Public-Domain, Documented Implementations of TEX and METAFONT for VAX/VMS

Brian Hamilton Kelly
Royal Military College of Science
Shrivenham
Janet: tex@uk.ac.cranfield.rmcs

Adrian F. Clark
University of Essex
Colchester
Janet: alien@uk.ac.essex

Abstract

In this paper, a VMS implementation of the most recent versions of TEX, METAFONT and all the important support utilities are described. Some of the important features include TEX sensitive VMS editors, command-line interfaces, and complete documentation, both HELP and printed manuals.

Two public-domain implementations of TEX on VAX/VMS have been in use for some time: the first was by David Fuchs of Stanford University and the second [*TUGboat*, V10], derived from Fuchs', was by one of the authors of this paper. The latter provided an interface to two of the most popular editors on VMS (TPU and EDT) and an expanded memory. However, most of the additions were written in Fortran rather than Pascal. In this paper, we present a compatible but greatly enhanced VMS implementation of TEX 3.0. Equivalent changes to the latest version of METAFONT are also summarised. This implementation was carried out by the first author, again derived from Fuchs' earlier work, with occasional assistance from Niel Kempson, also of RMCS; the second author merely hassled for particular features to be included.

Editing after Errors

The most significant enhancement again concerns the editor interface. When TEX encounters an error in an interactive run, typing "e" (or "E") to its prompt invokes an editor. The editor which is invoked may be one of the standard set of VMS editors (TPU, EDT, LSE, TECO), or an unsupported editor such as SOS. The one to be used is specified via the logical name TEX$EDIT; this is analogous to other VMS utilities such as MAIL. (If this logical has not been defined, or expands to something that this implementation can't handle, the default TEX response ensues, the user being told whereabouts to edit his/her file.) The four standard editors are specified by setting TEX$EDIT to one of CALLABLE_TPU, CALLABLE_EDT, CALLABLE_LSE or CALLABLE_TECO, so the desired editor could be defined *e.g.*

```
$ DEFINE TEX$EDIT CALLABLE_TPU
```

As these names suggest, the appropriate editor is dynamically linked to TEX and then invoked. Wherever possible, the cursor is moved to the point in the text at which TEX detected the error.

- For TPU and LSEdit, the cursor *is* always positioned correctly.
- In the case of TECO, the user is provided with a macro (M1) which moves to the correct place when it is executed.
- Since EDT does not really support the ability to position the cursor from outside its "change" mode, this is done by sleight-of-hand. The method adopted *does* position the cursor, but results in EDT generating an error message on entry; a message which can safely be ignored.

The alert reader might ask "What about users' initialisation files?". In the case of LSEdit and TPU, the editor interface need only invoke the editor and pass in the desired cursor position with the /START_POSITION qualifier, so any initialization file defined through the relevant logical will still be included. With EDT, the /COMMAND qualifier is used to pass the name of the file which positions the cursor correctly; this will be written to TEX$EDTINI.EDT; thus it may be written elsewhere if TEX$EDTINI is

defined as a suitable logical name. If the logical name EDTINI is defined, then the initialization file arranges to execute the commands in the file indicated by this logical after the cursor has been positioned. With TECO, things get more complicated: according to the documentation, the callable version of TECO is supposed to accept a parameter defining the initialization file; however, this doesn't work. Many hours spent poring over a trace of the execution of TECOSHR revealed that the code to interpret this third parameter is bypassed. Therefore, TEX redefines the logical TEC$INIT to reference its initialization file (which is written to TEX$TECOINI.EDT). Any user's initialization previously defined in this logical will first be included in the new file, before the cursor positioning macro is defined into register Q1. After editing is completed, the original definition of TEC$INIT is restored, if there was one.

Any other translation of TEX$EDIT is considered to be a DCL command to be executed in a subprocess. (This may, of course, be a command procedure, in which case the equivalence string will commence with an "@ character.) Three arguments are passed to the DCL command: the name of the file to edit, and the line and column at which the error was detected. (These latter numbers are one-based, *i.e.* the first character of the file is at line 1, column 1.) TEX examines the status returned by this DCL command or command procedure. Any error status causes TEX to revert to its default behaviour, exiting after telling the user where the file should be edited.

After the error has been corrected, the editor may be exited, creating a new version of the input file. At this point, control returns to TEX, so that the user may fix the problem within TEX and continue with the run if he or she so wishes. This feature permits several errors to be corrected in a single TEX execution; however, it is important to realise that TEX continues to process the *original* (*i.e.,* incorrect) file. However, further invocations of the editor will read in the *latest* version of the source file. (In the case of LSEdit and TPU, this implementation is able to recognize that the user has quit from the editor without writing a new file, and under these circumstances the original file will continue to be read until a new version is written.) It is not possible to make this discrimination with other editors, although a warning status returned by an editor run in a sub-process will be interpreted in this same manner. The dvi and log files continue to be written in the usual fashion.

Some criticism has been directed at the authors for continuing to process the TEX source after performing an edit. It has been suggested that the description of the "E" option in *The TEXbook* [p.32] implies that TEX *must* exit after performing the edit, just like typing "X." However, experience has shown the usefulness of being able to perform further edits (not the least of which is that LATEX's auxiliary files get completed correctly). The user can always type Control-Y to abort TEX if he/she considers that the error is too severe to permit sensible recovery after editing. All files created by TEX itself in the current run (dvi, lis, aux, etc.), will then be discarded, although the new source file(s) written by the editor will be retained.

Diagnostic Feedback whilst Editing

Another feature of this implementation, relating to the language-sensitive editor, LSE, is unconnected with the use of that editor from TEX's error prompt. However, it *does* prove extremely useful when used the other way round, *i.e.* when TEX is invoked from *within* the language-sensitive editor by use of the latter's COMPILE command. The definition of "language" LATEX for LSE includes the following:

```
/CAPABILITIES=DIAGNOSTICS -
/COMPILE_COMMAND= -
   "LATEX /BATCH 'LSE$FILE'" -
```

(Similar definitions could be provided for the "languages" TEX and SLITEX.) As TEX runs, in batch mode, all error messages are written in TEX's normal format to the log file, and are written additionally in the diagnostics file in a format defined by DEC. After processing has been completed, control is returned to LSEdit and the latter's REVIEW command will then read in the diagnostics file. By using the NEXT STEP and PREVIOUS STEP commands, each error report in the REVIEW window may be highlighted; the GOTO SOURCE command will then position the cursor at that point in the relevant source file at which the error was detected. (It is *not* recommended that LSEdit be invoked from within TEX when generating a diagnostics file, because the latter will not be available to LSEdit until TEX completes its processing.)

Command-Line Interface

Another major improvement offered by the new implementation is that it provides a proper command-line interface to DCL via a .CLD (command definition) file. The DCL commands defined in this file may be made available to the user's process by saying:

```
$ SET COMMAND TEX
```

This is usually done in the command procedure which establishes logical names and DCL symbols needed by TEX. However, this DCL operation is quite slow, so it is better performed (once only) by the system manager and installed in the standard SYS$SHARE:DCLTABLES.EXE file.

The DCL command TEX supports the following qualifiers:

/BATCH which tells TEX to start running in batch mode

/BIG to use the big-memory version of TEX

/DIAGNOSTICS=*filename* for specifying the name of the LSE diagnostics file. If no file specification is provided, TEX will use the name of the first file input, with an extension of .DIA

/DVI_FILE=*filename* for overriding the default name of the resulting dvi file. If this qualifier is not supplied, the name of the dvi file is the same as that of the first file input, with an extension of .DVI

This qualifier may be negated (/NODVI_FILE) and will then suppress generation of the dvi file. This can be useful when making the first one or two passes over LATEX source files to generate all cross-references and BIBTEX citations. The log file will report that no dvi file was generated, but it will also say how large it would have been.

/FORMAT=*filename* to specify the name of the format file; this defaults to

TEX$FORMATS:PLAIN.FMT

Other DCL verbs can be defined such that LATEX invokes the image of TEX with /FORMAT=TEX$FORMATS:LPLAIN.FMT as default, etc.

/INITEX to invoke iniTEX rather than TEX; iniTEX is used to build format (.FMT) files. As an amusing aside, when this command definition file was first written, it defined INITEX as a verb. Later that night the operators discovered that attempting to INITIALIZE a magnetic tape for backup actually invoked iniTEX: DCL only looks at the first four characters of a command verb!

/LOG_FILE=*filename* to override the default name of TEX's transaction log-file. If this qualifier is not supplied, the name of the log-file is based on that of the first file input, but with an extension of .LIS.

This qualifier may be negated (/NOLOG_FILE) to suppress generation of a log file.

/TRIP invokes a version of TEX which has arrays of the correct sizes for the TRIP test.

The TEX command also accepts a single parameter, which is normally the name of the file to be processed. Its default extension is, of course, .TEX. However, the entire command line is used for this parameter, and is folded to lowercase (DCL having "kindly" converted it to uppercase) before TEX sees it, so that one can include most TEX commands on the DCL command line. (They must, of course, be commands which contain no upper-case letters). Recent development work on METAFONT revealed that it was easier to require that strings containing multiple commands, with embedded spaces, should be entered as a standard DCL quoted string, replacing the usual single parameter of a file name. Since this quoting will preserve lowercase letters correctly, a future update of this TEX implementation will remove the folding to lowercase; this new version *will* permit the insertion of uppercase letters where desired in command lines.

If any input file cannot be found in the current directory, TEX looks in the directory (or all of the directories in a comma-separated list) specified by the logical name TEX$INPUTS. Similarly, TEX uses the logical name TEX$FONTS to specify the directory or directories in which to look for font (.TFM) files.

TEX returns a status to VMS as it exits. This will be one of STS$K_SUCCESS, STS$K_WARNING, STS$K_ERROR or STS$K_FATAL, depending on whether TEX detected errors and how they were handled. Although this status can be tested in the usual way, TEX inhibits VMS from outputting the corresponding error message.

Changing change files. As the changes are supplied, there is a master change file, TEX.CH and subsidiary change files, such as TEX-INITEX.CH for building iniTEX, and TEX-BIGTEX.CH to enlarge the size of many of TEX's internal tables. There is an accompanying program, WMERGE.C, which can be used to merge the change file and subsequent modifications to it.

This WEBmerge program must originally have been written in WEB, since the original C source contains references to wmerge.web, but there is no indication of the author. It was extended (and debugged) by BHK, who added facilities such that it could match WEB's various @x,@y and @z commands within change sections themselves. By applying these subsidiary change files to the master file, change files are created for generating the different variants from the one master TEX.WEB.

The reason for distributing the big TEX changes separately is that the version with smaller memory

requirements runs a little faster on smaller VAXen (approximately 16% faster on a VAX-11/750).

At present, no provision has been made for applying the changes to generate TEX-XET, but it is under consideration.

METAFONT

A similar implementation of METAFONT has also been performed. This supports an essentially identical editing interface, but controlled by the logical name `MF$EDIT`. It also has a similar command-line interface, supporting the following qualifiers:

`/BASE=`*filename* specifies the name of the base (analogous to TEX's preloaded format) file to be loaded. This defaults to the file `MF$BASES:PLAIN.BASE`. Again, another verb can be defined to run with CMplain by default.

`/BATCH` indicates that the run is to take place in batch mode.

`/DIAGNOSTICS=`*filename* specifies the diagnostic file for use with LSE.

`/GF_FILE=`*filename* specifies the name of the file to receive the *generic font* file, which can be further processed to a `pk` or `pxl` file. This defaults to the name of the first file read in, but with an extension of .⟨*mag*⟩`GF`, where ⟨*mag*⟩ reflects the pixels/inch of the file; the default will thus be `.2602GF` if `mode=proof`.

`/NOGF_FILE` suppresses output of the `gf` file.

`/INIMF` to invoke iniMETAFONT, which can build base (`.BASE`) files

`/LOG_FILE=`*filename* to specify the name of the file to be used for METAFONT's log-file. This defaults to the name of the first file read in, but with an extension of `.LIS`

`/NOLOG_FILE` suppresses output of the log file.

`/TRAP` invokes a version of METAFONT which has arrays of the correct sizes for the TRAP test

Analogously with TEX, METAFONT uses the default extension `.MF` on input files. The current directory is scanned for an input file. If it is not found there, the comma-separated list of directories specified by the logical name `MF$INPUTS` is searched.

Graphics support. There is one major difference between TEX and METAFONT. When designing fonts, some means of interactively viewing the glyphs is invaluable. This implementation allows previewing on several graphics terminals; the device to be used is selected by assigning a mnemonic to the logical name `MF$TERM` before invoking METAFONT. If this logical is not defined, or its definition doesn't correspond with one of the recognized values, then graphics output is suppressed. The recognized mnemonics are (note that these *are* case-sensitive):

`go140` GraphOn 140 terminal

`gp` Northwest Digital Technology's *Graphics Plus* terminal

`tek` Tektronix 4105 (or compatible) terminal

`Vis` Visual Technology's *Visual 550* terminal

Future Directions

Both these implementations have been in use for a reasonable amount of time at a number of sites and no problems have been reported. They are available from the UK TEX archive at Aston, both as change files and as object modules. Don Hosek has recently been extending the change files so as to permit the use of different logical names (some people believe that they should obey DEC's stricture that logical names containing dollar signs are private to DEC themselves). He has also made a major contribution by removing graphics support into separate shareable images (one for each terminal type supported) which means that a new graphics display may be supported without the necessity of revising the METAFONT change file and recompilation. His extensions to this implementation will then become the default for the VMS-world.

Bibliography

Clark, Adrian. "An enhanced TEX-editor interface for VMS," *TUGboat* 10(1), April 1989, pp.14–15.

Knuth, Donald E. *The TEXbook*, Addison-Wesley, 1986.