CONTEXT

METAPOST Support

group: CONTEXT Extra Modules

version: 1996.10.22

date: 1997 July 25

author: Hans Hagen

METAPOST is both an extension and a dialect of METAFONT and is written by John Hobby. This language combines a lot of the the strength of METAFONT, TEX and POSTSCRIPT in one package: a sophisticated graphics language, high quality typography and portability based on outlines. First we need some auxiliary macro's.

```
1  \ifx \undefined \writestatus \input supp-mis.tex \relax \fi
```

For some reason, METAPOST needs the public domain DVI to POSTSCRIPT converter DVIPS. This symbiosis originates in the need to include the fonts (glyphs) that METAPOST uses in the POSTSCRIPT file. Driver independancy was one of my prerequisites for using METAPOST, so I decided to build this kind of support myself. Personally I consider driver dependancy a drawback for the dissemination of such a package. This module more or less decouples METAPOST and DVIPS.

The next three commands do the job. They may be called more than once:

```
\UseMetaPostFile      {filenaam}
\UseMetaPostProofFont {fontname}
\UseMetaPostGraphic   {fiilename}
```

For testing purposes there is:

```
\ShowMetaPostData
```

And for troublesome situations, like independant page processing we've got:

```
\ReUseMetaPostData
```

This module is independant of CONTEXT, therefore we start with:

```
2  \ifx \undefined \writestatus \input supp-mis.tex \fi

3  \unprotect
```

The process for generating a METAPOST illustration looks more or less like:

```
mptotex filename.mp filename.tex
tex filename.tex
dvitomp filename.dvi filename.mpx
mp filename.mp
```

The file `filename.tex` contains the TEX specific commands and ships out a page for every piece of text in the metaposting. These files look like:

```
\shipout\hbox{\smash{\hbox{\hbox{% line 10 examples.mp
$a$}\vrule width1sp}}}
\shipout\hbox{\smash{\hbox{\hbox{% line 11 examples.mp
$b$}\vrule width1sp}}}
\end{document}
```

Where the last line takes care of both PLAIN TEX and LATEX, and kindly ignores all other formats that redefined `\end`, which makes it a bit more implementation dependent. By the way, there also seems to be a dependency on `\voffset` and `\hoffset`, which should be 0pt while shiping out.

An alternative to the next solution could be a utility that generates a decent prologue file based on the `filename.mpx`. For the moment we stick to the TEX based solution. There are two methods, the first one uses the intermediate TEX file, created by METAPOST, the second one uses the graphic files themselve. The latter method is the most secure.

\UseMetaPost..

The TEX file can be used to determine what fonts and glyphs are needed. We only have to take care of the \shipout and \end. The next command stores the glyphs in a box:

```
\UseMetaPostFile{examples}
```

We reserve a box for this manipulations and append successive calls to \UseMetaPostFile to this box. The local redefinition of \shipout takes care of the METAPOST ones, the global redefinition is needed later on. We need to reset \everypar, otherwise unwanted side effects can occur.

4
```
\newbox\MetaPostData
```

5
```
\def\UseMetaPostData#1%
  {\global\setbox\MetaPostData=\vbox
     {\everypar{}
      \unvbox\MetaPostData
      \hbox{#1}}%
   \global\let\shipout=\MetaPostShipOut}
```

6
```
\def\UseMetaPostFile#1%
  {\UseMetaPostData
     {\let\shipout=\relax
      \def\end##1{}
      \input #1\relax}}
```

One may wonder why we don't say \let\end=\endinput. It turns out that when we give an \endinput in the middle of a sentence, the rest of the line is still processed. This is very handy when writing macros, because these are considered one line. That way \endinput can be buried very deep in \if's.

The box is to be shipped out on the first occasion, if possible before the first graphic inclusion, otherwise some drivers still will not be able to produce the right files; one never knows in advance how a driver collects and writes down its fonts. The most secure way of doing this is putting the box somewhere on the page in a white color or scaled to zero. Both mechanism can fail, for instance when we use a background, or when scaling to zero is not supported. By including the box contents, the driver will embed the right glyphs, even when they are out of sight.

We use a brute mechanism and make use of the fact that most viewers and drivers clip the page, due to physical constraints. By putting the glyphs meters above the pagebody, we can be quite sure that they never show up, even on A0 paper format.

We can put the box contents somewhere by hand, but an automatic mechanism is more safe, because that way we can take care of unwanted interference. Putting the glyphs in a box at the top of a page (raised \maxdimen) undoubtedly interferes with \topskip, so I soon decided to manipulate one of the TEX primitives that is always used and that could be overloaded without problems. The primitive best suited for this purpose was (of course) \shipout. As always, we save the original meaning:

7
```
\let\normalshipout=\shipout
```

We cannot shipout the box separated from the page, because every \shipout generates a page. The next macros do the job.

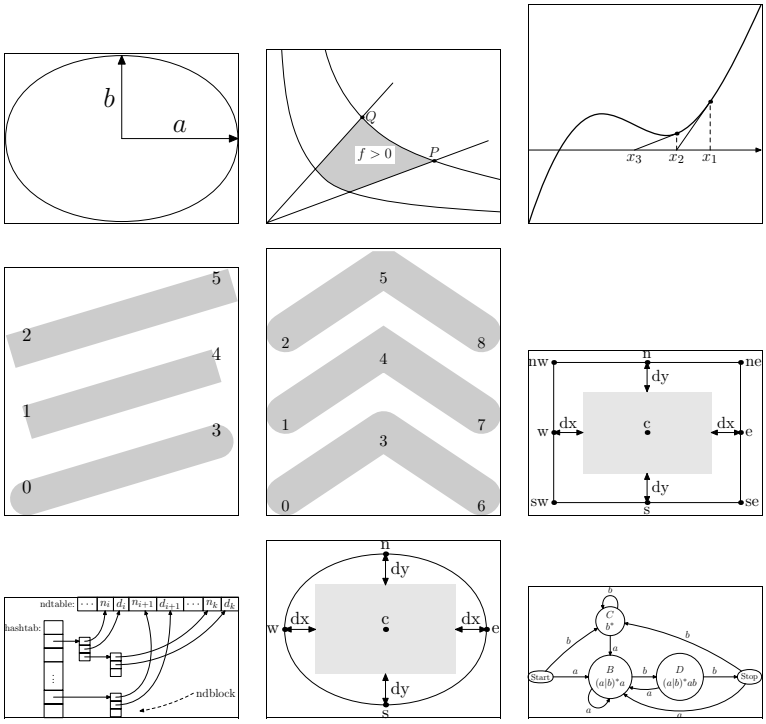
8
```
\def\DumpMetaPostGlyphs
  {\vbox
     {\wd\MetaPostData=\!!zeropoint
      \ht\MetaPostData=\!!zeropoint
      \dp\MetaPostData=\!!zeropoint
      \kern\maxdimen
```

```
      \copy\MetaPostData
      \kern-\maxdimen}}

\def\RestoreShipOut%
  {\global\let\shipout=\normalshipout}

\def\MetaPostShipOut%
  {\dowithnextbox
     {\normalshipout\vbox
        {\DumpMetaPostGlyphs
         \nointerlineskip
         \box\nextbox}
      \RestoreShipOut}}
```

Now let's prove that things work all right and show the example files that are part of the METAPOST distribution:



In this file we preloaded the right fonts by something like:

```
\bgroup
\switchtocorps         [12pt,cmr,rm]  % the 'original' fonts
\UseMetaPostFile       {examples}     % the mp generated file
\UseMetaPostProofFont {cmr10}         % the default prooffont
\egroup
```

The first line switches to the default CONTEXT fonts and is package dependant. In real PLAIN TEX one can omit this line.

\ReUseMetaPo..

When possible, the METAPOST files are to be produced with prologues, which can be accomplished by including the next command in the METAPOST source (the mp file):

```
prologues := 1 ;  % this should be default
```

If after all these precautions things still go wrong, for instance because the driver produced POSTSCRIPT files on a page by page base, one can use:

```
\ReUseMetaPostData
```

After which the 'invisible' box is output at every page. The extra overhead is not that large.

11
```
\def\ReUseMetaPostData%
  {\let\RestoreShipOut=\relax}
```

In most cases the amount of extra overhead is small compared to the rest of the data.

\UseMetaPost..

METAPOST does not use TEX for typesetting the proofings. This means that we have to load the used proof font, which is cmr10 by default, explicitly. The easiest way of doing this is calling an extra file, in which this font is called:

```
\UseMetaPostFile{mp-proof}
```

Such a file looks like:

```
\font\MetaPostProofFont=cmr10

\dostepwiserecurse{48}{127}{1}
  {{\MetaPostProofFont\char\recurselevel\ }}
```

or:

```
\font\MetaPostProofFont=cmr10 0 1 2 3 4 5 6 7 8 9
```

We provide an extra routine for the prooffonts:

```
\UseMetaPostProofFont{cmr10}
```

Because we want this module to be independant of CONTEXT, we use the more plain alternative instead of the more byte saving alternative \dostepwiserecurse.

12
```
\def\UseMetaPostProofFont#1%
  {\UseMetaPostData
     {\font\MetaPostProofFont=#1\relax
      \MetaPostProofFont
      \scratchcounter=32
      \loop
        \char\scratchcounter
        \hskip .5em plus .1em
        \ifnum\scratchcounter<\ifeightbitcharacters255\else126\fi
          \advance\scratchcounter by 1
      \repeat}}
```

Another pitfall lays in the format one uses. One must be sure that both the METAPOST run and the one that generates the document call the same fonts. It's best to use the same format and the same environment (e.g. corps size). Using scalable POSTSCRIPT fonts is less critical.

But, there is another way of doing things! The next solution is derived from the method we use to convert METAPOST code to PDF code.

\UseMetaPost..

After writing module `supp-pdf` I decided to use a bit more advanced way, using the METAPOST created graphic files themselves.

13
```
\def\UseMetaPostGraphic#1%
  {\bgroup
   \message{[MP fonts #1]}%
   %\uncatcodespecials
   \endlinechar=-1
   \setMPspecials
   \obeyMPspecials
   \doprocessfile\scratchread{#1}\handleMPSline
   \egroup}
```

This macro scans the graphics file for the `fshow` operator, that is, lines that start with `(`. If found it interprets the line, which looks like:

```
(string ... string) font size fshow
```

Font definitions specified in the preamble are simply ignored. Only lines starting with `(` are interpreted.

14
```
\def\dohandleMPSline#1#2\relax%
  {\if#1(%
     \expandafter\includeMPcharacters\fileline\relax
   \fi}
```

15
```
\def\handleMPSline%
  {\expandafter\dohandleMPSline\fileline\relax}
```

Before we start scanning for data, we first change some ⟨*catcodes*⟩. The first set of macro's is copied from module `supp-pdf`. This scheme is a bit overdone for this module, but using the same macros saves us some memory.

16
```
\def\octalMPcharacter#1#2#3%
  {\char'#1#2#3\relax}
```

17
```
\bgroup
\catcode`\|=\@@comment
\catcode`\%=\@@active
\catcode`\[=\@@active
\catcode`\]=\@@active
\catcode`\{=\@@active
\catcode`\}=\@@active
\catcode`B=\@@begingroup
\catcode`E=\@@endgroup
\gdef\ignoreMPspecials|
  B\def%BE|
   \def[BE|
   \def]BE|
   \def{BE|
   \def}BEE
\gdef\obeyMPspecials|
  B\def%B\char 37\relax E|
   \def[B\char 91\relax E|
   \def]B\char 93\relax E|
   \def{B\char123\relax E|
```

```
   \def}B\char125\relax EE
\gdef\setMPspecials|
  B\catcode`\%=\@@active
   \catcode`\[=\@@active
   \catcode`\]=\@@active
   \catcode`\{=\@@active
   \catcode`\}=\@@active
   \catcode`\$=\@@letter
   \catcode`\_=\@@letter
   \catcode`\#=\@@letter
   \catcode`\^=\@@letter
   \catcode`\&=\@@letter
   \catcode`\|=\@@letter
   \catcode`\~=\@@letter
   \def\(B\char40\relax     E|
   \def\)B\char41\relax     E|
   \def\\B\char92\relax     E|
   \def\0B\octalMPcharacter0E|
   \def\1B\octalMPcharacter1E|
   \def\2B\octalMPcharacter2E|
   \def\3B\octalMPcharacter3E|
   \def\4B\octalMPcharacter4E|
   \def\5B\octalMPcharacter5E|
   \def\6B\octalMPcharacter6E|
   \def\7B\octalMPcharacter7E|
   \def\8B\octalMPcharacter8E|
   \def\9B\octalMPcharacter9EE
\egroup
```

The lines starting with `(` are interpreted and handled by

18
```
\def\includeMPcharacters(#1) #2 #3 #4\relax%
  {\UseMetaPostData{\font\temp=#2 at #3bp\temp#1}}
```

This method is both robust and reasonable fast. The only disadvantage is that one has to load all graphics. This method is completely macro package independant.

`\ShowMetaPos..`

One may wonder what happens behind the screens. If wanted and needed one can show the texts METAPOST uses by calling:

```
\ShowMetaPostData
```

Because the labels have no height and depth, we use a bit different definition of `\UseMetaPostData`. This time we force a decent linedistance. Because we also want to typeset this data in this text, we also enable linebreaks and correct some spacing. This is how it looks:

- ! ” # $ % & ’ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; ¡ = ¿ ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [“] ^ ˙ ‘ a b c d e f g h i j k l m n o p q r s t u v w x y z – — ″ ~

$a \quad b \quad \begin{matrix} D \\ (a|b)^*ab \end{matrix} f > 0 \quad P \quad Q \quad x_1 \quad x_2 \quad x_3 \quad \cdots \quad n_i \quad d_i \quad n_{i+1} \quad d_{i+1} \quad n_k \quad d_k \quad \vdots \quad \text{ndblock} \quad \begin{matrix} B \\ (a|b)^*a \end{matrix} \quad \begin{matrix} C \\ b^* \end{matrix}$
$b \quad b \quad a \quad a \quad a \quad b \quad b \quad a \quad a \quad b$

The size of the characters corresponds to the size used during the TEX run needed for the METAPOST job. The vertical spacing is not optimal, but suits its purpose.

The less instructive definitions of both macros complete this module.

19
```
\def\UseMetaPostData#1%
  {\global\setbox\MetaPostData=\vbox
     {\everypar{}
      \unvbox\MetaPostData
      \prevdepth\!!zeropoint
      %\baselineskip30pt
      \ignorespaces#1}%
   \global\let\shipout=\MetaPostShipOut}
```

20
```
\def\UseMetaPostFile#1%
  {\UseMetaPostData
     {\def\shipout{ \discretionary{}{}{}}
      \def\end##1{}
      \input #1\relax}}
```

21
```
\def\ShowMetaPostData%
  {\unvbox\MetaPostData
   \vskip15pt}
```

This time I can't prove that things work ok, simply because the right glyphs are already in the file.

```
\UseMetaPostGraphic{mp-exa-1}
\UseMetaPostGraphic{mp-exa-2}
\UseMetaPostGraphic{mp-exa-3}
\UseMetaPostGraphic{mp-exa-4}
\UseMetaPostGraphic{mp-exa-5}
\UseMetaPostGraphic{mp-exa-6}
\UseMetaPostGraphic{mp-exa-7}
\UseMetaPostGraphic{mp-exa-8}
\UseMetaPostGraphic{mp-exa-9}
```

Would have done the job.

So in order to get the right glyphs in the POSTSCRIPT file we can use `\UseMetaPostFile` for loading the TEX file and `\UseMetaPostProofFont` for loading additional fonts:

1. say `prologues:=1` in the METAPOST file
2. (temporary) activate the fonts used in the graphs
3. reuse METAPOST data when needed
4. load the proofing font when used

or if we want the graphics do the job (the prefered way) use instead `\UseMetaPostGraphic`:

1. preload all graphic files
2. reuse METAPOST data when needed

This module will probably be enhanced and/or improved, when I'm past the first experiences with METAPOST. I did consider scanning the POSTSCRIPT file that METAPOST produces. A little bit of scanning and interpreting could do the job quite well, but I wonder if it could be done robust.

22
```
\protect
```