or are obvious.

5. Hints, Tips, Discussion, and Problems

The PostScript file is a human readable text file, allowing inspection and modification

with a standard text editor. This gives a vast freedom in editing the plots. It is e.g. very simple to locate and remove unwanted text (such as the date and the "hcopy" on the plot), change the scales, etc. A common problem in Simnon is that lines a priori know to be vertical comes out slanted in the plot. These can simply be "straightened up" in the PostScript file.

Scaling

The digits marking the scaling on the axis do not always occur in the right position. The hcopy meta file includes information about how wide etc. it considers its digit to be (see Appendix B). This information is presently discarded, since the plotting routines never was done with high-quality typesetting in mind anyhow. Manual paste-up with a text editor is of course possible, and recommended for high-quality applications. (This was done in [Mårtensson, 1986a].)

Different Pens

Hcopy2PS knows of eight different pens that the hcopy plots might include. These eight different pens all have their own parameters for linewidth, dashing patterns, line-ends etc. These are not accessible as qualifiers in Hcopy2PS, but the prolog file can be edited in order to satisfy special needs or tastes.

The programs Simnon etc. draws with several pens in a fairly disoriented fashion—first it draws a little bit with one pen, then changes and draws a little with another pen, then changes back ... Hcopy2PS orders these paths so it only draws one single time with each connected curve drawn with the same pen. This makes dashed lines of high quality possible. However, due to an implementation limit in the LaserWriter, no path is allowed to consist of more than 1500 line segments. Therefore, Hcopy2PS will insert a break after 1400 line segments, which will occasionally disturb a dashed line.

6. CC2PS—CTRL-C to PostScript Filter

CC2PS is a version of Hcopy2PS that produces PostScript code from CTRL-C's pen files. This is a program derived from Hcopy2PS by just some minor modifications. The operation and the qualifiers are all the same as for Hcopy2PS, even though some are meaningless, such as e.g. the font-selecting qualifiers.

(Actually, it is the same text file, where the CC2PS specific parts normally are commented out. A GNU Emacs function removes all that is Hcopy2PS specific, and uncomments the CC2PS specific parts.)

Acknowledgements

CC2PS was created by using a template written by Mats Lilja. The command decoding part of the program has been stolen from Leif Andersson.

References

ADOBE SYSTEMS INCORPORATED (1985): *PostScript Language Reference Manual*, Addison-Wesley, Reading, Massachusetts.

ÅSTRÖM (1985): "A Simnon Tutorial," Report CODEN: LUTFD2/(TFRT-3176)/1–87/(1985), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

CTRL-C (1983): *CTRL-C User's Guide.*, Systems Control Technology, Inc. Palo Alto, California, USA.

MÅRTENSSON, B. (1986a): "Adaptive Stabilization," Ph.D. thesis CODEN: LUTFD2/(TFRT-1028)/ 1–122/(1986), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MÅRTENSSON, B. (1986b): "DVILW—DVI to PostScript filter adapted for the LaserWriter," Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, To appear.

Appendix A. The dictionary PLOTDICT.PRO

This is the PostScript dictionary `plotdict.pro`.

```
%!PS-Adobe-1.0
%%Title: plotdict
%%CreationDate February 12, 1987
/hcopy-dictionary 32 dict def
hcopy-dictionary begin
/digit-x-offset 0 def
/digit-y-offset -25 def

/pen-width-array
0.4
0.4
0.4
0.7
0.4
0.4
0.4
0.4

8
array
astore
def

/pen-cap-array
0
1
1
0
1
1
1
1

8
array
astore
def

/pen-dash-array
[]
[100 25]
[100 25 50 25]
[100 25 25 25]
[150 25 50 25]
[150 25 25 25]
[150 25 100 25]
[200 25 25 25]

8
array
astore
def
```

```
/set-pen
{ 1 sub
  dup dup
  pen-width-array
  exch
  get
  default-scaling div
  setlinewidth
  pen-cap-array
  exch
  get
  setlinecap
  pen-dash-array
  exch
  get
  0
  setdash
  1 setlinejoin
} bind def

/hcopy-end
{ print-page-when-done {showpage} if
} bind def

/hcopy-initialize
{ landscape {270 rotate -840 10 translate} if
  x-offset 25.4 div 72 mul
  y-offset 25.4 div 72 mul translate
  scaling dup scale
  /true-h-font
    h-font findfont
    h-font-size default-scaling div scalefont
  def
  /true-v-font
    v-font findfont
    v-font-size default-scaling div scalefont
  def
  /true-d-font
    d-font findfont
    d-font-size default-scaling div scalefont
  def
  /true-d-exp-font
    d-font findfont
    d-font-size default-scaling div 0.7 mul scalefont
  def
} bind def

/d
{ set-pen
  moveto
  digit-x-offset digit-y-offset rmoveto
  true-d-font setfont
  show
} bind def

/d-exp
{ set-pen
  moveto
  digit-x-offset digit-y-offset rmoveto
  exch
  true-d-font setfont
  show
```

```
  (\32710) show
  0 (0) stringwidth pop rmoveto
  true-d-exp-font setfont
  show
} bind def

/h
{ set-pen
  moveto
  true-h-font setfont
  show
} bind def

/v
{ set-pen
  moveto gsave currentpoint translate 90 rotate
  true-v-font setfont
  show grestore
} bind def

/s
{ set-pen
  stroke
} bind def

/l { lineto } bind def
/m { moveto } bind def
/n { newpath } bind def
end
```

Appendix B. The HCOPY META format

This appendix documents the Hcopy meta format. The "7"-record is an addition by this author. The appendix is based on text written by Tomas Schönthal.

The meta file is a sequential, variable record length text file. It contains plot primitives, which are stored as one type record, followed by zero, one or two parameter records.

Initialization Record:
6
`ixmin,ixmax,iymin,iymax,nxch,nych,idxch,idych,minpen,maxpen`
Fortran format: (I1/I4,9(1X,I4))
`ixmin` — Smallest possible x-coordinate
`ixmax` — Largest possible x-coordinate
`iymin` — Smallest possible y-coordinate
`iymax` — Largest possible y-coordinate
`nxch` — Character width
`nych` — Character height
`idxch` — Horizontal character spacing
`idych` — Vertical character spacing
`minpen` — Smallest possible pen (color) number
`maxpen` — Largest possible pen (color) number
Note:
— (ixmin,iymin) corresponds to the lowest left corner of the display
— The size of a character cell is (nxch+idxch)*(nych+idych) points
— Hcopy2PS requires the first record to be an initialization record. Only one initialization record permitted by Hcopy2PS.

Pen Shift Record:
5
Fortran format: (I1/I2)
`ipen` — Pen (color) number

Horizontal Text Record:

1
buff(i), i=1, .., nch
Fortran format: (I1/I2/80A1)
nch —Effective length of string
buff —String, declared as character buff(80)

Line Record:
2
ix,iy
Fortran format: (I1/I4,1X,I4)
ix, iy — Destination coordinate pair

Move Record:
3
ix,iy
See LINE

Vertical Text Record:
4
buff(i), i=1, .., nch
See HTXT

End of Page Record:
7
Marks the end of a page description and (possibly) the start of a new page in the same file.

Appendix C. Down-loading plotdict

plotdict can either be loaded once for every plot sent to the LaserWriter printer, or it can be permanently down-loaded. (Also note the possibility of "global" specials in DVILW [Mårtensson 1986b].) The following PostScript file will permanently (i.e. until power-off) down-load plotdict.

```
0000
/#1 where
{pop pop(#1 in place - not loaded again\n)print flush stop}
{dup serverdict begin statusdict begin checkpassword
 {(#1 downloaded.\n)print flush exitserver}
 {pop(Bad Password on loading #1\n)print flush stop}ifelse
}ifelse
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here comes the dictionary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
(Finished downloading #1\n)print flush stop
```

Note that the new features in DVILW [Mårtensson 1986b] turns TEX into a powerful preprocessor for PostScript code, very suitable for creating files such as the one above.

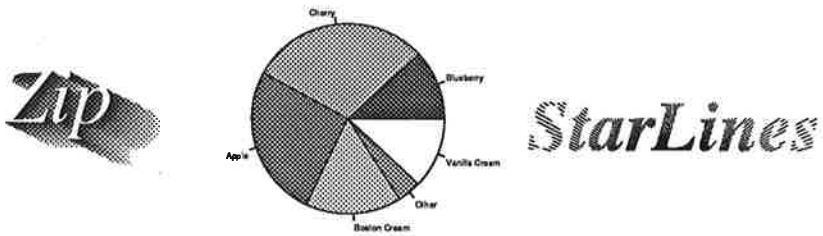
Incorporating PostScript and Macintosh figures in TEX

Trevor Darrell

January 9, 1987

Abstract

psfig/TEX is a new macro package for TEX that facilitates the inclusion of arbitrary PostScript figures into TEX documents. Figures are automatically scaled and positioned on the page, and the proper amount of space is reserved. For example:



Custom characters such as '⌧' and '◓' may be created and used freely throughout a document. Since the Macintosh drawing applications produce PostScript, they can be used to create figures.

1 Introduction

The TEX typesetting system is a powerful tool in the preparation of the written word, yet when the time comes to add figures or pictures to a document, one traditionally had to resort to tedious manual paste-up. With the advent of the PostScript page description language, which allows the 'nesting' of environments and is rapidly becoming a *de facto* standard, it is now possible to merge graphics directly into a document. psfig provides the facility for the nested inclusion of a PostScript figure in a TEX document.

2 Including a figure

To include a PostScript figure with psfig, simply load the psfig macros at the beginning of your document with

`\input{psfig}`

then invoke the macro

`\psfig{figure=`*input*`}`

where *input* is the name of a PostScript file. psfig will automatically position the figure at the current place on the page, and reserve the proper amount of space in TEX so that it doesn't conflict with any other objects.

For example, if we have a file called 'piechart.ps' which contains the PostScript code to draw the chart in the abstract, and it was the first figure in our non-page reversed document, we would use the commands

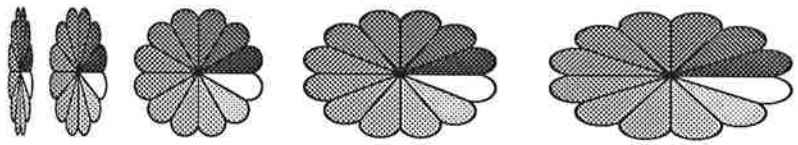
```
\input psfig
\centerline{\psfig{figure=piechart.ps}}
```

Since no mention of size was made in the above example, psfig draws the figure at its natural size (as if it was printed directly on a PostScript printer.) The pie's natural size is several inches across, which is a little large; the pie in the abstract was produced with:

```
\centerline{\psfig{figure=piechart.ps,height=1.5in}}
```

The `height` option specifies how tall the figure should be on the page. Since no `width` was specified, the figure was scaled equally in both dimensions. By listing both a `height` and a `width`, figures can be scaled disproportionatly, with interesting results.

For example:



was produced with:

```
\centerline{\hbox{
\psfig{figure=rosette.ps,height=.8in,width=.15in}
\psfig{figure=rosette.ps,height=.8in,width=.35in}
\psfig{figure=rosette.ps,height=.8in}
```

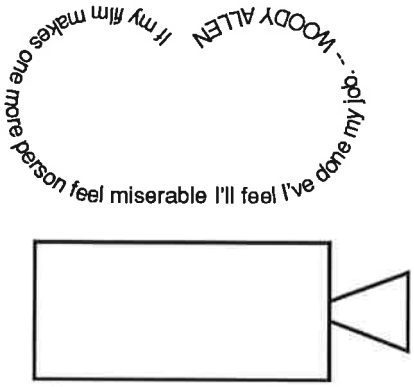


Figure 1. a PostScript figure

```
\psfig{figure=rosette.ps,height=.8in,width=1.2in}
\psfig{figure=rosette.ps,height=.8in,width=1.5in}}}
```

2.1 Caveats

For psfig to find the natural size of a figure, the figure must have a proper bounding box comment; see section 5 below. Draft mode (also detailed below) should be used liberally to speed throughput. Also, some versions of LaTeX will fail to properly center a lone figure in a centering environment; a good work-around is to precede the figure with a hard space. On very large documents with many figures, the printer memory allocated to dvips may have to be limited. Finally, the `\psfig` macro will be confused by extra white space or newlines in its argument.

3 Generating PostScript

Before you can include a figure, however, you must create the PostScript file that describes it. The most common methods for creating a PostScript figure are to use either a drawing application such as MacDraw, an image-to-ps or textronix-to-ps translator, or to directly code PostScript by hand. A brief PostScript tutorial is included as an appendix.

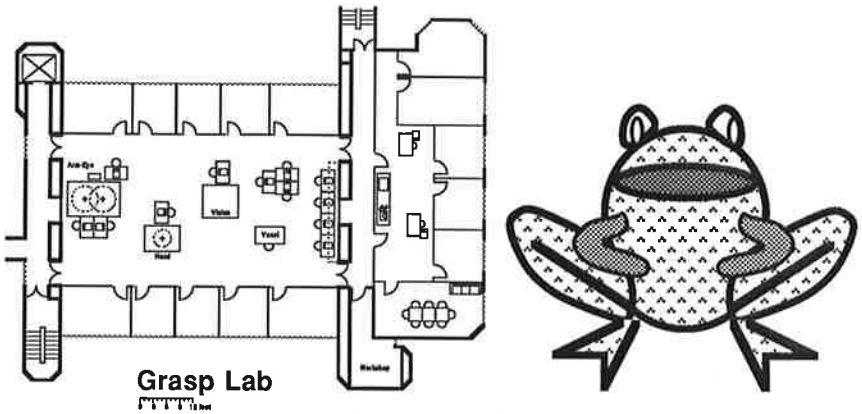


Figure 2a & 2b: Macintosh figures.

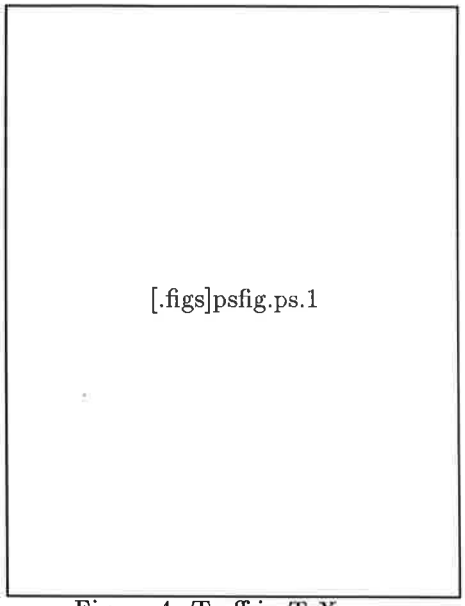
3.1 Macintosh files

Using a Macintosh (or any other system that supports mouse-based drawing utilities that use PostScript) is one of the easiest ways of creating a figure (figure 2a.) MacDraw is the recommended tool, since it produces code that is independent of scaling (as opposed to MacPaint, which produces a bitmap of the figure.) There are several known methods of capturing a MacDraw/MacWrite figure in PostScript form and transferring to the TEX host; most involve some mucking about with tricky control sequences, one is detailed in the appendix.

MacDraw creates a output file in the form of 'QuickDraw' calls, which are interpreted as a set of PostScript procedures. These procedures are defined in what is called the 'macintosh prolog', which must be prepended to any macintosh file to be sent to the printer. There is a `prolog` option in the psfig macro to specify a file that should be prepended to the figure. The name of the prolog is, of course, site dependent; we have used '/usr/lib/ps/mac.pro'. For example, if you had a file 'frog.mac' that contained the macintosh code to draw kermit (figure 2b), he could be included with:

```
\psfig{figure=frog.mac,prolog=/usr/lib/ps/mac.pro}}
```

If there are many such figures, it is probable that the repeated inclusion of the mac.pro file will cause a significant increase in file size and transmission time. An alternative method is to load the mac.pro file once globally, so that it will be available throughout the rest of the document. Use `\psglobal{\usr\lib\ps\mac.pro}` at the beginning of your document to achieve this effect. For this to work properly, the `\psglobal` must be before any Macintosh figures, and the final output



[.figs]fancyimage.ps

Figure 3: A bitmap image.

[.figs]psfig.ps.1

Figure 4: Troff in TEX

must not be page reversed. [1]

3.2 Images (ph), plot, and other sources

Any program that produces PostScript as output can be used for psfig figures. For instance, the *ph* program will convert a bitmap image to PostScript form and thus can be used to include an image in a document (figure 3.)

There are similar utilities that can convert files from unix plot(5) or Textronix 4014 format into PostScript. The Unix text processor *troff* produces PostScript, so one can slice a page out of a Troff document and include it in a TEX paper (figure 4.) Note that the *troff* page was processed by the *troff* counterpart to psfig, and itself contains two images and several PostScript tricks.

4 Draft figures

Certain PostScript figures (such as large bitmap images being transmitted at 9600 baud) can tie up a slower PostScript device such as an Apple LaserWriter

[1] It is possible to use psglobal in page reversed document; place it just before the last figure in your document. This is living dangerously, and you do so at your own risk.

for quite some time. To circumvent this, a figure may be printed in draft mode, which will occupy the same space on the page as far as TEX is concerened, but it will just print the name of the file from which the figure is derived, and will not actually include it. The macro `\psdraft` will switch into draft mode, and all subsequent psfig macros will produce draft figures. The macro `\psfull` will switch out of draft mode.

5 Bounding boxes

To properly translate and scale a figure psfig must know its 'natural' position on the page; this information is present in what is called the *bounding box* of a PostScript program. The bounding box is a outer limit to the marks created by a program, and is specified as four coordinates of a rectangle: the lower-left x coordinate (bbllx), the lower-left y coordinate (bblly), the upper-right x coordinate (bburx), and the upper-right y coordinate (bbury). Adobe has defined a convention whereby the bounding box of a program is contained in a 'bounding box comment'. [2] This comment, which must be present in any file to be used as a psfig figure, is a line of the form

`%%BoundingBox:` *bbllx bblly bburx bbury*

All values are in PostScript points, relative to the *default* transformation matrix. The only mandatory PostScript convention is that the first line of the file should begin with the characters '%!' (a '%' begins a comment in PostScript.) A good place for the bounding box comment is as the second line of the file.

There is a *bbfig* utility on systems to aid in calculating the bounding box, refer to the *bbfig(l)* manual page for further information.

6 Advanced topics

6.1 psfig internal structure

In including a figure, the `\psfig` macro performs the following operations: First, if bounding box information (see below) is omitted from the list of arguments, the file containing the figure is searched and the information recovered from the bounding box comment. Then, if both *height* and *width* are missing they are computed to be the height and width of the bounding box. If only one is

[2] See 'Appendix J: PostScript File Structuring Conventions' in *The PostScript Language Reference Manual*

missing, it is set to be the appropriate value such that there is no distorted scaling. If *rheight* or *rwidth* (see below) is missing it is presumed to be the same as the height and width.

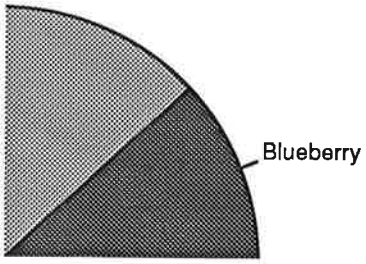
The `\psfig{figure=`*input*`}` macro uses a vbox in TEX to reserve the space. The actual inclusion of files is preformed with a `\special` command to the *dvips* postprocessor. Presently, *dvips* is the only supported post-processor, but it shouldn't be to difficult to port psfig to a different postprocessor, presuming similar capabilities and/or access to source code. psfig depends on certain PostScript function calls; these are downloaded with the `\psfiginit` macro. Do not use page-reversing on your output if you are manually initializing psfig with `\psfiginit`, or are using `\psglobal`. It is possible to include these definitions into the standard dvips (or whatever) header file; this has been done on our systems and users need not do a `\psfiginit`.

6.2 Reserved size

There are two sizes associated with each psfig figure: the size at which it is to be printed on the page and the size it reserves in TEX. This latter size is appropriately termed the *reserved size*, and is expressed as clauses of the form "`rheight=`*dimen*" and "`rwidth=`*dimen*". If ommited, the reserved size defaults to the real size. Some special effects need to be transparent to TEX and thus have a zero reserved size, such as the grey box enclosing this paragraph.

6.3 Clipping

Normally a PostScript program can be expected to not mark the page outside its bounding box. If this is not the case, or if you want to use the bounding box to isolate part of a larger figure, there is an option that sets the PostScript clip path so that no marks will show up outside the declared bounding box. Currently this is invoked by adding a clause of the form "`clip=`". Here a slice has been taken out of the pie chart in the example by specifying a smaller bounding box and adding the clip option.



Some PostScript programs use the clipping path to position their output on the page; if a figure is being drawn at its natural size and position despite psfig commands to the contrary, it may need the clip option.

6.4 PostScript environment

The PostScript environment within psfig is fairly robust. All of the usual PostScript operators will operate as desired; in particular the operators 'showpage', 'initgraphics', and 'defaultmatrix' will all behave consistently inside a figure, except that 'showpage' will only do an 'initgraphics' and will *not* print or erase the current page.

It is very possible, however, for a PostScript program to bypass the psfig environment and disrupt a document. These cases are infrequent, and a 'workaround' solution can usually be found.

7 Acknowledgements

Ned Batchelder co-developed the original *troff* version of this program with the author, and was responsible for much of the overall design. Greg Hager provided an initial TEX implementation. Figure 1 and the three broken out figures in the abstract were taken from examples in *The PostScript Language Tutorial and Cookbook.* The image in figure 3 was designed by Kamran Manoochehri, rendered with *CARICATURE*, by Cary Phillips.

Appendix A: PostScript Overview

Coding PostScript by hand gives you the most control possible over the shape and appearance of the figure; for example, few conventional document preparation packages could produce the graphic in figure PostScript is a stack-oriented language, very similar to Forth and RPN in the way that arguments are handled, yet it features strong typing and higher level control structures. It has an advanced imaging primatives, based on a *stencil and paint* imaging model. Objects are rendered on the page by applying *paint*, which can be any color or sampled image, through a *stencil*, a closed geometric path that limits where the paint should go. For example, a line can be described as applying black paint through a long thin stencil. This generality is preserved throughout PostScript, including in its treatment of fonts, but that does not prevent PostScript implementations from taking advantage of special cases for efficiency considerations. Rarely would an actual PostScript interpreter draw a line by creating a thin rectangle and performing a fill operation; what is important is that its behavior can be perfectly characterised as if it had.

PostScript has a full complement of data types, operators, and control structures. In general, arguments are pushed on an operand stack, then popped off and acted on by an operator or procedure. When PostScript encounters an object it can't execute immediately (such as a number or a string), it is pushed onto the operand stack. Thus, to provide parameters to an operator, simply list them before the operator name. Thus PostScript (like forth), is a prefix language. All expressions are thus unambiguous, and the syntax is very efficient to interpret.

For instance, arithmetic is easy:

```
2 2 add
```

would leave a '4' on the operand stack. Similarly, to add $\frac{1+2}{2+3}+13*9$, you would type

```
1 2 add 2 3 add div 13 9 mul add
```

In addition to numbers and operators there are several different types of objects that PostScript supports. A string, for instance, is denoted by text enclosed in parenthesis. An array of objects is denoted by brackets, and a procedure by braces. An identifier that is not a number, string, or composite object is a *name* object. There are two types of name objects, *executable*, meaning the value or procedure associated with the object will be evaluated immediately, and *literal*, which is not evaluated. A literal name has a '/' prepended to it, and is pushed directly on the operand stack, while an executable name is looked up

on the *dictionary stack*, and its associated value placed on the operand stack. A *dictionary* is a data structure that associates *key* - *value* pairs. All operators, variables, and procedures are referenced through the dictionary stack, which essentially establishes the hierarchical naming environment in PostScript.

There are always at least two entries on the dictionary stack: the system dictionary and the user dictionary. The system dictionary contains all the bindings for the built-in operators, while the user dictionary holds user variables and procedures. A PostScript program is free to modify the user dictionary (and to add new entries on the stack), but can not write to the system dictionary.

Assigment is with the `def` operator, which takes a literal name and value and stores it in the topmost dictionary; to initialize π you could say

```
/pi 3.14 def
```

After this `def`, typing the executable name `pi` will cause PostScript to look up that name in the dictionary stack and place 3.14 on the operand stack. [3] For instance,

```
pi r 2 exp mul
```

would evaluate πr^2 (of course, if `r` had not been defined, an error would have been generated).

The PostScript language has a rich collection of operators, including the usual stack operators (pop, dup, exch, ...), arithmetic operators (add, mul, sin, ...), and control operators (if, loop, for ...), as well as more advanced operators for manipulating dictionaries, arrays, strings, and files.

While PostScript has all the capabilities of a general purpose language, it is first and foremost a page description language, and as such has a whole range of operators that manipulate the graphics state and place marks on the page. All values given to graphic primitives in PostScript are transformed through the *current transformation matrix (CTM)* before any marks are made on the page. Thus the CTM establishes the scale and orientation of the coordinate system in which a program will run. The default PostScript CTM produces a coordinate system in PostScript points (PostScript points are the same as 'big points' (bp) in TEX- 72 to the inch) with the origin in the lower left hand corner of the page. The CTM can be scaled, rotated, or translated dynamically by a PostScript program.

[3] The value need not be numeric; it may be a string, array, or procedure, among other things.

```
newpath
50 50 moveto
150 50 lineto
100 150 lineto
closepath stroke
showpage
```

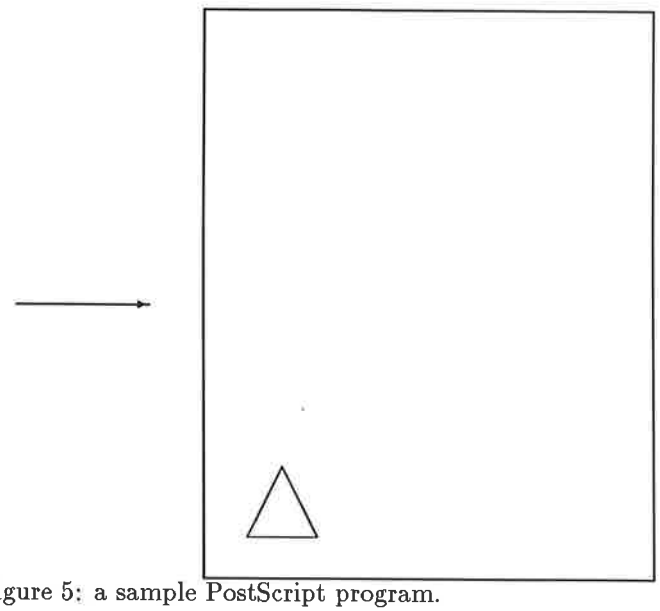


Figure 5: a sample PostScript program.

An important object in the graphics environment is the *current path*, an internal PostScript data structure that keeps track of geometric shapes. The current path is composed of an arbitrary number of connected or disjoint line segments, curves, and Bezier cubics.

The **newpath** operator clears the current path, and the **moveto** operator will move the current point to an arbitrary location. **Moveto** takes two arguments, an x and y location on the page, so to move to (307,397) you would say **307 397 moveto**. Exactly where (307,397) is on the page depends on the CTM; with the default matrix, it is roughly at the center of the page.

From the current point, an **x y lineto** will add a line segment to *(x,y)* on the current path, or a **closepath** will add a segment back to the first point in the path. To create arcs and circles, a **x y r ang1 ang2 arc** will add a counterclockwise segment from *ang1* to *ang2* of a circle of radius *r* whose center is at *(x,y)*. Note that none of these commands actually mark the page; they just build up the path structure. Two operators which will mark the page according to the current path are: **stroke**, which will draw a line along the current path, and **fill**, which will paint the region enclosed by the current path. Figure one depicts a sample PostScript program and its result.

Text can be equally as simple; first, you must set up the current font.

```
/Times-Roman findfont 10 scalefont setfont
```

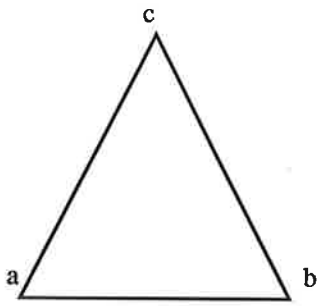
would set the current font to be ten point roman. A string can be printed at the current point using the **show** operator. In PostScript strings are delimited by parenthesis.

When a PostScript program has completed putting all useful marks onto a page, it should execute the **showpage** operator, which causes the printer to print and eject the current page.

Thus, to label the vertices in our triangle, we could modify our program as follows

```
newpath
50 50 moveto
150 50 lineto
100 150 lineto
closepath stroke
/Times-Roman findfont 10 scalefont setfont
45 55 moveto (a) show
155 55 moveto (b) show
95 55 moveto (c) show
60 35 moveto
(The Triangle ABC) show
showpage
```

which would produce:



The Triangle ABC

This overview was only meant to give a flavor of the PostScript language, and therefore has only touched on the simplest of it's commands. For a more thorough introduction, consult the *PostScript Language Tutorial and Cookbook* and *PostScript Language Reference Manual* from Adobe Systems.

Appendix B: Capturing PostScript files from a Macintosh

In general, a PostScript file can be transferred from a Macintosh to another host using any of the popular terminal emulators and a serial line. We have used MacTerminal and Kermit without any problems.

Slightly trickier is getting the PostScript into a file on the Macintosh. For MacDraw and MacWord (and perhaps others), there is an undocumented "feature" whereby the PostScript code can be diverted into a file rather than being sent to a printer. Immediately after clicking 'ok' from the print menu, hit clover–F; the code will be placed in a file with the name "PostScript" (there is no known way to change this). Clover-K will capture the file *and* the lengthy prolog mentioned above.

NAME

psfigTeX – PostScript figures in TeX

SYNTAX

\input **psfig**

DESCRIPTION

psfigTeX is a package that allows easy inclusion of arbitrary PostScript code into a TeX document. Figures are automatically scaled and translated, and the correct amount of space is reserved so nothing conflicts with other text on the page. The *dvips* postprocessor must be used to create the final PostScript output.

USAGE

To include a postscript file, invoke the psfig macro "\psfig{figure=fig.ps}", where "fig.ps" is the name of a PostScript file. Options may be specified in the form "\psfig{figure=fig.ps,opt=val,opt=val...}"; recognized options include height, width, prolog, and postlog. If a height but not a width is given or vice-versa, the figure will be scaled equally in both dimensions; if niether is given, the figure's "natural" size will be used.

For *psfig* to properly scale and move a figure, it must be able to tell what it's natural size and position on the page are. This is usually specified by the bounding box comment in the PostScript file. Unfortunately, some applications (including MacDraw) do not provide this information. If your figure doesn't have a bounding box comment (a line starting with %%BoundingBox:), it cannot be used by psfigtex. The bbfig utility can calculate the bounding box of a file (so can a ruler). See the bbfig(l) manual page for information on the correct format of the comment. Usually the bounding box comment is the second line of the file.

MACINTOSH FIGURES

Macintosh figures require a prolog file to be downloaded containing PostScript procedure definitions. Use the option "prolog=/usr/lib/ps/mac.pro" to achieve this. See the psfigtex paper for more detailed information, especially if there are many mac figures.

DRAFT MODE

If a figure is very expensive to print (say a 100K image) it can be set in draft mode, printing just the name of the file. The macro \psdraft switches into draft mode and \psfull returns to full mode. Use of draft mode is highly encouraged.

BUGS

The "\psfig{...}" command must be entirely on one input line, and no extra spaces may appear in the option list.

When a "\psfig" command is used alone in a centering enviornment, it must be preceded by a hard space "\ ". This may be a LaTeX bug.

On very large documents, the laserwriter has been known to run out of memory and only print a portion of the document. Use the printer memorysize command to dvips to limit the amount of memory dvips thinks it can use.

AUTHOR

Trevor Darrell

SEE ALSO

TeX(1), dvips(1), LaTeX(1)
DVILASER user's manual
T. Darrell, *Incorporating PostScript and Macintosh figures in TeX*