METATEX

Ramón Casares

Abstract

METATEX is a set of `plain` TEX and METAFONT macros that you can use to define both the text and the figures in a single source file. Because METATEX sets up two way communication, from TEX to METAFONT and back from METAFONT to TEX, drawing dimensions can be controlled by TEX and labels can be located by METAFONT. Only standard features of TEX and METAFONT are used, but two runs of TEX and one of METAFONT are needed.

Overview

Together, TEX and METAFONT define the page layout to the pixel. This means that nothing more is needed, not even a means of including figures in a TEX document. To prove this is the aim of this paper.

To split the typesetting process in two parts, one to define and draw the characters and the other to arrange the characters in paragraphs and pages, is surely the best way to reduce the complexity of the typesetting task, provided it needs simplification (see Figure 1). But this method makes it difficult, for example, to integrate labels with graphics in figures, because while TEX is best suited to typeset the labels, METAFONT is the appropriate tool to draw the graphics. And, of course, labels should be located in accordance with the graphics.

Therefore, the true successor of TEX has to include in a single program both the capabilities of TEX and METAFONT. Then the typesetting engine would include a powerful graphic tool, a grid in which to typeset if required, and it could take into account the shapes of the characters to determine, for example, kernings or italic corrections. The other way around is also possible. It could be seen as a graphic engine with a powerful typesetting tool. From this point of view, the page would be a graphic object that could contain paragraphs of different shapes requested from the typesetting tool.

METATEX, although it does not fulfill the requirements of such a successor, can be seen as an early sign of its possibilities. For the moment, METATEX takes advantage of METAFONT's equation solving capabilities to locate objects, including the labels, which are typeset by TEX. The cost of this nice feature is that two TEX passes are required.

During the first TEX pass a METAFONT file is written. As it is TEX itself who writes the METAFONT file, any dimension controlled by TEX can be used and incorporated in METAFONT's calculations. For example, the label sizes, as they will be typeset by TEX, are made known to METAFONT.

After the first TEX pass, METAFONT draws the graphic figures and writes the label locations in its log file. So it is METAFONT's responsibility to locate the labels. Note that, depending on the style of METAFONT programming, this can be completely determined from TEX. In other words, you can relate the label location to the location and size of other METAFONT objects, or not.

When TEX executes its second pass, it takes the graphics from the new font, reads the location of labels from the METAFONT log file, and then everything is complete.

Because labels are just `\hbox`es typeset by TEX, every macro currently defined for text automatically applies also to figures. For example, if a macro `\person` is defined to write its argument in a small caps font and save it to an index file, the same happens whenever it is used inside a figure label.

Methods

METATEX allows the source file to include, in addition to the customary TEX macros to control the text, other commands to generate figures with METAFONT.

Steps. In order to use METATEX the following three steps are to be executed:

1. The METATEX file, suppose it is `filename.ext`, is first processed by `TEX`, with the `plain` format, during which a METAFONT file named `auxiliar.mf` is created. This METAFONT file contains information provided by TEX concerning the size of the labels, so the `MF` program can delete this area from the figure if requested. If the output file `filename.dvi` were typeset now, it would have blanks in place of the figures, but otherwise be the same as the final document.
2. Then `MF`, with the `plain` base, is run on `auxiliar.mf`. As a result, information specifying where to typeset the labels is written in the log file, `auxiliar.log`. In addition, the metric file, `auxiliar.tfm`, and the generic format bitmap font, `auxiliar.329gf`, are created. On my system I have to process this `gf` file to get a `pk` file that my drivers can read, so I execute the program `GFtoPK` on it, getting the packed

bitmap font `auxiliar.329pk`. Please note three points. i) The number `329`, referring to the resolution, varies according to the METAFONT mode. ii) The `tfm` and `pk` files must be in or moved to directories where programs can find them. iii) METATEX sets the METAFONT mode to `localfont`, thus assuming that `localfont` is assigned the appropriate name.

3. Lastly, `filename.ext` is again run through `TEX`. During this second run, both the font `auxiliar` containing the figures and the information explaining where to locate the labels are available, so the document is complete.

The figures fill exactly the same area in both the first and second `TEX` program runs, so indices, tables of contents, and other references that also need two passes to be resolved can take advantage of the two runs needed by METATEX.

Use. To use the METATEX macros, they must be imported by writing in the source file:

```
\input metatex
```

This has to be written after `\mag` has been given its final value. When `metatex.tex` is read, METATEX checks whether the file `auxiliar.mf` exists. If it does not exist, then things are set up for the first pass; for example, `auxiliar.mf` is opened for writing. If it does exist, then things are set for the second pass; for example, `auxiliar.log` is opened for reading. This means that if `auxiliar.mf` is not deleted, then step 3, the second `TEX` program pass, is executed directly. This saves time when only the text in file `filename.ext`, but not the figures, were modified.

User macros. The METATEX user macros are:

- `\MTbeginchar(wd,ht,dp);` states that a figure sized as given (width `wd`, height `ht`, depth `dp`) will be created. These values should be known both by TEX and by METAFONT, so for example `12pt`, `6cm`, `\the\hsize` or `\the\dimen0`, always without `#`, are allowed. During the *first pass*, TEX writes in `auxiliar.mf` the METAFONT macro `beginchar` assigning character codes sequentially, and box `\MTbox` is made empty but sized as specified by the arguments of this macro. During the *second pass*, TEX puts the corresponding character of the font `auxiliar` in box `\MTbox`. The size of `\MTbox` is that specified and not affected by the character dimensions.
- `\MTendchar;` finishes the figure definition. During the *first pass*, TEX writes the METAFONT macro `endchar;` in file `auxiliar.mf`. During the *second pass*, box `\MTbox` contains the complete figure, including labels. Something like `\box\MTbox` is used to typeset the figure.
- `\MTlabel*(s)cc"Text";` adds a label to the current figure. The parameter between quotes, `Text` in the example, is the label content; it will be put inside an `\hbox` and therefore could be anything that TEX allows inside an `\hbox`. The optional asterisk after `\MTlabel` instructs METATEX to erase the area of the figure already drawn that it is under the label.

  The label will be located at METAFONT point `z.s`, where `s` is the parameter between parentheses. The reference point is further specified by the optional parameter after the right parenthesis, `cc` in the example. This parameter is composed of exactly two letters: the first can be `t` meaning top, `c` meaning center or `b` meaning bottom; and the second letter can be `l` meaning left, `c` meaning center or `r` meaning right. So, for example, `tl` means that the label reference point is its top left corner. The default value for the reference point is `cc`, that is, its center.

  `\MTlabel` should only be used between `\MTbeginchar` and `\MTendchar`. During the *first pass*, it writes the following three elements in `auxiliar.mf`: i) the METAFONT macros which in turn cause `MF` to write the label reference point location to its log file, `auxiliar.log`; ii) the four label sides, which are by this means made available to the following METAFONT code for the figure, notated as `y.s.t` for the top side, `y.s.b` for the bottom side, `x.s.l` for the left side and `x.s.r` for the right side; and iii) the code to delete, if requested, the figure area already drawn that is under the rectangle occupied by the label. During the *second pass*, it adds the label to the box `\MTbox` in the place that reads from file `auxiliar.log`, making no modification to the dimensions of `\MTbox`, even if the label is typeset outside the box.

There are three more macros for passing information to METAFONT, that is, for writing general text in `auxiliar.mf`: `\MT:`, `\MTcode` and `\MTline`. This happens only during the first pass; during the second pass, these macros do nothing.

- `\MT:` writes in file `auxiliar.mf` everything till the end of line. It writes verbatim except for the character `\`, which keeps its normal TEX `\catcode` of `0`. Spaces are *not* ignored after

macros. The sequence \\ writes a single \ in file auxiliar.mf.

- \MTcode writes in file auxiliar.mf everything until it finds a line equal (including \catcodes) to the current value of \MTendmark. By default, this is a blank line, thus, \def\MTendmark{}. As with \MT:, it writes verbatim except for \, which still operates as an escape character. The control sequence \\ writes a single \ in file auxiliar.mf.
- \MTline{text} writes its parameter to auxiliar.mf, text in the example. It does not change the \catcodes in the argument, so it does not perform verbatim writing. But all plain special characters can be written prefixing them by the escape character \. The plain special characters are (not including the first colon nor the final period): \{}$&#^_~%. For example, \# results in #.

When defining TEX macros that write to auxiliar.mf, \MTline should generally be used in preference to \MT: or \MTcode, because the latter two use the end of line in a special way that is not usually available when TEX is reading a macro.

TEX dimensions can be included using any of these three writing macros. For example, \the\hsize will be expanded to 225.0pt (for the present article), and written as such to the METAFONT file auxiliar.mf. Note that the character \ keeps its escape \catcode in all three writing macros. In the case of \MTline, braces {} also keep their \catcodes and therefore macros with parameters can be used normally.

Examples

Diagram. First a typical example of METATEX usage, showing the file formats, programs, and their relationships. The figure width is exactly \hsize, but what is more important is that the same code will adapt itself to any value for the measure. Well, of course, not to *any* width but to any width between, let's say, 8 cm and 25 cm.

Figure 1 is the one column version, and Figure 2 (above the appendix) is the two column version, generated by the same source.

Shadowing. Both TEX and METAFONT are ill suited to creating shadows. In TEX, one straightforward technique is double use of \leaders, but in practice this results in huge dvi files. In METAFONT, drawing lots of tiny points easily exceeds the capacity of the program. The solution is to coordinate the work of both programs.

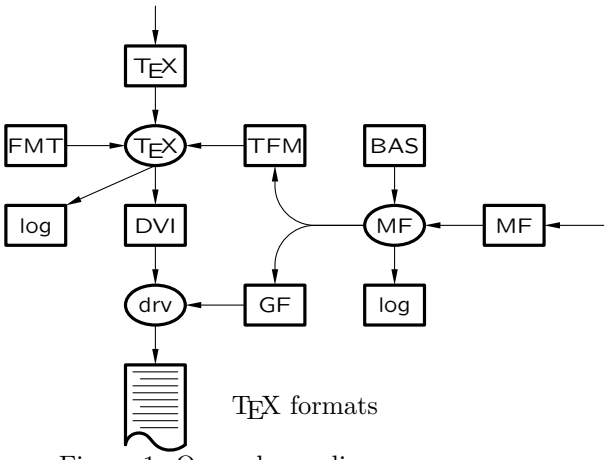


Figure 1: One column diagram

To create a large rectangular shadow we divide it into an array of $n \times m$ smaller rectangles. The smaller rectangles are all identical, so it is enough for METAFONT to draw one shadow character and then for TEX to typeset a solid area repeating it.

To simplify the tasks of both TEX and METAFONT, the size of the shadow character should be similar to that of normal characters, because neither program was designed to work well with extraordinarily large (or small) characters. So a good approach is to make the shadow character as big as possible but never wider nor higher than 16 pt.

For METATEX, each figure is a character. This causes problems with METAFONT when the figure is big and the resolution is high, because it cannot draw areas bigger than 4095×4095 pixels. This is not usually a problem working at 300 dpi. (It is never a problem with METAPOST, see the following section on PostScript. Another advantage of using PostScript is that you get a shadow simply by drawing a grey rule, and none of the above machinations are necessary.)

Keys. After the following METATEX macros:

```
1. \MTcode
2. def keybox =
3.  pickup pencircle scaled 0.8pt;
4.  x1 = x3 = 1pt;
5.  x2 = x4 = w - 1pt;
6.  x5 = 0; x6 = w;
7.  y1 = y2 = -d;
8.  y3 = y4 = h;
9.  y5 = y6 = (h - d)/2;
10.  draw z1 -- z2 .. z6{up} ..
11.     z4 -- z3 .. z5{down} .. cycle;
12.  z0 = (x1,0);
13. enddef;
```

```
14.
15. \def\defkey#1#2{\setbox0=\hbox{\sf#2}%
16.  \dimen0=\wd0\advance\dimen0 by 2pt
17.  \dimen2=\ht0\advance\dimen2 by 1pt
18.  \dimen4=\dp0\advance\dimen4 by 1pt
19.  \MTbeginchar(\the\dimen0,%
20.                \the\dimen2,%
21.                \the\dimen4);%
22.   \MTline{keybox;}%
23.   \MTlabel(0)bl"\sf #2";%
24.  \MTendchar;%
25.  \expandafter\newbox
26.   \csname\string#1box\endcsname
27.  \expandafter\setbox
28.   \csname\string#1box\endcsname
29.   =\vtop{\unvbox\MTbox}%
30.  \def#1{\expandafter\copy
31.   \csname\string#1box\endcsname}}
32.
33. \def\makekey#1{\expandafter\defkey%
34.  \csname#1\endcsname{#1}}
```

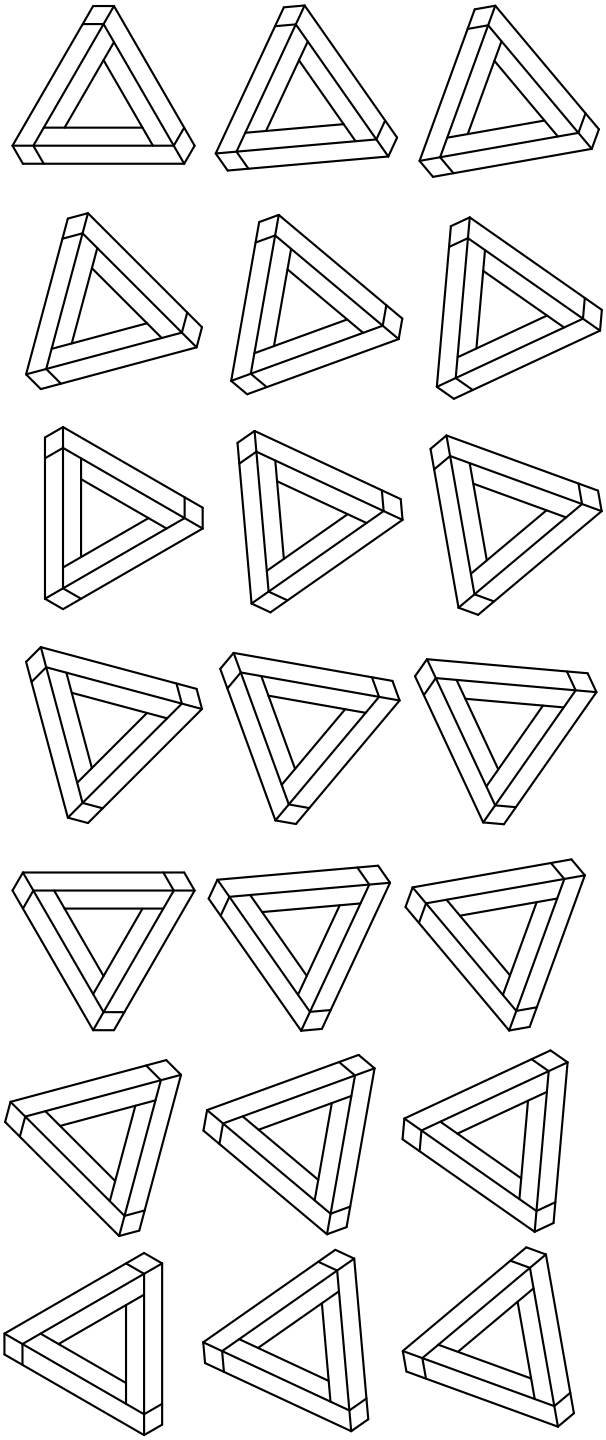
Then, we can declare `\makekey{Alt}` to typeset Alt simply via `\Alt`. It is also possible to declare `\defkey\escape{\tt\char92}` and then `\escape` results in \.

Baroque tables. Baroque periods are the result of new technical achievements providing unexplored possibilities and hence the urgent need to experiment with them, frequently far away from what discretion might recommend. This explains the time of baroque software that we live in, and increases the value of METATEX, because it provides the means to easily draw baroque tables. I am not a baroque man, so my baroque table example is not baroque but, and this is the point, it is at least not built with straight lines.

This is not a straight table but it's only an example and therefore not so ample of what's METATEX-able!

Ornate paragraphs. Only if you are truly baroque can you get the most from METATEX. If, for example, you like ornate paragraphs, you are in your element. Just put the material in a `\vbox` to get the height and the depth, and pass these dimensions to METAFONT to draw a right sized embellishment. Ah!, but be aware that Computer Modern is a neoclassical font, so it won't mix well with your elaborations.

Exercise.* Take three equal bars and build as shown. Ask Roger Penrose if you don't find the solution.



* The real exercise is to fill the rest of the page with tribars. A clue: let TEX to calculate how many are needed, so you can concentrate your efforts in drawing the figure.

PostScript

By taking advantage of METAPOST, we can make PostScript versions of the METATEX files. Simply execute `mpost &plain auxiliar.mf` instead of METAFONT and, after TEX's second pass, execute `dvips` (METATEX uses `dvips` specials). This works because `auxiliar.mf` is valid code for both METAFONT and METAPOST, and because, in the second TEX pass, METATEX checks which one was used and adapts itself to the situation.

And thanks to PDFTEX and the ConTEXt files `supp-mis.tex` and `supp-pdf.tex`, it is also possible to get PDF output. Just execute PDFTEX twice, instead of TEX, and once METAPOST, instead of METAFONT. (In practice, METATEX uses its own macro file `mtmp2pdf.tex` for this, instead of the ConTEXt files. I extracted all that METATEX needs from the ConTEXt files into `mtmp2pdf.tex`.)

This works because, if METATEX determines that METAPOST was employed to draw the figures, it then checks which program, TEX or PDFTEX, is executing. If TEX, then it includes the files produced by METAPOST using the `dvips` specials. If PDFTEX, it translates from `ps` to `pdf`.

Therefore, the same METATEX source file generates at will any of the three output formats—`dvi`, `ps`, or `pdf`—just by running the appropriate programs. In the appendix, as an example, there is a DOS batch file that shows how to get the `pdf` version of a file `filename.ext`.

Other graphical tools

There are other tools to include pictures in a TEX document. LATEX's `picture` environment, PICTEX, and `mfpic`, are three of them. METATEX is similar to `mfpic`, in that both use METAFONT to draw.

The aim of `mfpic` is to overcome the difficulties of LATEX's `picture` environment and of PICTEX. LATEX's `picture` environment uses four pre-cooked special fonts, and its drawings are just compositions of these characters (as well as TEX's builtin `\hrule`s and `\vrule`s). PICTEX only uses a tiny point to compose the pictures, so it is more general. But letting TEX to draw the figures setting point after point is painful, as noted above. The `mfpic` solution uses a better tool for drawing: METAFONT.

With these origins, for `mfpic`, METAFONT is a hidden back-end processor, and `mfpic` imposes only two requirements on its users: to know TEX, and to know `mfpic`. On the other hand, METATEX's approach is minimalist, at the cost of being more demanding with its users. A METATEX user has to know TEX, METAFONT, and METATEX — although this last requirement is small, because METATEX only builds the necessary bridges to use TEX and METAFONT in a cooperative way.

A feature that shows the different strategies employed in designing `mfpic` and METATEX is label positioning. METATEX labels are located by METAFONT, so TEX has to read the METAFONT log file to learn where to typeset them. For `mfpic`, TEX itself locates the labels, but by doing so `mfpic` has to give up some nice METAFONT characteristics, such as its equation solving capabilities.

In summary, `mfpic`'s aim is to draw pictures in TEX documents in a better way than using LATEX's `picture` environment or PICTEX; while METATEX's intention is to coordinate the work of TEX and METAFONT. In this way, METATEX provides the full raw power of TEX and METAFONT, and it is up to you to harness them.

Final remarks

I have been using METATEX for some years. The first version was dated 1994, but it has been used only for personal purposes. For this reason, it is not truly a straightforward end-user tool, as for example LATEX packages should be. It has to be used knowledgeably and with care. And though most tasks can be automated, by chaining TEX and METAFONT errors are even more difficult to pinpoint than in TEX or METAFONT alone.

Nevertheless, METATEX serves to validate the feasibility of a closer collaboration between TEX and METAFONT and to appraise the interest of such a collaboration. And, of course, if you dare, you can get lots of fun, and at least an equal amount of frustration, using METATEX. Try it!

The METATEX package is available from CTAN in CTAN:`/macros/plain/contrib/metatex`.

Happy METATEXing!

◇ Ramón Casares
Telefónica de España
`r.casares@computer.org`

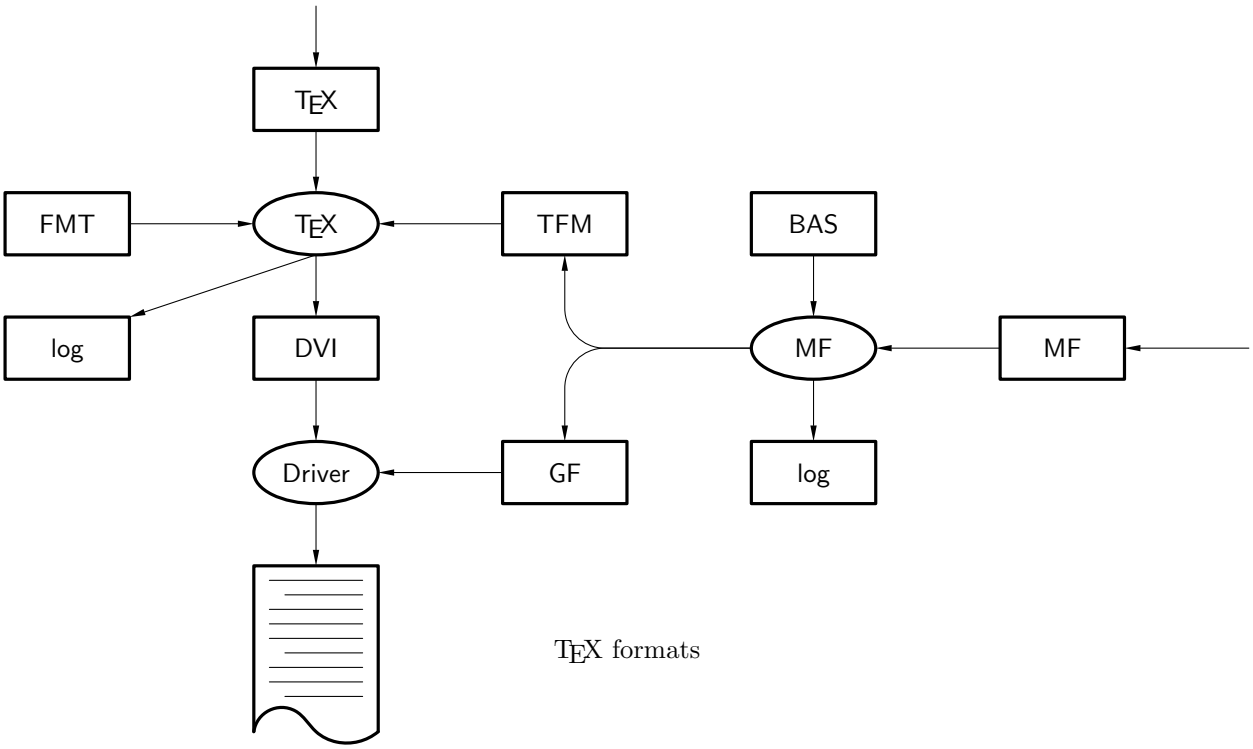


Figure 2: Two-column diagram

Appendix: Pseudo-batch example

This is an example based on DOS batch files that can be adapted for other operating systems. It processes the file `filename.ext`, generating `filename.pdf`. We will comment each line of pseudo-code.

1. We go to the directory where our working files are.
   ```
   cd c:\dir\subdir\mydir
   ```
2. We set environment variables (if necessary). In this case we are using a `web2c` system, so it is enough to set one. In `web2c`, by default, all programs look for files in the current dir, "`.`", and that is enough for us.
   ```
   set TEXMFCNF=c:\tex\texmf.local\web2c;c:\tex\texmf\web2c;d:\texmf\web2c
   ```
3. We tell the operating system where to find the programs.
   ```
   path=$path$;c:\tex\bin\dos
   ```
4. After the settings, we force the first TEX pass.
   ```
   if exist auxiliar.mf del auxiliar.mf
   ```
5. Then, we execute the first TEX pass (in this case, it is PDFTEX).
   ```
   pdftex &plain filename.ext
   ```
6. If METATEX was not used, and no `auxiliar.mf` was written, then we are done.
   ```
   if not exist auxiliar.mf goto end
   ```
7. Otherwise we run METAPOST with its `&plain` memory (format), also known as `&mpost`.
   ```
   mpost &plain auxiliar.mf
   ```
8. Finally, we execute the second TEX pass.
   ```
   pdftex &plain filename.ext
   ```
9. We now have the complete `filename.pdf` file.
   ```
   :end
   ```