Brief Functional Characterizations of the Procedures in the TEX/PASCAL Compilation Unit, SYSDEP

by

C. L. Lawson
I. Zabala
M. Díaz

Stanford University, January 27, 1981*

*This document is an enlarged and revised edition of a paper with the same name, published by C. L. Lawson; Computing Memorandum No. 166, September 10, 1980, Jet Propulsion Laboratory, California Institute of Technology.

Contents

Brief Functional Characterizations of the Procedures in the TEX/PASCAL Compilation Unit, SYSDEP

Introduction

In working on a Univac computer system with the PASCAL code for TEX, TEXPRE, and SYSDEP, I have felt a need for some types of documentation other than the documentation provided by the Stanford group. I have had telephone discussions with members of the Stanford group on this topic. The purpose of this memo is to provide an specific example of one type of document that I feel would be a very useful auxiliary document for anyone undertaking to install TEX/PASCAL on a different computer system.

I have used the following guidelines in composing this document.

A. Group the procedures into functionally related classes.

B. For each group describe the general functional area covered by the group, and any data structures, limited ranges of parameter values, etc., common to the whole group.

C. Within each group list the procedure names in alphabetic order with a brief functional characterization of each procedure. Format this on the page so the procedure names stand out prominently from the associated text.

D. Use only terminology that is well-defined in the PASCAL model of computer programming. Any other terminology must be defined in terms of PASCAL or English language primitives. Examples of apparently DEC-10 related terms to be eschewed include "WAITS system", "file extension", "file directory", "directory name", and "system directories".

E. Include a table of contents so the reader can see the major groupings of procedures at a glance.[1]

§1 Handling Printable Characters and Strings

The array STRINGPOOL holds strings of printable characters of two types. There is a printable string associated with each ASCII ordinal in the range [0...127], and there is also a miscellaneous set of printable strings for use as errror messages, etc. Pointers into the array STRINGPOOL are held in the array STRNG.

APPNDREAL

Appends a real number to a string (both things are arguments of the function). The rules for the string and for the returned value are the same as those in APPNDSTRING (below). Before appending, the real number is converted into a string of five characters that has a decimal point in the fourth position. Since it will always be positive, there's no need for a sign. For example, the string obtained from 3/2 is precisely "<space><space>1.5".

[1] See also, by the same author, *Detailed Specification of Procedures in the TEX/PASCAL Compilation Unit SYSDEP*, J. P. L. Section 366, Memo No. 467.

APPNDSTRING

This procedure appends string one string to another, fetching the former from either FILENAME or from STRINGPOOL.

In any case, the function returns an identifier for the string as required by PRODUCESTRING.

INITSTRINGS

Reads printable strings for the ASCII ordinals [0...127] from the external file "ASCII TBL", and additional printable strings from the external file "STRINITBL". These are all stored into the array STRINGPOOL, indexed from the array STRNG.

Reading is done using ICHAN1 which is reset for each of these two external files.

PRINT, PRINTINT, PRINTLN, PRINTOCTAL, and PRINTREAL

These procedures build print images from the strings in STRINGPOOL and their arguments, and output them to the terminal as well as to the error file. See Section 5 for the specific function of each of these five procedures.

PRODUCESTRING

Fetches a string from either STRINGPOOL or FILENAME, and returns it to the calling proceddure.

§2 Handling External File Names

This set of procedures keeps track of external file names and their associations with internal file names. These procedures are very strongly oriented toward a DEC-10 system. It appears that some of this functionality would be handled quite differently on other systems.

External file names are held in the array FILENAME. Indexes f in the FILENAME array are usually passed as FILNAM(f) to procedures dealing with strings, to help them find out whether the argument should be located in the FILENAME array or in STRNGPOOL.

INITFILENAME

Initializes the data structures that will be employed to parse an external file name. It is called once for each external file name. Its effect depends on whether the given file is an input file, an output file, or one of the font information files.

Refer to the description of function SCANFILENAME below.

APPENDTONAME

Refer to the description of function SCANFILENAME below.

PRINTFILENAME

Outputs a file name from the array FILENAME to the terminal and the error file.

PRODUCESTRING

Fetches a string from either FILENAME or STRINGPOOL, and returns it to the calling procedure.

SCANFILENAME

The function SCANFILENAME scans the input for the name of an external file and tries to open it. Its argument says whether it is a font file or send stream for which space has been reserved in the data structures or an input file that has not been allocated yet. This function belongs in the main TEX module, but because file names are very system dependent, the parsing is done in the system dependent module as follows:

First INITFILENAME is called with the same argument that SCANFILENAME received from its caller. INITFILENAME should then take care of any initialization needed for the parsing of the file name. Using its argument, it can decide whether the parsed name will refer to a font information file, to a "send" stream or to an ordinary input file. The negative of the argument is returned if there are too many open files (this can only happen with input files).

Next APPENDTONAME is called once for each token in the input. Using the data structures set up by INITFILENAME, APPENDTONAME decides everytime whether the part of the name received so far is syntactically correct for such a file name (otherwise it returns "malformedname") and if so, whether it has already received a complete file name (otherwise it returns "needmore"). In case the whole file name was received, APPENDTONAME tries to open that file. If it succeeds, it will return done, otherwise it returns "failure". In any case, APPENDTONAME saves the scanned string in a place from where it can be retrieved for use in error messages.

In the Stanford version of SYSDEP, some defaults are set for certain portions of certain file names. note APPENDTONAME accepts everything until it finds a token that could never be part of a file name. If it is a delimiter (space, period, etc.) it considers that the whole file name was received and tries to open it. Otherwise it returns "malformedname". Upon an unsuccessful return from APPENDTONAME, SCANFILENAME returns a value that allows its calling routine to recognize the failure and locate the stored name if it has to be printed in some error message.

§3 Operations on Input Files ICHANx

These procedures do various operations on the six input files ICHAN1 through ICHAN6. The file is selected by an integer parameter in the range [1...6].

External file names associated with these internal file names are stored in the array FILENAME with index values [MAXFNT+1+10+1...MAXFNT+1+10+6]. This is the last part of the FILENAME array, and operates in a LIFO manner. The name of the last \input command is stored in the first free entry in *that* range. When the end-of-file is reached, that entry is liberated.

NOTE: SYSDEP uses ICHANx with x in the range [1...6] because of the existence of a compiler imposed limit on the number of files that a PASCAL program can keep open. This is clearly a system dependent parameter. Six input files may not be enough for some complicated TEX sources.

CHANPTR

Returns the ordinal value of the current input character in INCHANx; namely, it executes CHANPTR:= ORD(ICHANx↑).

EOFCHAN

Returns the value of EOF(ICHANx).

GETCHAN

Executes GET(ICHANx).

GETFIRSTLINE

Skips system header stuff, if any, at the beginning of a file to get a position for reading first meaningful information. This is highly system dependent.

INLN

Reads one line from ICHANx.

RELEASE

Releases the input file ICHANx by executing RESET(ICHANx) followed by FILPTR:=FILPTR-1. The latter statement frees the top entry in FILENAME, the top of the stack of input files.

RSETFILE

Opens ICHANx for input using RESET with a nonstandard parameter list.

This procedure is completely analogous to the PDP-10 PASCAL `reset` procedure. Its arguments FNAME, FDIRECTORY, and FDEVICE are required by it precisely in the given form.

§4 Operations on Output Files OCHANx

These procedures do various operations on the ten output files OCHAN0 through OCHAN9. The file is selected by an integer parameter in the range [0...9].

External file names associated with these internal file names are stored in the array FILENAME with index values [MAXFONT...MAXFNT+11]. (Entry "MAXFNT+1" is reserved for temporary storage of font names, when the font itself was preloaded by TEXPRE)

CLOSE

Executes RESET(0CHANx) to close a file.

RWRITEFILE

Opens file 0CHANx for output using REWRITE with a nonstandard parameter list.

This procedure is completely analogous to the PDP-10 PASCAL `rewrite` procedure. Its arguments FNAME, FDIRECTORY, and FDEVICE are required by it precisely in the given form.

SENDCH

Outputs one character to 0CHANx.

SENDLN

Outputs a carriage return and line feed to 0CHANx.

SENDSTARTED

Executes SENDSTARTED:=EOF(0CHANx). The result is true only if this output file has been started but not opened.

§5 Terminal I/O and Output to the Error File

These procedures handle I/O from and to the terminal and output to the error file. The internal file names used are TERIN, TEROUT, and ERRFIL. The external names for both input and output to the terminal is "FOOBARTTY" and thus TERIN and TEROUT must each be reopened every time there is a switch between input and output or vice versa. The external name of ERRFIL is "ERRORSTEM".

FORCEBUFFEROUT

Completes output to the terminal before doing input from the terminal. Uses the nonstandard Pascal function BREAK(TEROUT).

INCHTER

Executes BREAK(TEROUT), opens TERIN for input, inputs the ordinal value of one character from TERIN, and skips to a carriage return.

INITSYSDEP

Opens TEROUT and ERRFIL and does other initializations.

INLNTER

Reads one line from TERIN.

OUTCHERR

Outputs one character to ERRFIL.

OUTCHTER

Outputs one character to TEROUT.

OUTLNERR

Outputs a carriage return and line feed to ERRFIL.

PRINT

Outputs a string from STRINGPOOL to TEROUT and ERRFIL.

PRINTFILENAME

Outputs a file name from the array FILENAME to TEROUT and ERRFIL.

PRINTINT

Outputs a print image of an integer to EROUT and ERRFIL.

PRINTLN

Outputs a carriage return and line feed and a string from STRINGPOOL to TEROUT and ERRFIL.

PRINTOCTAL

Outputs the octal print image of an integer to TEROUT and ERRFIL.

PRINTREAL

Output a print image of REAL number to TEROUT and ERRFIL.

TRACELINE

Displays a line to the terminal and the error file. Awaits a signal from the terminal. Anything except a carriage return causes this procedure to read a new line from the terminal replacing the displayed line.

§6 Read Font Information

Each font used by TEX has an associated font information file –called TFM, for TEX Font Metrics– that must be read into the arrays FONTINFO, FMEM (updating also the pointer FMEMPTR), WDBASE, HTBASE, DPBASE, ICBASE, LGBASE, KRBASE, EXTBASE, and PARBASE, for internal use by the TEX program.

In DEC machines, the name of this file is obtained by appending the extension code "TFM" to the font file name. For example, the TEX font metrics for the font CMR10 appear on the file CMR10.TFM.

READFONTINFO

Takes the font information from a file and puts it in the internal character metric arrays employed by TEX. The contents of these arrays is described in *TEX.DOC*. Notice that, sometimes, FONTINFO is set to integer 0, and that FONTFIL.FOURBYTES is assigned directly to FONTINFO entries. This works correctly for the PDP-10 pascal compiler where fields are packed left-to-right in each word (thus, unused bits are always on the right).

§7 Outputting Initialized Tables from TEXPRE

These procedures output various tables from TEXPRE for later input to TEX to initialize TEX. The file being written has the internal name TBLFIL and the external name "TEXINITBL".

Thus, TEX does not have to go through all the initialization; it is enough to read the tables from that file.

SETTABLESIZES

Opens file TBLFIL and outputs twelve numbers giving sizes of tables to follow.

WRITEDELIMTB

Outputs the delimiters table.

WRITEEQTB

Outputs the equivalents and related tables.

WRITEFMEM

Outputs the font memory.

WRITEHYPHENTB

Outputs hyphenation tables.

WRITEPAGETB

Writes-out original contents of the memory table. This table contains such paremeters as `\hsize`, `\vsize`, `\parindent`, `\topbaseline`, `\varunit`, etc.

WRITESECONDMEM

Outputs the latter part of the array MEM. It is necessary to initialize this part because it may contain names (character strings) of control sequences that were defined in the files preloaded by TEXPRE.

§8 Reading Initialized Tables into TEX

These procedures read various tables into TEXfor initialization. The file being read has the internal name TBLFIL and the external name "TEXINITBL".

GETTABLESIZES

Opens file TBLFIL for input and reads twelve integers giving sizes of the tables which follow on this file.

INITDELIMTB

Inputs the delimiters table.

INITEQTB

Inputs the equivalents and related tables.

INITFMEM

Inputs the font memory.

INITHYPHENTB

Inputs the hyphenation tables.

INITPAGETB

Inputs the page memory. Cf. WRITEPAGETB.

INITSECONDMEM

Inputs the latter part of the array MEM. Cf. WRITESECONDMEM.

§9 Handling the Device-independent Output File, DVI

This file contains the main output from TEX. Its internal name is OUTFIL. Its default external name is "TEXTOUTDVI".

CLOSEOUT

Called by TEX just before stopping to write postamble information on OUTFIL. This information is partially system dependent, because it contains font file names.

DECLAREOFIL

Initializes the output on the file indicated by its argument; that is, sets up correspondence between the internal file name OUTFIL and the external name.

This procedure is called when the name of the output file is first known. It then opens file OUTFIL for writing on a certain external file. The name of that external file name is the default (system, dependent, "TEXTOUTDVI" or something like that) if no input file was ever mentioned, that is, if all user input was given on the terminal. Otherwise, it is a name related in some form with the name of the first mentioned input file.

DVI

Packs a byte into a quarter word position of a single word buffer DVIWORD.INT, and outputs this word to OUTFIL when filled.

INTOUT

Breaks an integer into four bytes and outputs these to OUTFIL.