The DVItype processor

(Version 3.6, December 1995)

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926 and MCS-8300984, and by the System Development Foundation. ‘TeX’ is a trademark of the American Mathematical Society.
Introduction. The \texttt{DVItype} utility program reads binary device-independent ("DVI") files that are produced by document compilers such as \TeX, and converts them into symbolic form. This program has two chief purposes: (1) It can be used to determine whether a DVI file is valid or invalid, when diagnosing compiler errors; and (2) it serves as an example of a program that reads DVI files correctly, for system programmers who are developing DVI-related software.

Goal number (2) needs perhaps a bit more explanation. Programs for typesetting need to be especially careful about how they do arithmetic; if rounding errors accumulate, margins won’t be straight, vertical rules won’t line up, and so on. But if rounding is done everywhere, even in the midst of words, there will be uneven spacing between the letters, and that looks bad. Human eyes notice differences of a thousandth of an inch in the positioning of lines that are close together; on low resolution devices, where rounding produces effects four times as great as this, the problem is especially critical. Experience has shown that unusual care is needed even on high-resolution equipment; for example, a mistake in the sixth significant hexadecimal place of a constant once led to a difficult-to-find bug in some software for the Alphatype CRS, which has a resolution of 5333 pixels per inch (make that 5333.33333333 pixels per inch). The document compilers that generate DVI files make certain assumptions about the arithmetic that will be used by DVI-reading software, and if these assumptions are violated the results will be of inferior quality. Therefore the present program is intended as a guide to proper procedure in the critical places where a bit of subtlety is involved.

The first DVItype program was designed by David Fuchs in 1979, and it went through several versions on different computers as the format of DVI files was evolving to its present form. Peter Breitenlohner helped with the latest revisions.

The \texttt{banner} string defined here should be changed whenever DVItype gets modified.

\begin{verbatim}
define my_name ≡ 'dvitype'
define banner ≡ 'This is DVItype, Version 3.6'  { printed when the program starts }
\end{verbatim}

The binary input comes from \texttt{dvi file}, and the symbolic output is written on Pascal’s standard \texttt{output} file. The term \texttt{print} is used instead of \texttt{write} when this program writes on \texttt{output}, so that all such output could easily be redirected if desired.

\begin{verbatim}
define print(#) ≡ write(stdout,#)
define print_ln(#) ≡ writeLn(stdout,#)
\end{verbatim}

\texttt{program DVI type(dvi file, output);}  \label{Labels in the outer block 4*}
\begin{verbatim}
label (Labels in the outer block 4*)
const (Constants in the outer block 5*)
type (Types in the outer block 8*)
var ( Globals in the outer block 10)
(Define parse_arguments 112*)
procedure initialize;  { this procedure gets things started properly }
  var i: integer;  { loop index for initializations }
  begin kpse_set_program_name(argv[0], my_name); parse_arguments; print(banner);
    print_ln(version_string); { Set initial values 11 } 
  end;
\end{verbatim}

\texttt{Label done} is used when stopping normally.

\begin{verbatim}
define done = 30  { go here when finished with a subtask }
\end{verbatim}

This code is used in section 3*.
The following parameters can be changed at compile time to extend or reduce DVIType’s capacity.

\begin{verbatim}
\begin{verbatim}
max_fonts = 500; \{ maximum number of distinct fonts per DVI file \}
max_widths = 25000; \{ maximum number of different characters among all fonts \}
line_length = 79; \{ bracketed lines of output will be at most this long \}
stack_size = 100; \{ DVI files shouldn’t push beyond this depth \}
name_size = 10000; \{ total length of all font file names \}
\end{verbatim}
\end{verbatim}

This code is used in section 3*.

If the DVI file is badly malformed, the whole process must be aborted; DVIType will give up, after issuing an error message about the symptoms that were noticed.

Such errors might be discovered inside of subroutines inside of subroutines, so a procedure called \texttt{jump\_out} has been introduced.

\begin{verbatim}
\textbf{define} \texttt{jump\_out} \equiv \texttt{uexit(1)}
\textbf{define} \texttt{abort(#)} \equiv
  \textbf{begin} \texttt{write\_ln(stderr,#); jump\_out;}
  \textbf{end}
\textbf{define} \texttt{bad\_dvi(#)} \equiv \texttt{abort(`Bad DVI file: #,`!')}
\end{verbatim}
The character set. Like all programs written with the WEB system, DVItype can be used with any character set. But it uses ASCII code internally, because the programming for portable input-output is easier when a fixed internal code is used, and because DVI files use ASCII code for file names and certain other strings.

The next few sections of DVItype have therefore been copied from the analogous ones in the WEB system routines. They have been considerably simplified, since DVItype need not deal with the controversial ASCII codes less than '40' or greater than '176'. If such codes appear in the DVI file, they will be printed as question marks.

\[
\text{(Types in the outer block 8*) } \equiv \\
\text{ASCII code } = 0 \ldots 255; \quad \{ \text{a subrange of the integers} \}
\]

See also sections 9* and 21.

This code is used in section 3*.

The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lower case letters. Nowadays, of course, we need to deal with both upper and lower case alphabets in a convenient way, especially in a program like DVItype. So we shall assume that the Pascal system being used for DVItype has a character set containing at least the standard visible characters of ASCII code ("!" through "~").

Some Pascal compilers use the original name char for the data type associated with the characters in text files, while other Pascals consider char to be a 64-element subrange of a larger data type that has some other name. In order to accommodate this difference, we shall use the name text_char to stand for the data type of the characters in the output file. We shall also assume that text_char consists of the elements chr(first_text_char) through chr(last_text_char), inclusive. The following definitions should be adjusted if necessary.

\[
\text{define text_char } \equiv \text{ASCII code } \quad \{ \text{the data type of characters in text files} \}
\]
\[
\text{define first_text_char } = 0 \quad \{ \text{ordinal number of the smallest element of text_char} \}
\]
\[
\text{define last_text_char } = 255 \quad \{ \text{ordinal number of the largest element of text_char} \}
\]

\[
\text{(Types in the outer block 8*) } + \equiv \\
\text{text_file } = \text{packed file of text_char};
\]
To prepare these files for input, we reset them. An extension of Pascal is needed in the case of tfm_file, since we want to associate it with external files whose names are specified dynamically (i.e., not known at compile time). The following code assumes that ‘reset(f, s)’ does this, when f is a file variable and s is a string variable that specifies the file name. If eof(f) is true immediately after reset(f, s) has acted, we assume that no file named s is accessible.

```pascal
procedure open_dvi_file;  { prepares to read packed bytes in dvi_file }
  begin
    resetbin(dvi_file, dvi_name);  cur_loc ← 0;
  end;

procedure open_tfm_file;  { prepares to read packed bytes in tfm_file }
  var full_name: ↑char;
  begin
    full_name ← kpse_find_tfm(cur_name);
    if full_name then
      begin
        tfm_file ← fopen(full_name, FOPEN_RBIN_MODE);
      end
    else begin tfm_file ← nil;
    end
  end;
```

If you looked carefully at the preceding code, you probably asked, “What are cur_loc and cur_name?” Good question. They’re global variables: cur_loc is the number of the byte about to be read next from dvi_file, and cur_name is a string variable that will be set to the current font metric file name before open_tfm_file is called.

```pascal
( Globals in the outer block 10 ) ⊔
cur_loc: integer;  { where we are about to look, in dvi_file }
cur_name: ↑char;  { external name }
```

Finally we come to the routines that are used only if random_reading is true. The driver program below needs two such routines: dvi_length should compute the total number of bytes in dvi_file, possibly also causing eof(dvi_file) to be true; and move_to_byte(n) should position dvi_file so that the next get_byte will read byte n, starting with n = 0 for the first byte in the file.

Such routines are, of course, highly system dependent. They are implemented here in terms of two assumed system routines called set_pos and cur_pos. The call set_pos(f, n) moves to item n in file f, unless n is negative or larger than the total number of items in f; in the latter case, set_pos(f, n) moves to the end of file f. The call cur_pos(f) gives the total number of items in f, if eof(f) is true; we use cur_pos only in such a situation.

```pascal
function dvi_length: integer;
  begin
    xfseek(dvi_file, 0, 2, dvi_name);
    cur_loc ← xftell(dvi_file, dvi_name);
    dvi_length ← cur_loc;
  end;

procedure move_to_byte(n: integer);
  begin
    xfseek(dvi_file, n, 0, dvi_name);
    cur_loc ← n;
  end;
```
The starting page specification is recorded in two global arrays called start_count and start_there. For example, ‘1.*-5’ is represented by start_there[0] = true, start_count[0] = 1, start_there[1] = false, start_there[2] = true, start_count[2] = -5. We also set start_vals = 2, to indicate that count 2 was the last one mentioned. The other values of start_count and start_there are not important, in this example.

(Globals in the outer block 10) +≡

\[
\begin{align*}
\text{start_count: array } [0 \ldots 9] \text{ of integer; } & \{ \text{count values to select starting page} \} \\
\text{start_there: array } [0 \ldots 9] \text{ of boolean; } & \{ \text{is the start_count value relevant?} \} \\
\text{start_vals: 0 \ldots 9; } & \{ \text{the last count considered significant} \} \\
\text{count: array } [0 \ldots 9] \text{ of integer; } & \{ \text{the count values on the current page} \}
\end{align*}
\]

Initializations are done sooner now.

No dialog.

47* During the dialog, DVItype will treat the first blank space in a line as the end of that line. Therefore input_in makes sure that there is always at least one blank space in buffer.

No dialog.

No dialog (50).

No dialog (51).

No dialog (52).

No dialog (53).

No dialog (54).

No dialog (55).
After the dialog is over, we print the options so that the user can see what DVI type thought was specified.

(Print all the selected options 56*)

\[
\text{print}_\text{ln}(`\text{Options selected:}`); \text{ print}(`\text{Starting page=}`); \\
\text{for } k \leftarrow 0 \text{ to } \text{start vals do } \\
\quad \text{begin if } \text{start there}[k] \text{ then } \text{print}(`\text{start count}[k]:1`) \\
\quad \quad \text{else } \text{print}(`\text{*}`); \\
\quad \quad \text{if } k < \text{start vals then print}(`\text{.}`) \\
\quad \quad \text{else } \text{print}_\text{ln}(`\text{}`); \\
\quad \text{end;}
\]

\[
\text{print}_\text{ln}(`\text{Maximum number of pages}=`, \text{max pages}:1); \\
\text{print}(`\text{Output level}=`, \text{out mode}:1);
\]

\[
\text{case out mode of } \\
\quad \text{errors only: print}_\text{ln}(`\text{(showing bops, fonts, and error messages only)}`); \\
\quad \text{terse: print}_\text{ln}(`\text{(terse)}`); \\
\quad \text{mnemonics only: print}_\text{ln}(`\text{(mnemonics)}`); \\
\quad \text{verbose: print}_\text{ln}(`\text{(verbose)}`); \\
\quad \text{the works: if random reading then print}_\text{ln}(`\text{(the works)}`) \\
\quad \quad \text{else begin out mode } \leftarrow \text{verbose; print}_\text{ln}(`\text{(the works: same as level 3 in this DVItype)}`); \\
\quad \quad \text{end;}
\]

\[
\text{end;}
\]

\[
\text{print}(`\text{Resolution=}`); \text{ print}_\text{real}(\text{resolution},12,8); \text{ print}_\text{ln}(`\text{pixels per inch}`);
\]

\[
\text{if new mag > 0 then } \\
\quad \text{begin print}(`\text{New magnification factor=}`); \text{ print}_\text{real}(\text{new mag}/1000.0,8,3); \text{ print}_\text{ln}(`\text{}`) \\
\quad \text{end}
\]

This code is used in section 107*. 

\[\]
59* The following subroutine does the necessary things when a `fnt_def` command is being processed.

```c
procedure define_font(e : integer); { e is an external font number }
var f: 0 .. max_fonts; p: integer; { length of the area/directory spec }
n: integer; { length of the font name proper }
c, q, d, m: integer; { check sum, scaled size, design size, magnification }
r: 0 .. name_size; { current filename length }
j, k: 0 .. name_size; { indices into names }
mismatch: boolean; { do names disagree? }
begin if nf = max_fonts then
  abort(`DVItype\capacity\ exceeded, (max_fonts=`max_fonts : 1, `)!)`);
font_num[nf] ← e; f ← 0;
while font_num[f] ≠ e do incr(f);
(Read the font parameters into position for font nf, and print the font name 61);
if ((out_mode = the_works) ∧ in_postamble) ∨ ((out_mode < the_works) ∧ ¬in_postamble) then
  begin if f < nf then print_in(`---this\font\ was\ already\ defined!`);
  end
else begin if f = nf then print_in(`---this\font\ wasn`t\ loaded\ before!`);
   end;
if f = nf then { Load the new font, unless there are problems 62* }
else { Check that the current font definition matches the old one 60 };
end;
```

62* 《Load the new font, unless there are problems 62*》≡
begin (Move font name into the `cur_name` string 66*);
open_tfm_file;
if EOF(tfm_file) then print(`---not\ loaded, `TFM_file\ can`t\ be\ opened!`)
else begin if (q ≤ 0) ∨ (q ≥ 1000000000) then print(`---not\ loaded, `bad\ scale\(``, `1, `)!)`)
  else if (d ≤ 0) ∨ (d ≥ 1000000000) then print(`---not\ loaded, `bad\ design\ size\(``, `1, `)!)`)
  else if in_TFM(q) then {Finish loading the new font info 63} ;
end;
if out_mode = errors_only then print_in(`")`);
if tfm_file then xfclose(tfm_file, `cur_name`);  { should be the kpse_find_tfm result }
free(`cur_name`);  { We xmalloc’d this before we got called. }
end
This code is used in section 59*.

64* If p = 0, i.e., if no font directory has been specified, `DVItype` is supposed to use the default font directory, which is a system-dependent place where the standard fonts are kept. The string variable `default_directory` contains the name of this area.
Under Unix, users have a path searched for fonts, there’s no single default directory.

65* 《No initialization needs to be done. Keep this module to preserve numbering.》
The string `cur_name` is supposed to be set to the external name of the TFM file for the current font. We do not impose a maximum limit here. It’s too bad there is a limit on the total length of all filenames, but it doesn’t seem worth reprogramming all that.

\[
\begin{align*}
\text{define } & \text{ name\_start } \equiv \text{ font\_name}[nf] \\
\text{define } & \text{ name\_end } \equiv \text{ font\_name}[nf + 1]
\end{align*}
\]

\langle \text{Move font name into the cur\_name string } 66^* \rangle \equiv
\begin{align*}
& r \leftarrow \text{ name\_end } - \text{ name\_start}; \text{ cur\_name } \leftarrow \text{xmalloc\_array}(\text{char}, r) \\
& \{ \text{strncpy might be faster, but it’s probably a good idea to keep the xchr translation.} \}
\end{align*}
\begin{align*}
\text{for } k \leftarrow \text{ name\_start to name\_end do} & \\
& \text{begin } \text{cur\_name}[k - \text{ name\_start}] \leftarrow \text{xchr}[\text{names}[k]] \end{align*}
\begin{align*}
\text{end;} & \\
& \text{cur\_name}[r] \leftarrow 0; \{ \text{Append null byte for C.} \}
\end{align*}

This code is used in section 62*. 

§66  DVI type changes for C  DEFining fonts  409
Before we get into the details of do_page, it is convenient to consider a simpler routine that computes the first parameter of each opcode.

\[
\begin{align*}
\text{define } & \text{four_cases}(#) \equiv #, # + 1, # + 2, # + 3 \\
\text{define } & \text{eight_cases}(#) \equiv \text{four_cases}(#), \text{four_cases}(# + 4) \\
\text{define } & \text{sixteen_cases}(#) \equiv \text{eight_cases}(#), \text{eight_cases}(# + 8) \\
\text{define } & \text{thirty_two_cases}(#) \equiv \text{sixteen_cases}(#), \text{sixteen_cases}(# + 16) \\
\text{define } & \text{sixty_four_cases}(#) \equiv \text{thirty_two_cases}(#), \text{thirty_two_cases}(# + 32)
\end{align*}
\]

\text{function } \text{first_par}(o : \text{eight_bits}: \text{integer}; \\
\text{begin case } o \text{ of} \\
\text{sixty_four_cases(set_char_0), sixty_four_cases(set_char_0 + 64): first_par } \leftarrow o - \text{set_char_0}; \\
\text{set1}, \text{put1}, \text{fnt1}, \text{xxx1}, \text{fnt_def1}: \text{first_par } \leftarrow \text{get_byte}; \\
\text{set1 } + 1, \text{put1 } + 1, \text{fnt1 } + 1, \text{xxx1 } + 1, \text{fnt_def1 } + 1: \text{first_par } \leftarrow \text{get_two_bytes}; \\
\text{set1 } + 2, \text{put1 } + 2, \text{fnt1 } + 2, \text{xxx1 } + 2, \text{fnt_def1 } + 2: \text{first_par } \leftarrow \text{get_three_bytes}; \\
\text{right1}, w1, x1, down1, y1, z1: \text{first_par } \leftarrow \text{signed_byte}; \\
\text{right1 } + 1, w1 + 1, x1 + 1, down1 + 1, y1 + 1, z1 + 1: \text{first_par } \leftarrow \text{signed_pair}; \\
\text{right1 } + 2, w1 + 2, x1 + 2, down1 + 2, y1 + 2, z1 + 2: \text{first_par } \leftarrow \text{signed_trio}; \\
\text{set1 } + 3, \text{set_rule}, \text{put1 } + 3, \text{put_rule}, \text{right1 } + 3, w1 + 3, x1 + 3, down1 + 3, y1 + 3, z1 + 3, \text{fnt1 } + 3, \\
\text{xxx1 } + 3, \text{fnt_def1 } + 3: \text{first_par } \leftarrow \text{signed_quad}; \\
\text{nop, bop, cop, push, pop, pre, post, post_post, undefined_commands: first_par } \leftarrow 0; \\
w0: \text{first_par } \leftarrow w; \\
x0: \text{first_par } \leftarrow x; \\
y0: \text{first_par } \leftarrow y; \\
z0: \text{first_par } \leftarrow z; \\
\text{sixty_four_cases(fnt_num_0): first_par } \leftarrow o - \text{fnt_num_0}; \\
\text{othercases abort(”internal_error”);} \\
\text{endcases}; \\
\text{end};
Commands are broken down into "major" and "minor" categories: A major command is always shown in full, while a minor one is put into the buffer in abbreviated form. Minor commands, which account for the bulk of most DVI files, involve horizontal spacing and the typesetting of characters in a line; these are shown in full only if out_mode ≥ verbose.

```plaintext
define show(#) ≡
    begin flush_text; showing ← true; print(a : 1, `:`, #);
    if show_opcodes ∧ (o ≥ 128) then print(`{`, o : 1, `}`);
end

define major(#) ≡
    if out_mode > errors_only then show(#)

define minor(#) ≡
    if out_mode > terse then
        begin showing ← true; print(a : 1, `:`,
        if show_opcodes ∧ (o ≥ 128) then print(`{`, o : 1, `}`);
    end

define error(#) ≡
    if ¬showing then show(#)
    else print(`=`, #)

(Translate the next command in the DVI file; goto 9999 with do_page = true if it was eop; goto 9998 if premature termination is needed 80*)

begin a ← cur_loc; showing ← false; o ← get_byte; p ← first_par(o);
if eof(dvi_file) then bad_dvi(`the file ended prematurely');
(Start translation of command o and goto the appropriate label to finish the job 81);
fin_set: (Finish a command that either sets or puts a character, then goto move_right or done 89);
fin_rule: (Finish a command that either sets or puts a rule, then goto move_right or done 90);
move_right: (Finish a command that sets h ← h + q, then goto done 91);
show_state: (Show the values of ss, h, v, w, x, y, z, hh, and vv; then goto done 93);
done: if showing then print ln(`=');
end
```

This code is used in section 79.
The main program. Now we are ready to put it all together. This is where DVItype starts, and where it ends.

begin initialize;  { get all variables initialized }
(Print all the selected options 56*);
(Process the preamble 109);
if out_mode = the_works then  { random_reading = true }
begin  { Find the postamble, working back from the end 100 }
in_postamble ← true; read_postamble; in_postamble ← false;
(Count the pages and move to the starting page 102);
end;
skip_pages(false);
if ¬in_postamble then  { Translate up to max_pages pages 111 }
if out_mode < the_works then
begin if ¬in_postamble then skip_pages(true);
if signed_quad ≠ old_backpointer then
   println(`backpointer in byte, cur_loc − 4 : 1, `should be`, old_backpointer : 1, `!`);
read_postamble;
end;
end.

The conversion factor conv is figured as follows: There are exactly \( \frac{n}{d} \) decimicrons per DVI unit, and 254000 decimicrons per inch, and resolution pixels per inch. Then we have to adjust this by the stated amount of magnification.

(Compute the conversion factors 110*) \( \equiv \)

numerater ← signed_quad; denominator ← signed_quad;
if numerator ≤ 0 then bad_dvi(`numerator is `, numerator : 1);
if denominator ≤ 0 then bad_dvi(`denominator is `, denominator : 1);
println(`numerator/denominator=`, numerator : 1, `\;`, denominator : 1);
ifm_conv ← (25400000.0/numerator) * (denominator/473628672)/16.0;
conv ← (numerator/254000.0) * (resolution/denominator); mag ← signed_quad;
if new_mag > 0 then mag ← new_mag
else if mag ≤ 0 then bad_dvi(`magnification is `, mag : 1);
true_conv ← conv; conv ← true_conv * (mag/1000.0); println(`magnification=` , mag : 1, `; `);
println(`pixels per DVI unit`)

This code is used in section 109.
§112* System-dependent changes. Parse a Unix-style command line.

```
define argument_is(#) ≡ (strcmp(long_options[option_index].name, #) = 0)

(Define parse_arguments 112*) ≡
procedure parse_arguments;
  const n_options = 8; {Pascal won’t count array lengths for us.}
  var long_options: array [0..n_options] of getopt_struct;
  getopt_return_val: integer; option_index: c_int_type; current_option: 0..n_options; end_num: ↑char;
    { for page-start }
begin (Define the option table 113*);
repeat getopt_return_val ← getopt_long_only(argc, argv, "", long_options, address_of(option_index));
  if getopt_return_val = −1 then
    begin do nothing; {End of arguments; we exit the loop below.}
  end else if argument_is("-h") then
    begin usage(my_name);
  end else if argument_is("-help") then
    begin usage_help(DVITYPE_HELP, nil);
  end else if argument_is("-version") then
    begin print_version_and_exit(banner, nil, "D.E. Knuth", nil);
  end else if argument_is("-output-level") then
    begin if (optarg[0] < '0') ∨ (optarg[0] > '4') ∨ (optarg[1] ≠ 0) then
      begin write_ln(stderr, 'Value for \-output-level\ must be >= 0 and <= 4. ');
        uexit(1);
      end;
      out_mode ← optarg[0] − '0';
    end
  else if argument_is("-page-start") then
    begin (Determine the desired start_count values from optarg 117*);
  end else if argument_is("-max-pages") then
    begin max_pages ← atou(optarg);
  end else if argument_is("-dpi") then
    begin resolution ← atof(optarg);
  end else if argument_is("-magnification") then
    begin new_mag ← atou(optarg);
  end until getopt_return_val = −1; {Now optind is the index of first non-option on the command line.}
if (optind + 1 ≠ argc) then
  begin write_ln(stderr, my_name, "Need exactly one file argument."); usage(my_name);
end;
  dvi_name ← extend_filename(cmdline(optind), "dvi");
end;
```
Here are the options we allow. The first is one of the standard GNU options.

(Determine the desired start_count values from optarg
\[117\]) \(\equiv\)

\[
k \leftarrow 0; \quad \{ \text{which \count register we're on} \}
\]
\[
m \leftarrow 0; \quad \{ \text{position in optarg} \}
\]
\[
\text{while optarg}[m] \text{ do}
\]
\[
\quad \text{begin if optarg}[m] = "*" then}
\]
\[
\quad \quad \text{begin start_there}[k] \leftarrow \text{false}; \text{incr}(m);
\]
\[
\quad \text{end}
\]
\[
\quad \text{else if optarg}[m] = "." then}
\]
\[
\quad \quad \text{begin incr}(k);
\]
\[
\text{if } k \geq 10 \text{ then } \begin{align*}
\quad & \text{begin write}_\text{ln}(\text{stderr, my_name, } \text{\"More than\ten\count registers\specified\"});
\quad & \text{uexit}(1);
\quad & \text{end};
\quad & \text{incr}(m);
\quad \end{align*}
\]
\[
\text{end}
\]
\[
\text{else begin}
\]
\[
\quad \text{start_count}[k] \leftarrow \text{strtol(optarg + m, address_of(end_num), 10)};
\]
\[
\text{if end_num = optarg + m then}
\]
\[
\quad \text{begin write}_\text{ln}(\text{stderr, my_name, } \text{\"-page-start\values must be numeric or \*\"});
\quad \text{uexit}(1);
\quad \text{end};
\]
\[
\quad \text{start_there}[k] \leftarrow \text{true}; \quad m \leftarrow m + \text{end_num} - (\text{optarg + m});
\]
\[
\text{end;}
\]
\[
\text{start_vals} \leftarrow k;
\]

This code is used in section \[112\].
How many pages to do.

\[\text{Define the option table 113*} \equiv\]
\begin{verbatim}
  long_options[current_option].name ← `max-pages`; long_options[current_option].has_arg ← 1;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
  max_pages ← 1000000; { default }
\end{verbatim}

Resolution, in pixels per inch.

\[\text{Define the option table 113*} \equiv\]
\begin{verbatim}
  long_options[current_option].name ← `dpi`; long_options[current_option].has_arg ← 1;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
  resolution ← 300.0; { default }
\end{verbatim}

Magnification to apply.

\[\text{Define the option table 113*} \equiv\]
\begin{verbatim}
  long_options[current_option].name ← `magnification`; long_options[current_option].has_arg ← 1;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
  new_mag ← 0; { default is to keep the old one }
\end{verbatim}

Whether to show numeric opcodes.

\[\text{Define the option table 113*} \equiv\]
\begin{verbatim}
  long_options[current_option].name ← `show-opcodes`; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← address_of(show_opcodes); long_options[current_option].val ← 1;
  incr(current_option);
\end{verbatim}

\[\text{(Globals in the outer block 10) } \equiv\]
\begin{verbatim}
show_opcodes: c_int_type;
\end{verbatim}

An element with all zeros always ends the list.

\[\text{Define the option table 113*} \equiv\]
\begin{verbatim}
  long_options[current_option].name ← 0; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0;
\end{verbatim}

Global filenames.

\[\text{(Globals in the outer block 10) } \equiv\]
\begin{verbatim}
dvi_name: const_c_string;
\end{verbatim}
Index. Pointers to error messages appear here together with the section numbers where each identifier is used.

The following sections were changed by the change file: 1, 3, 4, 5, 7, 8, 9, 23, 24, 28, 42, 43, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 59, 62, 64, 65, 66, 75, 80, 107, 110, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125.

-dpi: 119*
-accuracy: 112*
-be: 113*
-magnification: 120*
-max-pages: 118*
-output-level: 115*
-page-start: 116*
-show-opcodes: 121*
-version: 114*
a: 27, 79, 82.
abort: 77* 59*, 61, 75* 102.
abs: 63, 73, 85, 91, 92.
address_of: 112*, 117*, 121*
after_pre: 101, 102, 109.
all 223s: 100.
alpha: 34, 37, 38.
argv: 112*
argument_is: 112*
argv: 3* 112*

arithmetic overflow...: 91, 92.
ASCII_code: 8* 9*, 10, 30, 67, 70.
atoi: 112*
atou: 112*
b: 27.
backpointer...should be p: 99, 107*
bad design size: 62*
Bad DVI file: 7*
bad postamble pointer: 105.
bad scale: 62*
bad_char: 82, 87.
bad_dvi: 7* 80*, 96, 99, 100, 102, 105, 109, 110*, 111.
banner: 1* 3* 112*.
beta: 34, 37, 38.
beware: check sums do not agree: 63.
beware: design sizes do not agree: 63.
boolean: 34, 42* 44, 57, 59* 78, 79, 82, 95, 97.
bop occurred before eop: 83.
bop_seem: 95.
break: 46.
Breitenlohner, Peter: 1*
buffer: 47*
byte n is not bop: 99, 102.
byte n is not post: 100.
byte n is not postpost: 106.
byte_file: 21, 22.
b0: 25, 26, 35, 36, 37.
b1: 25, 26, 35, 37.
b2: 25, 26, 35, 37.
b3: 25, 26, 35, 37.
c: 27, 59*
c_int_type: 112* 122*
change_font: 77, 82, 86.
char: 9* 23*, 24*, 66*, 112*
char_pixel_width: 30, 89.
char_width: 30, 39, 89.
char_width_end: 30, 39.
character c invalid...: 89.
check sum: 18.
check sum doesn't match: 60.
check sums do not agree: 63.
Chinese characters: 15, 89.
chr: 9* 10, 12.
cmdline: 112*
const_c_string: 124*
conv: 39, 40, 61, 63, 76, 110*
count: 42* 44, 99, 102, 111.
cur_font: 77, 78, 79, 84, 85, 89, 94.
cur_name: 23*, 24*, 62*, 66*
cur_pos: 28*
d: 27, 59*
decr: 6, 83, 96, 100, 109, 111.
deeper than claimed...: 83.
default_directory: 64*
define_font: 59* 86, 96, 99, 106.
den: 15, 17, 19.
denominator: 39, 103, 110*
denominator doesn't match: 103.
denominator is wrong: 110*
design size doesn't match: 60.
design size does not agree: 63.
do nothing: 6, 96, 112*
do_page: 71, 75*, 77, 78, 79, 81, 83, 95, 111.
done: 4* 79, 80*, 81, 82, 83, 86, 87, 89, 90, 91, 92, 93, 94, 111.
down_the_drain: 95, 96.
down1: 15, 16, 75*, 85.
down2: 15.
down3: 15.
down4: 15.
DVI files: 13.
DVI type changes for C

font name doesn't match: fnt4

four_cases: 112* 81, 82, 84, 85, 86, 96.

free: 62*

Fuchs, David Raymond: 1* 13, 20.

full_name: 23*

get_byte: 27, 28* 61, 75* 80, 87, 96, 99, 100, 102, 105, 106, 109.

get_three_bytes: 27, 75*

get_two_bytes: 27, 75* 103.

g getopt: 112*

g getopt_long_only: 112*

g getopt_return_val: 112*

g getopt_struc: 112*

h: 72.

has_arg: 113* 114* 115* 116* 118* 119* 120* 121* 123*

hh: 72, 79, 83, 84, 89, 90, 91, 93.

hhhh: 79, 91.

hhstack: 72, 83.

hstack: 72, 83.

i: 3* 17.

ID byte is wrong: 100.

id_byte: 17, 100, 105, 109.

identification...should be n: 105, 109.

illegal command at byte n: 96.

in_postamble: 57, 58, 59* 95, 99, 102, 107* 111.

in_TFM: 34, 37, 62*.

in_width: 33, 37, 40.

incr: 6, 27, 59* 60, 63, 70, 83, 94, 99, 102, 113*

114* 115* 116* 117* 118* 119* 120* 121* 123*

infinity: 91, 92.

initialize: 3* 107*.

input_bin: 47*

integer: 3* 21, 24* 27, 28* 30, 32, 33, 34, 39,

41, 42* 59* 72, 73, 75* 76, 78, 79, 82, 85, 95, 97,

101, 103, 108, 112*.

invalid_font: 30, 31, 32, 79, 89, 94.

invalid_width: 30, 40, 89.

j: 59*

Japanese characters: 15, 89.

jump_out: 7*

k: 17, 32, 34, 44, 59* 69, 82, 95, 99, 103, 108.

kpse_find_fmt: 23* 62*

kpse_set_program_name: 3*

last_text_char: 9* 12.

lh: 34, 35.

line_length: 7*, 67, 69, 70.

long_options: 112* 113* 114* 115* 116* 118* 119* 120* 121* 123*

m: 59* 103, 108.

mag: 15, 17, 18, 19, 39, 103, 110*

magnification doesn't match: 103.

magnification is wrong: 110*
right1: 15, 16, 75, 84.
right2: 15.
right3: 15.
right4: 15.
round: 35, 40, 61, 63.
rule_pixels: 15, 76, 90.
s: 78.
scaled: 61.
scaled size doesn't match: 60.
scan_bop: 95, 99, 111.
set_char_0: 15, 16, 75, 81.
set_char_1: 15.
set_char_127: 15.
set_pos: 28.
set_rule: 13, 15, 16, 75, 81, 96.
sel1: 15, 16, 75, 81.
sel2: 15.
sel3: 15.
sel4: 15.
show: 80.
show_opcodes: 80, 121, 122.
show_state: 77, 79, 80, 83.
showing: 61, 78, 80, 87, 90, 91, 92, 93, 94, 95, 103.
signature...should be...: 105.
signed_byte: 27, 75.
signed_pair: 27, 75.
signed_quad: 27, 61, 75, 90, 96, 99, 100, 102, 103, 105, 107, 110.
signed_trio: 27, 75.
sixteen_cases: 75.
sixty_four_cases: 75, 86.
skip_pages: 95, 107.
sp: 17.
special_cases: 78, 81, 82.
ss: 78, 83, 93.
stack not empty...: 83.
stack_size: 57, 72, 74, 83.
start_count: 42, 44, 56, 117.
start_loc: 101, 102.
start_match: 44, 95, 102.
start_token: 42, 44, 56, 117.
start_vals: 42, 44, 56, 111, 117.
started: 95, 97, 98.
starting page number...: 102.
stderr: 7, 112, 117.
stdout: 3.
strncpy: 112.
string of negative length: 87.
strncpy: 66.
strtol: 117.
system dependencies: 2, 7, 9, 20, 21, 23, 26, 27, 28, 40, 41, 46, 47, 64, 66.
term_out: 46.
terse: 41, 56, 80.
text_buf: 67, 69, 70.
text_char: 9, 10.
text_file: 9.
text_ptr: 67, 68, 69, 70.
TFM files: 29.
TFM file can't be opened: 62.
TFM file is bad: 34.

*fn_check_sum: 33, 35, 63.
*fn_conv: 33, 35, 110.
*fn_design_size: 33, 35, 63.
*fn_file: 22, 23, 26, 33, 35, 62.
the file ended prematurely: 80, 96, 99.
the_works: 41, 56, 57, 59, 100, 103, 107, 115.
there are really n pages: 102, 104.

thirty_two_cases: 75.
this font is magnified: 63.
this font was already defined: 59.
this font wasn't loaded before: 59.
total_pages: 73, 102, 103, 104.
true: 2, 28, 34, 42, 44, 60, 79, 80, 82, 83, 87, 95, 99, 100, 102, 107, 117.
true_conv: 39, 61, 63, 110.
trunc: 76.
uexit: 7, 112, 117.
UNDEFINED: 32.
undefined_command: 82.
undefined_commands: 16, 75, 96.
update_terminal: 46.
usage: 112.
usage_help: 112.
v: 72.
verbose: 41, 56, 80.
version_string: 3.

vstack: 72, 83.
vw: 72, 79, 83, 85, 92, 93.
vstack: 72, 83.
vw: 82, 92.
w: 72.
warning: |h|...: 91.
warning: |v|...: 92.
warning: observed maxh...: 104.
warning: observed maxv...: 104.
warning: observed maxv...: 104.
width: 30, 36, 39, 40.
width_base: 30, 39, 40.
width_ptr: 30, 31, 34, 35, 36, 40.
wp: 34, 35, 36, 40.
write: 3.
write_line: 3, 7, 112, 117.
INDEX

wstack: 72, 83.
w0: 15, 16, 75*, 84.
w1: 15, 16, 75*, 84.
w2: 15.
w3: 15.
w4: 15.
x: 17, 72.
xchr: 10, 11, 12, 32, 66*, 69, 87, 109.
xfclose: 62*
xfseek: 28*
xftell: 28*
xmalloc: 62*
xmalloc_array: 66*
xord: 10, 12.
xstack: 72, 83.
xxx1: 15, 16, 75*, 82, 96.
xxx2: 15.
xxx3: 15.
xxx4: 15, 16.
x0: 15, 16, 75*, 84.
x1: 15, 16, 75*, 84.
x2: 15.
x3: 15.
x4: 15.
y: 72.
ystack: 72, 83.
y0: 15, 16, 75*, 85.
y1: 15, 16, 75*, 85.
y2: 15.
y3: 15.
y4: 15.
z: 34, 72.
zstack: 72, 83.
z0: 15, 16, 75*, 85.
z1: 15, 16, 75*, 85.
z2: 15.
z3: 15.
z4: 15.
\textbackslash{} (Cases for commands \textit{nop, bop, \ldots, pop} 83) \textbackslash{} Used in section 81.
\textbackslash{} (Cases for fonts 86) \textbackslash{} Used in section 82.
\textbackslash{} (Cases for horizontal motion 84) \textbackslash{} Used in section 81.
\textbackslash{} (Cases for vertical motion 85) \textbackslash{} Used in section 82.
\textbackslash{} (Check that the current font definition matches the old one 60) \textbackslash{} Used in section 59*.
\textbackslash{} (Compare the \textit{lust} parameters with the accumulated facts 104) \textbackslash{} Used in section 103.
\textbackslash{} (Compute the conversion factors 110*) \textbackslash{} Used in section 109.
\textbackslash{} (Constants in the outer block 5*) \textbackslash{} Used in section 3*.
\textbackslash{} (Count the pages and move to the starting page 102) \textbackslash{} Used in section 107*.
\textbackslash{} (Declare the function called \textit{special\_cases} 82) \textbackslash{} Used in section 79.
\textbackslash{} (Declare the procedure called \textit{scan\_bop} 99) \textbackslash{} Used in section 95.
\textbackslash{} (Define the option table \textit{113*, 114*, 115*, 116*, 118*, 119*, 120*, 121*, 123*}) \textbackslash{} Used in section 112*.
\textbackslash{} (Define \textit{parse\_arguments} 112*) \textbackslash{} Used in section 3*.
\textbackslash{} (Determine the desired \textit{start\_count} values from \textit{optarg} 117*) \textbackslash{} Used in section 112*.
\textbackslash{} (Find the postamble, working back from the end 100) \textbackslash{} Used in section 107*.
\textbackslash{} (Finish a command that changes the current font, then \textit{goto done} 94) \textbackslash{} Used in section 82.
\textbackslash{} (Finish a command that either sets or puts a character, then \textit{goto move\_right} or \textit{done} 89) \textbackslash{} Used in section 80*.
\textbackslash{} (Finish a command that either sets or puts a rule, then \textit{goto move\_right} or \textit{done} 90) \textbackslash{} Used in section 80*.
\textbackslash{} (Finish a command that sets \textit{h} \leftarrow \textit{h} + \textit{q}, then \textit{goto done} 91) \textbackslash{} Used in section 80*.
\textbackslash{} (Finish a command that sets \textit{v} \leftarrow \textit{v} + \textit{p}, then \textit{goto done} 92) \textbackslash{} Used in section 82.
\textbackslash{} (Finish loading the new font info 63) \textbackslash{} Used in section 62*.
\textbackslash{} (Globals in the outer block \textit{10, 22, 24*, 25, 30, 33, 39, 41, 42*, 57, 67, 72, 73, 78, 97, 101, 108, 122*, 124*}) \textbackslash{} Used in section 3*.
\textbackslash{} (Labels in the outer block 4*) \textbackslash{} Used in section 3*.
\textbackslash{} (Load the new font, unless there are problems 62*) \textbackslash{} Used in section 59*.
\textbackslash{} (Make sure that the end of the file is well-formed 105) \textbackslash{} Used in section 103.
\textbackslash{} (Move font name into the \textit{cur\_name} string 66*) \textbackslash{} Used in section 62*.
\textbackslash{} (Move the widths from \textit{in\_width} to \textit{width}, and append \textit{pixel\_width} values 40) \textbackslash{} Used in section 34.
\textbackslash{} (Print all the selected options 56*) \textbackslash{} Used in section 107*.
\textbackslash{} (Process the font definitions of the postamble 106) \textbackslash{} Used in section 103.
\textbackslash{} (Process the preamble 109) \textbackslash{} Used in section 103.
\textbackslash{} (Read and convert the width values, setting up the \textit{in\_width} table 37) \textbackslash{} Used in section 34.
\textbackslash{} (Read past the header data; \textit{goto} 9997 if there is a problem 35) \textbackslash{} Used in section 34.
\textbackslash{} (Read the font parameters into position for font \textit{nf}, and print the font name 61) \textbackslash{} Used in section 59*.
\textbackslash{} (Replace \textit{z} by \textit{z'} and compute \textit{\alpha, \beta} 38) \textbackslash{} Used in section 37.
\textbackslash{} (Set initial values \textit{11, 12, 31, 58, 68, 74, 98}) \textbackslash{} Used in section 3*.
\textbackslash{} (Show the values of \textit{ss, h, v, w, x, y, z, hh, and vv}; then \textit{goto done} 93) \textbackslash{} Used in section 80*.
\textbackslash{} (Skip until finding \textit{eop 96}) \textbackslash{} Used in section 95.
\textbackslash{} (Start translation of command \textit{o} and \textit{goto} the appropriate label to finish the job 81) \textbackslash{} Used in section 80*.
\textbackslash{} (Store character-width indices at the end of the \textit{width} table 36) \textbackslash{} Used in section 34.
\textbackslash{} (Translate a \textit{set\_char} command 88) \textbackslash{} Used in section 81.
\textbackslash{} (Translate an \textit{xxx} command and \textit{goto done} 87) \textbackslash{} Used in section 82.
\textbackslash{} (Translate the next command in the \textit{DVI} file; \textit{goto} 9999 with \textit{do\_page = true} if it was \textit{eop}; \textit{goto} 9998 if premature termination is needed 80*) \textbackslash{} Used in section 79.
\textbackslash{} (Translate up to \textit{max\_pages} pages 111) \textbackslash{} Used in section 107*.
\textbackslash{} (Types in the outer block 8*, 9*, 21) \textbackslash{} Used in section 3*.