

**1\* Introduction.** The GFtoDVI utility program reads binary generic font (“GF”) files that are produced by font compilers such as METAFONT, and converts them into device-independent (“DVI”) files that can be printed to give annotated hardcopy proofs of the character shapes. The annotations are specified by the comparatively simple conventions of plain METAFONT; i.e., there are mechanisms for labeling chosen points and for superimposing horizontal or vertical rules on the enlarged character shapes.

The purpose of GFtoDVI is simply to make proof copies; it does not exhaustively test the validity of a GF file, nor do its algorithms much resemble the algorithms that are typically used to prepare font descriptions for commercial typesetting equipment. Another program, GFtype, is available for validity checking; GFtype also serves as a model of programs that convert fonts from GF format to some other coding scheme.

The *banner* string defined here should be changed whenever GFtoDVI gets modified.

```
define banner ≡ ‘This is GFtoDVI, C Version 3.0’ { printed when the program starts }
```

**3\*** The main input and output files are not mentioned in the program header, because their external names will be determined at run time (e.g., by interpreting the command line that invokes this program). Error messages and other remarks are written on the *output* file, which the user may choose to assign to the terminal if the system permits it.

The term *print* is used instead of *write* when this program writes on the *output* file, so that all such output can be easily deflected.

```
define term_out ≡ stdout
define print(#) ≡ write(term_out, #)
define print_ln(#) ≡ write_ln(term_out, #)
define print_nl(#) ≡ begin write_ln(term_out); write(term_out, #); end
```

```
program GF_to_DVI;
const < Constants in the outer block 5* >
type < Types in the outer block 9 >
var < Globals in the outer block 12 >
procedure initialize; { this procedure gets things started properly }
  var i, j, m, n: integer; { loop indices for initializations }
  begin print_ln(banner);
  < Set initial values 13 >
  end;
```

**4\*** This module deleted, since it only defined the label *final\_end*.

**5\*** The following parameters can be changed at compile time to extend or reduce GFtoDVI’s capacity.

```
define file_name_size ≡ FILENAME_SIZE { a file name shouldn’t be longer than this }
< Constants in the outer block 5* > ≡
  max_labels = 2000; { maximum number of labels and dots and rules per character }
  pool_size = 10000; { maximum total length of labels and other strings }
  max_strings = 1100; { maximum number of labels and other strings }
  terminal_line_length = 150;
  { maximum number of characters input in a single line of input from the terminal }
  font_mem_size = 2000; { space for font metric data }
  dvi_buf_size = 800; { size of the output buffer; must be a multiple of 8 }
  widest_row = 8192; { maximum number of pixels per row }
  lig_lookahead = 20; { size of stack used when inserting ligature characters }
```

This code is used in section 3\*.

**8\*** If the GF file is badly malformed, the whole process must be aborted; GFtoDVI 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 *jump\_out* has been introduced. This procedure, which simply transfers control to the label *final\_end* at the end of the program, contains the only non-local **goto** statement in GFtoDVI.

```
define abort(#) ≡ begin print_ln(#); uexit(1); end
define bad_gf(#) ≡ abort(`Bad_GF_file:`, #, `!(at_byte`, cur_loc - 1 : 1, `)`)
procedure jump_out;
begin uexit(0);
end;
```

**11\*** The original Pascal compiler was designed in the late 60s, when six-bit character sets were common, so it did not make provision for lowercase letters. Nowadays, of course, we need to deal with both capital and small letters in a convenient way. So we shall assume that the Pascal system being used for GFtoDVI has a character set containing at least the standard visible ASCII characters ("!" through "~"). If additional characters are present, GFtoDVI can be configured to work with them too.

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.

```
define text_char  $\equiv$  ASCII_code { the data type of characters in text files }
define first_text_char = 0 { ordinal number of the smallest element of text_char }
define last_text_char = 255 { ordinal number of the largest element of text_char }
```

<Types in the outer block 9> + $\equiv$

```
text_file = packed file of text_char;
```

**14\*** Here now is the system-dependent part of the character set. If GFtoDVI is being implemented on a garden-variety Pascal for which only standard ASCII codes will appear in the input and output files, you don't need to make any changes here. But if you have, for example, an extended character set like the one in Appendix C of *The T<sub>E</sub>Xbook*, the first line of code in this module should be changed to

```
for i  $\leftarrow$  0 to '37 do xchr[i]  $\leftarrow$  chr(i);
```

WEB's character set is essentially identical to T<sub>E</sub>X's.

<Set initial values 13> + $\equiv$

```
for i  $\leftarrow$  1 to '37 do xchr[i]  $\leftarrow$  chr(i);
for i  $\leftarrow$  '177 to '377 do xchr[i]  $\leftarrow$  chr(i);
```

**16\*** The *input\_ln* routine waits for the user to type a line at his or her terminal; then it puts ASCII-code equivalents for the characters on that line into the *buffer* array. The *term\_in* file is used for terminal input.

Since the terminal is being used for both input and output, some systems need a special routine to make sure that the user can see a prompt message before waiting for input based on that message. (Otherwise the message may just be sitting in a hidden buffer somewhere, and the user will have no idea what the program is waiting for.) We shall call a system-dependent subroutine *update\_terminal* in order to avoid this problem.

```
define update_terminal  $\equiv$  flush(stdout) { empty the terminal output buffer }
define term_in  $\equiv$  stdin { standard input }
```

<Globals in the outer block 12> + $\equiv$

```
buffer: array [0 .. terminal_line_length] of 0 .. 255;
```

**17\*** A global variable *line\_length* records the first buffer position after the line just read.

```
procedure input_ln; { inputs a line from the terminal }
```

```
  begin update_terminal;
  if eoln(term_in) then read_ln(term_in);
  line_length  $\leftarrow$  0;
  while (line_length < terminal_line_length)  $\wedge$   $\neg$ eoln(term_in) do
    begin buffer[line_length]  $\leftarrow$  xord[getc(term_in)]; incr(line_length);
    end;
  end;
```

**47\*** To prepare these files for input or output, we *reset* or *rewrite* them. An extension of Pascal is needed, 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)*' and '*rewrite(f,s)*' do this, when *f* is a file variable and *s* is a string variable that specifies the file name.

In C, we use the external *test\_read\_access* procedure, which also does path searching based on the user's environment or the default path. We also read the command line and print the banner here (since we don't want to print the banner if the command line is unreasonable).

```

procedure open_gf_file; { prepares to read packed bytes in gf_file }
  begin if test_read_access(name_of_file, GFFILEPATH) then
    begin reset(gf_file, name_of_file);
    end
  else begin print_pascal_string(name_of_file); abort(':_GF_file_not_found. ');
  end;
  cur_loc ← 0;
end;

procedure open_tfm_file; { prepares to read packed bytes in tfm_file }
  begin if test_read_access(name_of_file, TFMFILEPATH) then
    begin reset(tfm_file, name_of_file);
    end
  else begin print_pascal_string(name_of_file); abort(':_TFM_file_not_found. ');
  end;
end;

procedure open_dvi_file; { prepares to write packed bytes in dvi_file }
  begin rewrite(dvi_file, name_of_file);
end;

```

**51\*** We shall use another set of simple functions to read the next byte or bytes from *gf\_file*. There are four possibilities, each of which is treated as a separate function in order to minimize the overhead for subroutine calls.

```

function get_byte: integer; { returns the next byte, unsigned }
  var b: eight_bits;
  begin if eof(gf_file) then get_byte ← 0
  else begin read(gf_file, b); incr(cur_loc); get_byte ← b;
  end;
end;

function get_two_bytes: integer; { returns the next two bytes, unsigned }
  var a, b: eight_bits;
  begin read(gf_file, a); read(gf_file, b); cur_loc ← cur_loc + 2; get_two_bytes ← a * toint(256) + b;
end;

function get_three_bytes: integer; { returns the next three bytes, unsigned }
  var a, b, c: eight_bits;
  begin read(gf_file, a); read(gf_file, b); read(gf_file, c); cur_loc ← cur_loc + 3;
  get_three_bytes ← (a * toint(256) + b) * 256 + c;
end;

function signed_quad: integer; { returns the next four bytes, signed }
  var a, b, c, d: eight_bits;
  begin read(gf_file, a); read(gf_file, b); read(gf_file, c); read(gf_file, d); cur_loc ← cur_loc + 4;
  if a < 128 then signed_quad ← ((a * toint(256) + b) * 256 + c) * 256 + d
  else signed_quad ← (((a - 256) * toint(256) + b) * 256 + c) * 256 + d;
end;

```

**52\* Reading the font information.** Now let's get down to brass tacks and consider the more substantial routines that actually convert TFM data into a form suitable for computation. The routines in this part of the program have been borrowed from T<sub>E</sub>X, with slight changes, since GFtoDVI has to do some of the things that T<sub>E</sub>X does.

The TFM data is stored in a large array called *font\_info*. Each item of *font\_info* is a *memory\_word*; the *fix\_word* data gets converted into *scaled* entries, while everything else goes into words of type *four\_quarters*. (These data structures are special cases of the more general memory words of T<sub>E</sub>X. On some machines it is necessary to define *min\_quarterword* = -128 and *max\_quarterword* = 127 in order to pack four quarterwords into a single word.)

```

define min_quarterword = 0 { change this to allow efficient packing, if necessary }
define max_quarterword = 255 { ditto }
define qi(#) ≡ # + min_quarterword { to put an eight_bits item into a quarterword }
define qo(#) ≡ # - min_quarterword { to take an eight_bits item out of a quarterword }
define title_font = 1
define label_font = 2
define gray_font = 3
define slant_font = 4
define logo_font = 5
define non_char ≡ qi(256)
define non_address ≡ font_mem_size
⟨Types in the outer block 9⟩ +≡
font_index = 0 .. font_mem_size; quarterword = min_quarterword .. max_quarterword; { 1/4 of a word }
four_quarters = packed record B0: quarterword;
    B1: quarterword;
    B2: quarterword;
    B3: quarterword;
end;
@\\
#include_"memory.h";@\\ { note the ; so web2c will translate types that come after this }
internal_font_number = title_font .. logo_font;

```

**55\*** Of course we want to define macros that suppress the detail of how font information is actually packed, so that we don't have to write things like

$$font\_info[width\_base[f] + font\_info[char\_base[f] + c].qqqq.b0].sc$$

too often. The WEB definitions here make  $char\_info(f)(c)$  the *four-quarters* word of font information corresponding to character  $c$  of font  $f$ . If  $q$  is such a word,  $char\_width(f)(q)$  will be the character's width; hence the long formula above is at least abbreviated to

$$char\_width(f)(char\_info(f)(c)).$$

In practice we will try to fetch  $q$  first and look at several of its fields at the same time.

The italic correction of a character will be denoted by  $char\_italic(f)(q)$ , so it is analogous to  $char\_width$ . But we will get at the height and depth in a slightly different way, since we usually want to compute both height and depth if we want either one. The value of  $height\_depth(q)$  will be the 8-bit quantity

$$b = height\_index \times 16 + depth\_index,$$

and if  $b$  is such a byte we will write  $char\_height(f)(b)$  and  $char\_depth(f)(b)$  for the height and depth of the character  $c$  for which  $q = char\_info(f)(c)$ . Got that?

The tag field will be called  $char\_tag(q)$ ; and the remainder byte will be called  $rem\_byte(q)$ .

```

define char_info_end(#)  $\equiv$  # ] .qqqq
define char_info(#)  $\equiv$  font_info [ char_base[#] + char_info_end
define char_width_end(#)  $\equiv$  #.B0 ] .sc
define char_width(#)  $\equiv$  font_info [ width_base[#] + char_width_end
define char_exists(#)  $\equiv$  (#.B0 > min_quarterword)
define char_italic_end(#)  $\equiv$  (qo(#.B2)) div 4 ] .sc
define char_italic(#)  $\equiv$  font_info [ italic_base[#] + char_italic_end
define height_depth(#)  $\equiv$  qo(#.B1)
define char_height_end(#)  $\equiv$  (#) div 16 ] .sc
define char_height(#)  $\equiv$  font_info [ height_base[#] + char_height_end
define char_depth_end(#)  $\equiv$  # mod 16 ] .sc
define char_depth(#)  $\equiv$  font_info [ depth_base[#] + char_depth_end
define char_tag(#)  $\equiv$  ((qo(#.B2)) mod 4)
define skip_byte(#)  $\equiv$  qo(#.B0)
define next_char(#)  $\equiv$  #.B1
define op_byte(#)  $\equiv$  qo(#.B2)
define rem_byte(#)  $\equiv$  #.B3

```

```

60* define read_two_halves_end(#)  $\equiv$  #  $\leftarrow$  b2 * toint(256) + b3
define read_two_halves(#)  $\equiv$  read_tfm_word; #  $\leftarrow$  b0 * toint(256) + b1; read_two_halves_end

```

$\langle$  Read the TFM size fields 60\*  $\equiv$

```

begin read_two_halves(lf)(lh); read_two_halves(bc)(ec);
if (bc > ec + 1)  $\vee$  (ec > 255) then abend;
if bc > 255 then { bc = 256 and ec = 255 }
begin bc  $\leftarrow$  1; ec  $\leftarrow$  0;
end;
read_two_halves(nw)(nh); read_two_halves(nd)(ni); read_two_halves(nl)(nk); read_two_halves(ne)(np);
if lf  $\neq$  6 + lh + (ec - bc + 1) + nw + nh + nd + ni + nl + nk + ne + np then abend;
end

```

This code is used in section 59.

**62\*** Only the first two words of the header are needed by GFtoDVI.

```
define store_four_quarters(#) ≡
    begin read_tfm_word; qw.B0 ← qi(b0); qw.B1 ← qi(b1); qw.B2 ← qi(b2); qw.B3 ← qi(b3);
    # ← qw;
end
```

⟨Read the TFM header 62\*⟩ ≡

```
begin if lh < 2 then abend;
store_four_quarters(font_check[f]); read_tfm_word;
if b0 > 127 then abend; { design size must be positive }
z ← ((b0 * toint(256) + b1) * toint(256) + b2) * 16 + (b3 div 16);
if z < unity then abend;
while lh > 2 do
    begin read_tfm_word; decr(lh); { ignore the rest of the header }
    end;
font_dsize[f] ← z;
if s > 0 then z ← s;
font_size[f] ← z;
end
```

This code is used in section 59.

```
66* define check_byte_range(#) ≡
    begin if (# < bc) ∨ (# > ec) then abend
    end
```

⟨Read ligature/kern program 66\*⟩ ≡

```
begin bch_label ← '777777'; bchar ← 256;
if nl > 0 then
    begin for k ← lig_kern_base[f] to kern_base[f] - 1 do
        begin store_four_quarters(font_info[k].qqqq);
        if b0 > stop_flag then
            begin if 256 * b2 + b3 ≥ nl then abend;
            if b0 = 255 then
                if k = lig_kern_base[f] then bchar ← b1;
            end
        else begin if b1 ≠ bchar then check_byte_range(b1);
            if b2 < kern_flag then check_byte_range(b3)
            else if toint(256) * (b2 - 128) + b3 ≥ nk then abend;
            end;
        end;
    if b0 = 255 then bch_label ← 256 * b2 + b3;
    end;
for k ← kern_base[f] to exten_base[f] - 1 do store_scaled(font_info[k].sc);
end
```

This code is used in section 59.

**88\*** Font metric files whose areas are not given explicitly are assumed to appear in a standard system area called *home\_font\_area*. This system area name will, of course, vary from place to place. Under the Berkeley UNIX version, we set *home\_font\_area* to *null\_string* because the default areas to search for TFM files are built into the routine *test\_read\_access*.

```
⟨Initialize the strings 77⟩ +=
  l ← 0; init_str0(home_font_area);
```

**90\*** And here's the second.

```
function more_name(c : ASCII_code): boolean;
  begin if c = "␣" then more_name ← false
  else begin if (c = "/" ) then
    begin area_delimiter ← pool_ptr; ext_delimiter ← 0;
    end
    else if c = "." then ext_delimiter ← pool_ptr;
    str_room(1); append_char(c); { contribute c to the current string }
    more_name ← true;
    end;
  end;
```

**94\*** The *start\_gf* procedure obtains the name of the generic font file to be input from the command line. It opens the file, making sure that some input is present; then it opens the output file.

```
procedure start_gf;
  label done;
  var arg_buffer: packed array [1 .. FILENAMESIZE] of char;
  arg_buf_ptr: 1 .. FILENAMESIZE;
  begin if (argc < 1) ∨ (argc > 2) then abort('Usage: ␣gftodvi␣<gf␣file>.');
  if argc = 1 then
    begin print_nl('GF␣file␣name:␣'); input_ln;
    end
  else begin argv(1, arg_buffer); arg_buffer[FILENAMESIZE] ← '␣'; arg_buf_ptr ← 1; line_length ← 0;
    while (arg_buf_ptr < FILENAMESIZE) ∧ (arg_buffer[arg_buf_ptr] = '␣') do incr(arg_buf_ptr);
    while (arg_buf_ptr < FILENAMESIZE) ∧ (line_length < terminal_line_length) ∧ (arg_buffer[arg_buf_ptr] ≠
      '␣') do
      begin buffer[line_length] ← xord[arg_buffer[arg_buf_ptr]]; incr(line_length); incr(arg_buf_ptr);
      end;
    end;
    buf_ptr ← 0; buffer[line_length] ← "?";
    while buffer[buf_ptr] = "␣" do incr(buf_ptr);
    if buf_ptr < line_length then
      begin ⟨Scan the file name in the buffer 95⟩;
      if cur_ext = null_string then cur_ext ← gf_ext;
      pack_file_name(cur_name, cur_area, cur_ext); open_gf_file;
      end;
    job_name ← cur_name; pack_file_name(job_name, null_string, dvi_ext); open_dvi_file;
    end;
```



**107\*** The actual output of *dvi\_buf*[*a* .. *b*] to *dvi\_file* is performed by calling *write\_dvi*(*a*, *b*). It is safe to assume that *a* and *b* + 1 will both be multiples of 4 when *write\_dvi*(*a*, *b*) is called; therefore it is possible on many machines to use efficient methods to pack four bytes per word and to output an array of words with one system call.

In C, we can write out the entire array with one call.

```
procedure write_dvi (a, b : dvi_index);
begin write_chunk (dvi_file, dvi_buf, a, b);
end;
```

**111\*** Here's a procedure that outputs a font definition.

```
define select_font(#) ≡ dvi_out(fnt_num_0 + #) { set current font to # }
procedure dvi_font_def (f : internal_font_number);
var k: integer; { index into str_pool }
begin dvi_out(fnt_def1); dvi_out(f);
dvi_out(qo(font_check[f].B0)); dvi_out(qo(font_check[f].B1)); dvi_out(qo(font_check[f].B2));
dvi_out(qo(font_check[f].B3));
dvi_four(font_size[f]); dvi_four(font_dsize[f]);
dvi_out(length(font_area[f])); dvi_out(length(font_name[f]));
⟨Output the font name whose internal number is f 112⟩
end;
```

⟨Declare the procedure called *load\_fonts* 98⟩

**115\*** At the end of the program, we must finish things off by writing the postamble. An integer variable *k* will be declared for use by this routine.

```
⟨Finish the DVI file and goto final_end 115*⟩ ≡
begin dvi_out(post); { beginning of the postamble }
dvi_four(last_bop); last_bop ← dvi_offset + dvi_ptr - 5; { post location }
dvi_four(25400000); dvi_four(473628672); { conversion ratio for sp }
dvi_four(1000); { magnification factor }
dvi_four(max_v); dvi_four(max_h);
dvi_out(0); dvi_out(3); { 'max_push' is said to be 3 }
dvi_out(total_pages div 256); dvi_out(total_pages mod 256);
if ¬fonts_not_loaded then
  for k ← title_font to logo_font do
    if length(font_name[k]) > 0 then dvi_font_def(k);
  dvi_out(post_post); dvi_four(last_bop); dvi_out(dvi_id_byte);
  k ← 4 + ((dvi_buf_size - dvi_ptr) mod 4); { the number of 223's }
  while k > 0 do
    begin dvi_out(223); decr(k);
    end;
  ⟨Empty the last bytes out of dvi_buf 109⟩;
  uexit(0);
end
```

This code is used in section 219\*.

**118\*** ⟨Set initial values 13⟩ +≡

*dummy\_info.B0* ← *qi*(0); *dummy\_info.B1* ← *qi*(0); *dummy\_info.B2* ← *qi*(0); *dummy\_info.B3* ← *qi*(0);

**138\*** The following error message is given when an absent slant has been requested.

```
procedure slant_complaint(r : real);  
  begin if abs(r - slant_reported) > 0.001 then  
    begin print_nl('Sorry, I can't make diagonal rules of slant'); print_real(r, 10, 5);  
    print('!'); slant_reported ← r;  
    end;  
  end;
```

**164\*** The process of ferreting everything away comes to an abrupt halt when a *boc* command is sensed. The following steps are performed at such times:

```
<Process a character 164*> ≡  
  begin check_fonts; <Finish reading the parameters of the boc 165>;  
  <Get ready to convert METAFONT coordinates to DVI coordinates 170>;  
  <Output the bop and the title line 172>;  
  <Output all rules for the current character 173>;  
  <Output all labels for the current character 181>;  
  do_pixels; dvi_out(eop); { finish the page }  
  <Adjust the maximum page width 203>;  
  end
```

This code is used in section 219\*.

```

215* define do_skip  $\equiv z \leftarrow 0$ ; paint_black  $\leftarrow$  false
define end_with(#)  $\equiv$ 
    begin #; cur_gf  $\leftarrow$  get_byte; goto done1; end
define five_cases(#)  $\equiv$  #, # + 1, # + 2, # + 3, # + 4
define eight_cases(#)  $\equiv$  #, # + 1, # + 2, # + 3, # + 4, # + 5, # + 6, # + 7
define thirty_two_cases(#)  $\equiv$  eight_cases(#), eight_cases(# + 8), eight_cases(# + 16), eight_cases(# + 24)
define sixty_four_cases(#)  $\equiv$  thirty_two_cases(#), thirty_two_cases(# + 32)
⟨Read and process GF commands until coming to the end of this row 215*⟩  $\equiv$ 
loop begin continue: if (cur_gf  $\geq$  new_row_0)  $\wedge$  (cur_gf  $\leq$  new_row_0 + 164) then
    end_with(z  $\leftarrow$  cur_gf - new_row_0; paint_black  $\leftarrow$  true)
else case cur_gf of
    sixty_four_cases(0): k  $\leftarrow$  cur_gf;
    paint1: k  $\leftarrow$  get_byte;
    paint2: k  $\leftarrow$  get_two_bytes;
    paint3: k  $\leftarrow$  get_three_bytes;
    eoc: goto done1;
    skip0: end_with(blank_rows  $\leftarrow$  0; do_skip);
    skip1: end_with(blank_rows  $\leftarrow$  get_byte; do_skip);
    skip2: end_with(blank_rows  $\leftarrow$  get_two_bytes; do_skip);
    skip3: end_with(blank_rows  $\leftarrow$  get_three_bytes; do_skip);
    xxx1, xxx2, xxx3, xxx4, yyy, no_op: begin skip_nop; goto continue;
    end;
    othercases bad_gf(`Improper_opcode`)
endcases;
    ⟨Paint k bits and read another command 216⟩;
end;
done1:

```

This code is used in section 214.

**219\*** **The main program.** Now we are ready to put it all together. This is where GFtoDVI starts, and where it ends.

```

begin initialize; { get all variables initialized }
< Initialize the strings 77>; { initialize paths from environment variables }
set_paths(GF_FILE_PATH_BIT + TFM_FILE_PATH_BIT); start_gf; { open the input and output files }
< Process the preamble 221>;
cur_gf ← get_byte; init_str_ptr ← str_ptr;
loop begin < Initialize variables for the next character 144>;
  while (cur_gf ≥ xxx1) ∧ (cur_gf ≤ no_op) do < Process a no-op command 154>;
  if cur_gf = post then < Finish the DVI file and goto final_end 115*>;
  if cur_gf ≠ boc then
    if cur_gf ≠ boc1 then abort(`Missing␣boc!`);
  < Process a character 164*>;
  cur_gf ← get_byte; str_ptr ← init_str_ptr; pool_ptr ← str_start[str_ptr];
  end;
print_ln(`␣`);
end.

```

**223\* Index.** Here is a list of the section numbers where each identifier is used. Cross references to error messages and a few other tidbits of information also appear.

The following sections were changed by the change file: 1, 3, 4, 5, 8, 11, 14, 16, 17, 47, 51, 52, 55, 60, 62, 66, 88, 90, 94, 107, 111, 115, 118, 138, 164, 215, 219, 223.

*a*: 51\* 92, 107\* 211.  
*abend*: 58, 60\* 62\* 63, 64, 66\*  
*abort*: 8\* 47\* 58, 61, 73, 74, 75, 91, 94\* 141,  
 165, 169, 184, 219\*  
*abs*: 138\* 151, 152, 173, 178.  
*adjust*: 69.  
*alpha*: 58, 64, 65.  
*append\_char*: 73, 75, 83, 90\* 101, 221.  
*append\_to\_name*: 92.  
*area\_code*: 77, 98, 100, 101, 154.  
*area\_delimiter*: 87, 89, 90\* 91.  
*arg\_buf\_ptr*: 94\*  
*arg\_buffer*: 94\*  
*argc*: 94\*  
*argv*: 94\*  
*ASCII\_code*: 10, 11\* 12, 71, 73, 90\* 92, 116.  
 at size: 39.  
*at\_code*: 77, 154.  
*b*: 51\* 107\* 207.  
 backpointers: 32.  
 Bad GF file: 8\*  
 Bad label type...: 163.  
 Bad TFM file...: 58.  
*bad\_gf*: 8\* 215\* 221.  
*bad\_tfm*: 58.  
*banner*: 1\* 3\*  
*bc*: 37, 38, 40, 42, 58, 60\* 61, 66\* 69.  
*bch\_label*: 58, 66\* 69.  
*bchar*: 58, 66\* 69, 116, 122.  
*bchar\_label*: 53, 69, 120.  
*begin\_name*: 86, 89, 95.  
*best\_q*: 150, 151, 152.  
*beta*: 58, 64, 65.  
 BigEndian order: 19, 37.  
*black*: 28, 29.  
*blank\_rows*: 212, 214, 215\* 217, 218.  
*boc*: 27, 29, 30, 31, 32, 35, 85, 96, 153, 154,  
 164\* 165, 219\*  
*boc1*: 29, 30, 165, 219\*  
*boolean*: 90\* 96, 116, 117, 145, 149, 194, 218, 220.  
*bop*: 19, 21, 22, 24, 25, 102, 172.  
*bot*: 43.  
*bot\_coords*: 186, 190, 196, 197, 198, 199.  
*box\_depth*: 116, 117, 121, 185, 186.  
*box\_height*: 116, 117, 121, 185, 186.  
*box\_width*: 116, 117, 119, 121, 185, 186.  
*buf\_ptr*: 18, 94\* 95, 100, 101.  
*buffer*: 16\* 17\* 18, 75, 82, 83, 94\* 95, 100, 101, 114.  
*byte\_file*: 45, 46.  
*b0*: 49, 50, 53, 55\* 60\* 62\* 63, 64, 66\* 67, 68.  
*B0*: 52\* 55\* 62\* 111\* 118\*  
*b1*: 49, 50, 60\* 62\* 63, 64, 66\* 67, 68.  
*B1*: 52\* 55\* 62\* 111\* 118\*  
*b2*: 49, 50, 60\* 62\* 63, 64, 66\* 67, 68.  
*B2*: 52\* 55\* 62\* 111\* 118\*  
*b3*: 49, 50, 60\* 62\* 63, 64, 66\* 67, 68.  
*B3*: 52\* 55\* 62\* 111\* 118\*  
*c*: 51\* 75, 81, 90\* 92, 113, 127.  
*char*: 11\* 48, 94\*  
*char\_base*: 53, 55\* 61.  
*char\_code*: 165, 166, 172.  
*char\_depth*: 55\* 121.  
*char\_depth\_end*: 55\*  
*char\_exists*: 55\* 121, 169, 184, 205.  
*char\_header*: 78, 172.  
*char\_height*: 55\* 121, 137, 169, 184.  
*char\_height\_end*: 55\*  
*char\_info*: 40, 53, 55\* 117, 120, 121, 137, 169,  
 184, 205, 210.  
*char\_info\_end*: 55\*  
*char\_info\_word*: 38, 40, 41.  
*char\_italic*: 55\*  
*char\_italic\_end*: 55\*  
*char\_kern*: 56, 120.  
*char\_kern\_end*: 56.  
*char\_loc*: 29, 32.  
*char\_loc0*: 29.  
*char\_tag*: 55\* 120, 210.  
*char\_width*: 55\* 121, 169, 184.  
*char\_width\_end*: 55\*  
 Character too wide: 165.  
 check sum: 24, 31, 39.  
*check\_byte\_range*: 66\* 67.  
*check\_fonts*: 96, 164\*  
 Chinese characters: 32.  
*chr*: 11\* 12, 14\* 15.  
 coding scheme: 39.  
*continue*: 6, 98, 99, 116, 122, 123, 215\* 218.  
*convert*: 167, 168, 173, 187.  
*cs*: 31.  
*cur\_area*: 86, 91, 94\*  
*cur\_ext*: 86, 91, 94\*  
*cur\_gf*: 79, 80, 81, 82, 84, 85, 154, 165, 214,  
 215\* 216, 217, 218, 219\*  
*cur\_l*: 116, 120, 121, 122, 123.  
*cur\_loc*: 8\* 47\* 48, 51\* 154, 163.

- cur\_name*: [86](#), [91](#), [94](#)\*  
*cur\_r*: [116](#), [120](#), [122](#), [123](#).  
*cur\_string*: [79](#), [80](#), [81](#), [83](#), [154](#), [162](#), [163](#).  
*d*: [51](#)\* [127](#), [150](#).  
*d\_min*: [150](#), [151](#), [152](#).  
*decr*: [7](#), [62](#)\* [69](#), [75](#), [95](#), [114](#), [115](#)\* [122](#), [179](#), [180](#),  
[210](#), [213](#), [214](#), [217](#).  
default fonts: [78](#).  
*default\_gray\_font*: [78](#), [97](#).  
*default\_label\_font*: [78](#), [97](#).  
*default\_rule\_thickness*: [44](#), [57](#), [135](#), [175](#).  
*default\_title\_font*: [78](#), [97](#).  
*deLm*: [29](#).  
*deLn*: [29](#).  
*delta*: [183](#), [184](#), [185](#), [186](#).  
*delta\_x*: [167](#), [168](#), [170](#), [209](#), [218](#).  
*delta\_y*: [167](#), [168](#), [170](#), [218](#).  
*den*: [21](#), [23](#), [25](#).  
*depth\_base*: [53](#), [55](#)\* [61](#), [64](#).  
*depth\_index*: [40](#), [55](#)\*  
design size: [31](#), [39](#).  
*dfl*: [194](#), [195](#), [196](#), [197](#), [198](#), [199](#).  
*dm*: [29](#).  
*do\_nothing*: [7](#), [63](#), [154](#).  
*do\_pixels*: [164](#)\* [218](#).  
*do\_skip*: [215](#)\*  
*done*: [6](#), [58](#), [69](#), [81](#), [83](#), [85](#), [94](#)\* [95](#), [98](#), [99](#), [116](#),  
[120](#), [122](#), [208](#), [218](#).  
*done1*: [6](#), [81](#), [82](#), [215](#)\* [218](#).  
*dot\_for\_label*: [188](#), [190](#), [192](#), [193](#), [194](#), [195](#), [202](#).  
*dot\_height*: [148](#), [183](#), [184](#), [185](#), [186](#), [188](#).  
*dot\_width*: [148](#), [183](#), [184](#), [186](#), [188](#).  
*down1*: [21](#).  
*down2*: [21](#).  
*down3*: [21](#).  
*down4*: [21](#), [22](#), [171](#).  
*ds*: [31](#).  
*dummy\_info*: [117](#), [118](#)\* [120](#).  
DVI files: [19](#).  
*dvi\_buf*: [104](#), [105](#), [107](#)\* [108](#).  
*dvi\_buf\_size*: [5](#)\* [104](#), [105](#), [106](#), [108](#), [109](#), [115](#)\*  
*dvi\_ext*: [78](#), [94](#)\*  
*dvi\_file*: [46](#), [47](#)\* [107](#)\*  
*dvi\_font\_def*: [98](#), [111](#)\* [115](#)\*  
*dvi\_four*: [110](#), [111](#)\* [115](#)\* [119](#), [121](#), [171](#), [172](#), [176](#),  
[177](#), [179](#), [180](#), [209](#), [221](#).  
*dvi\_goto*: [171](#), [172](#), [176](#), [177](#), [178](#), [188](#), [190](#),  
[194](#), [202](#), [218](#).  
*dvi\_id\_byte*: [23](#), [115](#)\* [221](#).  
*dvi\_index*: [104](#), [105](#), [107](#)\*  
*dvi\_limit*: [104](#), [105](#), [106](#), [108](#), [109](#).  
*dvi\_offset*: [104](#), [105](#), [106](#), [108](#), [115](#)\* [172](#).  
*dvi\_out*: [108](#), [110](#), [111](#)\* [112](#), [113](#), [114](#), [115](#)\* [119](#),  
[121](#), [164](#)\* [171](#), [172](#), [176](#), [177](#), [178](#), [179](#), [180](#), [188](#),  
[190](#), [194](#), [202](#), [208](#), [209](#), [218](#), [221](#).  
*dvi\_ptr*: [104](#), [105](#), [106](#), [108](#), [109](#), [115](#)\* [172](#).  
*dvi\_scaled*: [114](#), [172](#), [202](#).  
*dvi\_swap*: [108](#).  
*dvi\_x*: [167](#), [168](#), [173](#), [176](#), [177](#), [178](#), [185](#), [186](#),  
[187](#), [188](#), [190](#), [194](#), [195](#).  
*dvi\_y*: [167](#), [168](#), [173](#), [176](#), [177](#), [178](#), [185](#), [186](#),  
[187](#), [188](#), [190](#), [194](#), [195](#).  
*dx*: [29](#), [32](#), [192](#), [220](#).  
*dy*: [29](#), [32](#), [179](#), [180](#), [192](#), [220](#).  
*d0*: [148](#), [150](#), [151](#), [152](#).  
*e*: [92](#).  
*ec*: [37](#), [38](#), [40](#), [42](#), [58](#), [60](#)\* [61](#), [66](#)\* [69](#).  
*eight\_bits*: [45](#), [49](#), [51](#)\* [52](#)\* [53](#), [80](#), [105](#), [113](#), [116](#), [218](#).  
*eight\_cases*: [215](#)\*  
*eighth\_octant*: [192](#), [196](#), [197](#), [198](#), [199](#).  
**else**: [2](#).  
**end**: [2](#).  
*end\_k*: [116](#), [122](#), [123](#).  
*end\_name*: [86](#), [91](#), [95](#).  
*end\_of\_list*: [142](#), [144](#), [145](#), [160](#), [161](#), [189](#), [193](#), [201](#).  
*end\_with*: [215](#)\*  
**endcases**: [2](#).  
*eoc*: [27](#), [29](#), [30](#), [31](#), [81](#), [85](#), [214](#), [215](#)\* [217](#).  
*eof*: [51](#)\*  
*eoln*: [17](#)\*  
*eop*: [19](#), [21](#), [22](#), [24](#), [164](#)\*  
*equals\_sign*: [78](#), [202](#).  
*exit*: [6](#), [7](#), [145](#), [194](#), [218](#).  
*ext*: [165](#), [166](#), [172](#).  
*ext\_delimiter*: [87](#), [89](#), [90](#)\* [91](#).  
*ext\_header*: [78](#), [172](#).  
*ext\_tag*: [41](#), [63](#).  
*exten*: [41](#).  
*exten\_base*: [53](#), [61](#), [66](#)\* [67](#), [69](#).  
*extensible\_recipe*: [38](#), [43](#).  
*extra\_space*: [44](#).  
*f*: [58](#), [98](#), [111](#)\* [116](#).  
*false*: [90](#)\* [97](#), [98](#), [116](#), [120](#), [145](#), [150](#), [190](#), [194](#),  
[195](#), [215](#)\* [218](#), [221](#).  
*fifth\_octant*: [192](#), [196](#), [197](#), [198](#), [199](#).  
*file\_name\_size*: [5](#)\* [48](#), [92](#).  
*FILENAME\_SIZE*: [5](#)\* [94](#)\*  
*final\_end*: [4](#)\* [8](#)\*  
*finishing\_col*: [208](#), [216](#), [217](#), [218](#).  
*first\_dot*: [148](#), [149](#), [161](#), [187](#), [201](#).  
*first\_octant*: [192](#), [196](#), [197](#), [198](#), [199](#).  
*first\_string*: [75](#), [76](#).  
*first\_text\_char*: [11](#)\* [15](#).  
*five\_cases*: [215](#)\*



- fix\_word*: [38](#), [39](#), [44](#), [52\\*](#) [64](#).  
*flush*: [16\\*](#)  
*fnem\_ptr*: [53](#), [54](#), [61](#), [63](#), [69](#).  
*font\_def1*: [21](#), [22](#), [111\\*](#)  
*font\_def2*: [21](#).  
*font\_def3*: [21](#).  
*font\_def4*: [21](#).  
*font\_num\_0*: [21](#), [22](#), [111\\*](#)  
*font\_num\_1*: [21](#).  
*font\_num\_63*: [21](#).  
*font1*: [21](#).  
*font2*: [21](#).  
*font3*: [21](#).  
*font4*: [21](#).  
*font\_area*: [96](#), [97](#), [98](#), [101](#), [111\\*](#) [112](#), [154](#).  
*font\_at*: [96](#), [97](#), [98](#), [101](#), [154](#).  
*font\_bc*: [53](#), [69](#), [120](#), [205](#).  
*font\_bchar*: [53](#), [69](#), [116](#).  
*font\_change*: [154](#).  
*font\_check*: [53](#), [62\\*](#) [111\\*](#)  
*font\_dsize*: [53](#), [62\\*](#) [111\\*](#)  
*font\_ec*: [53](#), [69](#), [120](#), [137](#), [205](#).  
*font\_index*: [52\\*](#) [53](#), [58](#), [116](#).  
*font\_info*: [52\\*](#) [53](#), [55\\*](#) [56](#), [57](#), [58](#), [61](#), [63](#), [64](#),  
[66\\*](#) [67](#), [68](#), [120](#).  
*font\_mem\_size*: [5\\*](#) [52\\*](#) [61](#).  
*font\_name*: [96](#), [97](#), [98](#), [101](#), [111\\*](#) [112](#), [115\\*](#) [137](#), [154](#).  
*font\_size*: [53](#), [62\\*](#) [111\\*](#)  
*fonts\_not\_loaded*: [96](#), [97](#), [98](#), [115\\*](#) [154](#).  
*found*: [6](#), [98](#), [99](#), [100](#), [194](#), [196](#), [197](#), [198](#), [199](#).  
*four\_quarters*: [52\\*](#) [53](#), [55\\*](#) [58](#), [98](#), [116](#), [117](#), [218](#).  
*fourth\_octant*: [192](#), [196](#), [197](#), [198](#), [199](#).  
Fuchs, David Raymond: [19](#), [26](#), [33](#).  
*fudge\_factor*: [168](#), [169](#), [202](#).  
*get\_avail*: [141](#), [159](#), [162](#), [163](#), [188](#).  
*get\_byte*: [51\\*](#) [81](#), [82](#), [83](#), [84](#), [85](#), [165](#), [215\\*](#) [216](#),  
[218](#), [219\\*](#) [221](#).  
*get\_three\_bytes*: [51\\*](#) [81](#), [85](#), [215\\*](#)  
*get\_two\_bytes*: [51\\*](#) [81](#), [85](#), [215\\*](#)  
*get\_yyy*: [84](#), [154](#), [157](#), [159](#), [163](#).  
*getc*: [17\\*](#)  
GF file name:: [94\\*](#)  
*gf\_ext*: [78](#), [94\\*](#)  
*gf\_file*: [46](#), [47\\*](#) [48](#), [51\\*](#) [80](#).  
GF\_FILE\_PATH\_BIT: [219\\*](#)  
*gf\_id\_byte*: [29](#), [221](#).  
GF\_to\_DVI: [3\\*](#)  
GFFILEPATH: [47\\*](#)  
gray fonts: [35](#), [39](#), [124](#).  
*gray\_font*: [52\\*](#) [58](#), [77](#), [78](#), [97](#), [154](#), [169](#), [175](#), [181](#),  
[184](#), [205](#), [210](#), [218](#).  
*gray\_rule\_thickness*: [173](#), [174](#), [175](#).  
*half\_buf*: [104](#), [105](#), [106](#), [108](#), [109](#).  
*half\_x\_height*: [183](#), [184](#), [186](#).  
*hbox*: [116](#), [117](#), [172](#), [183](#), [185](#), [190](#), [194](#), [202](#).  
*hd*: [116](#), [121](#).  
*header*: [39](#).  
*height\_base*: [53](#), [55\\*](#) [61](#), [64](#).  
*height\_depth*: [55\\*](#) [121](#), [137](#), [169](#), [184](#).  
*height\_index*: [40](#), [55\\*](#)  
*home\_font\_area*: [78](#), [88\\*](#) [98](#).  
*hppp*: [31](#).  
*i*: [3\\*](#) [23](#), [98](#), [116](#), [218](#).  
*incr*: [7](#), [17\\*](#) [51\\*](#) [73](#), [74](#), [75](#), [82](#), [83](#), [91](#), [92](#), [94\\*](#) [95](#),  
[100](#), [101](#), [108](#), [114](#), [116](#), [119](#), [122](#), [123](#), [128](#), [129](#),  
[141](#), [172](#), [180](#), [192](#), [202](#), [208](#), [217](#), [221](#).  
*info*: [139](#), [140](#), [148](#), [162](#), [163](#), [172](#), [188](#), [190](#),  
[194](#), [202](#).  
*init\_str\_ptr*: [71](#), [101](#), [154](#), [219\\*](#)  
*init\_str0*: [75](#), [77](#), [88\\*](#)  
*init\_str1*: [75](#).  
*init\_str10*: [75](#), [77](#).  
*init\_str11*: [75](#), [77](#).  
*init\_str12*: [75](#), [77](#), [78](#).  
*init\_str13*: [75](#), [77](#).  
*init\_str2*: [75](#), [78](#).  
*init\_str3*: [75](#), [78](#).  
*init\_str4*: [75](#), [77](#), [78](#).  
*init\_str5*: [75](#), [77](#), [78](#).  
*init\_str6*: [75](#), [77](#), [78](#).  
*init\_str7*: [75](#), [77](#), [78](#).  
*init\_str8*: [75](#), [77](#), [78](#).  
*init\_str9*: [75](#), [77](#).  
*initialize*: [3\\*](#) [219\\*](#)  
*input\_ln*: [16\\*](#) [17\\*](#) [18](#), [94\\*](#) [99](#).  
*integer*: [3\\*](#) [9](#), [45](#), [48](#), [51\\*](#) [53](#), [58](#), [75](#), [76](#), [81](#), [85](#),  
[92](#), [98](#), [102](#), [105](#), [110](#), [111\\*](#) [114](#), [134](#), [166](#),  
[182](#), [212](#), [218](#), [220](#).  
*interaction*: [95](#), [96](#), [97](#), [98](#).  
*internal\_font\_number*: [52\\*](#) [53](#), [96](#), [98](#), [111\\*](#) [116](#).  
*interpret\_xxx*: [79](#), [81](#), [154](#).  
*italic\_base*: [53](#), [55\\*](#) [61](#), [64](#).  
*italic\_index*: [40](#).  
*j*: [3\\*](#) [81](#), [85](#), [92](#), [98](#), [116](#), [218](#).  
Japanese characters: [32](#).  
*job\_name*: [93](#), [94\\*](#)  
*jump\_out*: [8\\*](#)  
*k*: [23](#), [58](#), [81](#), [85](#), [92](#), [98](#), [111\\*](#) [114](#), [116](#), [220](#).  
*kern*: [42](#).  
*kern\_amount*: [116](#), [120](#), [121](#).  
*kern\_base*: [53](#), [56](#), [61](#), [66\\*](#) [69](#), [120](#).  
*kern\_flag*: [42](#), [66\\*](#) [120](#).  
*keyword\_code*: [79](#), [81](#).  
*l*: [76](#), [81](#), [116](#), [218](#).

- lab\_typ*: [160](#), [163](#), [181](#), [187](#), [189](#), [190](#), [191](#), [193](#).  
*label\_font*: [52\\*](#), [58](#), [77](#), [78](#), [97](#), [154](#), [181](#), [184](#),  
[190](#), [194](#), [202](#).  
*label\_for\_dot*: [188](#), [193](#), [201](#), [202](#).  
*label\_head*: [160](#), [161](#), [181](#), [187](#), [189](#), [191](#), [193](#), [200](#).  
*label\_tail*: [160](#), [161](#), [163](#), [181](#).  
*label\_type*: [79](#), [80](#), [83](#), [163](#).  
*last\_bop*: [102](#), [103](#), [115\\*](#), [172](#).  
*last\_text\_char*: [11\\*](#), [15](#).  
*left\_coords*: [186](#), [190](#), [196](#), [197](#), [198](#), [199](#).  
*left\_quotes*: [78](#), [172](#).  
*length*: [72](#), [83](#), [98](#), [100](#), [111\\*](#), [115\\*](#), [137](#).  
*lf*: [37](#), [58](#), [60\\*](#), [61](#), [69](#).  
*lh*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [62\\*](#).  
*lig\_kern*: [41](#), [42](#), [53](#).  
*lig\_kern\_base*: [53](#), [56](#), [61](#), [64](#), [66\\*](#), [69](#).  
*lig\_kern\_command*: [38](#), [42](#).  
*lig\_kern\_restart*: [56](#), [120](#).  
*lig\_kern\_restart\_end*: [56](#).  
*lig\_kern\_start*: [56](#), [120](#).  
*lig\_lookahead*: [5\\*](#), [116](#), [117](#).  
*lig\_stack*: [116](#), [117](#), [122](#).  
*lig\_tag*: [41](#), [63](#), [120](#).  
*line\_length*: [17\\*](#), [18](#), [94\\*](#), [95](#), [99](#), [100](#), [101](#).  
*list\_tag*: [41](#), [63](#), [210](#).  
*load\_fonts*: [96](#), [98](#).  
*logo\_font*: [52\\*](#), [58](#), [97](#), [98](#), [115\\*](#), [172](#).  
*logo\_font\_name*: [78](#), [97](#).  
*longest\_keyword*: [75](#), [81](#), [82](#), [98](#).  
**loop**: [6](#), [7](#).  
*m*: [3\\*](#), [81](#), [98](#), [114](#), [220](#).  
*mag*: [21](#), [23](#), [24](#), [25](#).  
*make\_string*: [74](#), [83](#), [91](#), [101](#), [221](#).  
*max\_depth*: [140](#), [143](#), [144](#), [147](#).  
*max\_h*: [102](#), [103](#), [115\\*](#), [203](#).  
*max\_height*: [140](#), [143](#), [144](#), [146](#).  
*max\_k*: [116](#).  
*max\_keyword*: [77](#), [78](#), [79](#), [83](#).  
*max\_labels*: [5\\*](#), [139](#), [140](#), [141](#), [142](#), [161](#).  
*max\_m*: [29](#), [31](#).  
*max\_n*: [29](#), [31](#).  
*max\_node*: [140](#), [141](#), [144](#), [187](#).  
*max\_quarterword*: [52\\*](#).  
*max\_strings*: [5\\*](#), [70](#), [74](#), [91](#).  
*max\_v*: [102](#), [103](#), [115\\*](#), [170](#).  
*max\_x*: [165](#), [166](#), [170](#), [218](#).  
*max\_y*: [165](#), [166](#), [167](#), [170](#), [218](#).  
*memory\_word*: [52\\*](#), [53](#).  
*mid*: [43](#).  
*min\_m*: [29](#), [31](#).  
*min\_n*: [29](#), [31](#).  
*min\_quarterword*: [52\\*](#), [53](#), [55\\*](#), [61](#), [69](#), [116](#).  
*min\_x*: [165](#), [166](#), [167](#), [170](#), [218](#).  
*min\_y*: [165](#), [166](#), [170](#).  
Missing boc: [219\\*](#).  
Missing dot char: [184](#).  
Missing pixel char: [169](#).  
*more\_name*: [86](#), [90\\*](#), [95](#).  
*n*: [3\\*](#), [92](#), [114](#).  
*name\_length*: [92](#).  
*name\_of\_file*: [47\\*](#), [48](#), [92](#).  
*nd*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [63](#).  
*ne*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [63](#).  
*nearest\_dot*: [148](#), [150](#), [192](#), [201](#), [202](#).  
*new\_row\_0*: [29](#), [30](#), [215\\*](#).  
*new\_row\_1*: [29](#).  
*new\_row\_164*: [29](#).  
*next*: [139](#), [140](#), [143](#), [144](#), [145](#), [146](#), [151](#), [159](#),  
[160](#), [162](#), [163](#), [172](#), [173](#), [181](#), [187](#), [189](#), [191](#),  
[193](#), [200](#), [201](#), [202](#).  
*next\_char*: [42](#), [55\\*](#), [120](#).  
*nh*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [63](#).  
*ni*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [63](#).  
**nil**: [7](#).  
*nk*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [66\\*](#).  
*nl*: [37](#), [38](#), [42](#), [58](#), [60\\*](#), [61](#), [63](#), [66\\*](#), [69](#).  
No preamble: [221](#).  
No room for TFM file: [61](#).  
*no\_op*: [29](#), [30](#), [32](#), [79](#), [81](#), [85](#), [154](#), [215\\*](#), [219\\*](#).  
*no\_operation*: [79](#), [81](#), [154](#).  
*no\_tag*: [41](#), [63](#).  
*node\_ins*: [143](#), [188](#), [190](#), [194](#).  
*node\_pointer*: [139](#), [140](#), [141](#), [143](#), [145](#), [149](#), [150](#),  
[158](#), [160](#), [185](#), [186](#), [194](#).  
*non\_address*: [52\\*](#), [53](#), [69](#), [120](#).  
*non\_char*: [52\\*](#), [53](#), [116](#), [122](#).  
*nop*: [19](#), [21](#), [24](#), [25](#).  
*not\_found*: [6](#), [81](#), [82](#), [98](#), [99](#).  
*np*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [68](#).  
*null*: [139](#), [150](#), [161](#), [162](#), [172](#), [173](#), [181](#), [187](#), [189](#),  
[191](#), [192](#), [193](#), [200](#), [201](#), [202](#).  
*null\_string*: [77](#), [79](#), [81](#), [83](#), [86](#), [88\\*](#), [91](#), [94\\*](#), [97](#),  
[98](#), [101](#), [154](#).  
*num*: [21](#), [23](#), [25](#).  
*nw*: [37](#), [38](#), [58](#), [60\\*](#), [61](#), [63](#).  
*n1*: [81](#), [83](#), [98](#), [100](#).  
*n2*: [81](#), [83](#), [98](#), [100](#).  
*oct*: [194](#), [195](#), [196](#), [197](#), [198](#), [199](#).  
*octant*: [191](#), [192](#), [194](#), [195](#).  
*odd*: [210](#).  
*offset\_code*: [77](#), [154](#).  
*offset\_x*: [155](#), [156](#), [157](#), [170](#).  
*offset\_y*: [155](#), [156](#), [157](#), [170](#).  
*op\_byte*: [42](#), [55\\*](#), [56](#), [120](#), [122](#).

- open\_dvi\_file*: [47\\*](#), [94\\*](#)  
*open\_gf\_file*: [47\\*](#), [94\\*](#)  
*open\_tfm\_file*: [47\\*](#), [98](#).  
*ord*: [12](#).  
 oriental characters: [32](#).  
**othercases**: [2](#).  
*others*: [2](#).  
*output*: [3\\*](#)  
*over\_col*: [168](#), [170](#), [202](#), [203](#).  
*overflow\_line*: [181](#), [182](#), [202](#), [203](#).  
*overlap*: [145](#), [146](#), [147](#), [196](#), [197](#), [198](#), [199](#).  
*p*: [143](#), [145](#), [150](#), [185](#), [186](#), [194](#), [220](#).  
*pack\_file\_name*: [92](#), [94\\*](#), [98](#).  
*page\_header*: [78](#), [172](#).  
*page\_height*: [168](#), [170](#).  
*page\_width*: [168](#), [170](#), [203](#).  
*paint\_black*: [215\\*](#), [216](#), [218](#).  
*paint\_switch*: [28](#), [29](#).  
*paint\_0*: [29](#), [30](#).  
*paint1*: [29](#), [30](#), [215\\*](#)  
*paint2*: [29](#), [30](#), [215\\*](#)  
*paint3*: [29](#), [30](#), [215\\*](#)  
*param*: [39](#), [44](#), [57](#).  
*param\_base*: [53](#), [57](#), [61](#), [67](#), [68](#), [69](#).  
*param\_end*: [57](#).  
*place\_label*: [193](#), [194](#), [195](#).  
*plus\_sign*: [78](#), [202](#).  
*pool\_pointer*: [70](#), [71](#), [81](#), [87](#), [98](#), [116](#).  
*pool\_ptr*: [71](#), [73](#), [74](#), [75](#), [77](#), [90\\*](#), [219\\*](#)  
*pool\_size*: [5\\*](#), [70](#), [73](#).  
*pop*: [20](#), [21](#), [22](#), [25](#), [171](#), [172](#), [176](#), [177](#), [178](#), [188](#),  
[190](#), [194](#), [202](#), [208](#), [218](#).  
*pop\_stack*: [122](#), [123](#).  
*post*: [19](#), [21](#), [22](#), [25](#), [26](#), [27](#), [29](#), [31](#), [33](#), [115\\*](#), [219\\*](#)  
*post\_post*: [21](#), [22](#), [25](#), [26](#), [29](#), [31](#), [33](#), [115\\*](#)  
*pre*: [19](#), [21](#), [22](#), [27](#), [29](#), [221](#).  
*pre\_max\_x*: [155](#), [156](#), [159](#), [163](#), [170](#).  
*pre\_max\_y*: [155](#), [156](#), [159](#), [163](#), [170](#).  
*pre\_min\_x*: [155](#), [156](#), [159](#), [163](#), [170](#).  
*pre\_min\_y*: [155](#), [156](#), [159](#), [163](#), [170](#).  
*prev*: [139](#), [140](#), [143](#), [144](#), [147](#), [152](#), [160](#), [201](#), [202](#).  
*print*: [3\\*](#), [99](#), [138\\*](#)  
*print\_ln*: [3\\*](#), [8\\*](#), [219\\*](#)  
*print\_nl*: [3\\*](#), [58](#), [94\\*](#), [99](#), [138\\*](#), [154](#), [163](#).  
*print\_pascal\_string*: [47\\*](#)  
*print\_real*: [138\\*](#)  
*proofing*: [32](#).  
*push*: [20](#), [21](#), [22](#), [25](#), [171](#), [208](#).  
*put\_rule*: [21](#), [22](#), [176](#), [177](#).  
*put1*: [21](#).  
*put2*: [21](#).  
*put3*: [21](#).  
*put4*: [21](#).  
*q*: [143](#), [145](#), [220](#).  
*qi*: [52\\*](#), [62\\*](#), [69](#), [116](#), [118\\*](#), [120](#).  
*qo*: [52\\*](#), [55\\*](#), [69](#), [111\\*](#), [122](#), [123](#), [210](#).  
*qqqq*: [53](#), [55\\*](#), [63](#), [66\\*](#), [67](#), [120](#).  
*quad*: [44](#).  
*quarterword*: [52\\*](#), [117](#).  
*qw*: [58](#), [62\\*](#)  
*r*: [138\\*](#), [143](#), [145](#), [220](#).  
*read*: [50](#), [51\\*](#)  
*read\_font\_info*: [58](#), [98](#).  
*read\_ln*: [17\\*](#)  
*read\_tfm\_word*: [50](#), [60\\*](#), [62\\*](#), [64](#), [68](#).  
*read\_two\_halves*: [60\\*](#)  
*read\_two\_halves\_end*: [60\\*](#)  
*real*: [114](#), [134](#), [138\\*](#), [168](#).  
*rem\_byte*: [55\\*](#), [56](#), [122](#), [210](#).  
*remainder*: [40](#), [41](#), [42](#).  
*rep*: [43](#).  
*reset*: [47\\*](#)  
*reswitch*: [6](#), [210](#), [218](#).  
**return**: [6](#), [7](#).  
*rewrite*: [47\\*](#)  
*rho*: [206](#), [207](#), [217](#).  
*right\_coords*: [186](#), [190](#), [196](#), [197](#), [198](#), [199](#).  
*right\_quotes*: [78](#), [172](#).  
*right1*: [21](#).  
*right2*: [21](#).  
*right3*: [21](#).  
*right4*: [21](#), [22](#), [119](#), [121](#), [171](#), [209](#).  
*round*: [114](#), [167](#), [170](#), [178](#), [179](#), [180](#), [209](#), [218](#).  
*rule\_code*: [77](#), [154](#).  
*rule\_ptr*: [158](#), [159](#), [161](#), [173](#).  
*rule\_size*: [158](#), [159](#), [173](#), [176](#), [177](#), [178](#).  
*rule\_slant*: [134](#), [137](#), [173](#), [178](#).  
*rule\_thickness*: [154](#), [155](#), [156](#), [159](#).  
*rule\_thickness\_code*: [77](#), [79](#), [154](#).  
*s*: [58](#), [116](#), [220](#).  
 Samuel, Arthur Lee: [191](#).  
*save\_c*: [116](#).  
*sc*: [53](#), [55\\*](#), [56](#), [57](#), [64](#), [66\\*](#), [68](#).  
*scaled*: [9](#), [29](#), [31](#), [32](#), [52\\*](#), [53](#), [58](#), [84](#), [96](#), [102](#), [116](#),  
[117](#), [140](#), [145](#), [150](#), [155](#), [167](#), [168](#), [171](#), [174](#), [183](#).  
*second\_octant*: [192](#), [196](#), [197](#), [198](#), [199](#).  
*select\_font*: [111\\*](#), [172](#), [173](#), [181](#), [218](#).  
*send\_it*: [116](#), [119](#), [121](#).  
*set\_char\_0*: [21](#).  
*set\_char\_1*: [21](#).  
*set\_char\_127*: [21](#).  
*set\_cur\_r*: [116](#), [122](#), [123](#).  
*set\_paths*: [219\\*](#)  
*set\_rule*: [19](#), [21](#).

- set1*: 21, 22, 113.  
*set2*: 21.  
*set3*: 21.  
*set4*: 21.  
*seventh\_octant*: 192, 196, 197, 198, 199.  
*signed\_quad*: 51\*, 81, 84, 85, 165.  
*sixth\_octant*: 192, 196, 197, 198, 199.  
*sixty\_four\_cases*: 215\*.  
*skip\_byte*: 42, 55\*, 120.  
*skip\_nop*: 85, 215\*.  
*skip0*: 29, 30, 215\*.  
*skip1*: 29, 30, 215\*.  
*skip2*: 29, 30, 215\*.  
*skip3*: 29, 30, 215\*.  
*slant*: 44, 57, 68, 137, 169.  
 slant fonts: 35, 39.  
*slant\_complaint*: 138\*, 178.  
*slant\_font*: 52\*, 58, 77, 97, 98, 100, 137, 154, 173.  
*slant\_n*: 134, 137, 178.  
*slant\_ratio*: 167, 168, 169, 170.  
*slant\_reported*: 134, 137, 138\*.  
*slant\_unit*: 134, 137, 178, 179, 180.  
*small\_logo*: 78, 172.  
 Sorry, I can't...: 138\*.  
 sp: 23.  
*space*: 44, 57, 119, 184.  
*space\_shrink*: 44.  
*space\_stretch*: 44.  
 Special font subst...: 99.  
*stack\_ptr*: 116, 122, 123.  
*start\_gf*: 94\*, 219\*.  
*starting\_col*: 208, 214, 217, 218.  
*stdin*: 16\*.  
*stdout*: 3\*, 16\*.  
*stop\_flag*: 42, 66\*, 120.  
*store\_four\_quarters*: 62\*, 63, 66\*, 67.  
*store\_scaled*: 64, 66\*, 68.  
*str\_number*: 70, 71, 74, 80, 86, 92, 93, 96, 116, 140, 220.  
*str\_pool*: 70, 71, 73, 74, 75, 83, 92, 100, 111\*, 112, 116, 221.  
*str\_ptr*: 71, 74, 75, 77, 91, 101, 154, 219\*, 221.  
*str\_room*: 73, 83, 90\*, 101.  
*str\_start*: 70, 71, 72, 74, 75, 77, 83, 91, 92, 100, 112, 116, 219\*, 221.  
*suppress\_lig*: 116, 117, 120, 122.  
*sw*: 58, 64, 68.  
 system dependencies: 2, 3\*, 8\*, 11\*, 14\*, 16\*, 17\*, 26, 33, 45, 47\*, 50, 51\*, 52\*, 78, 86, 87, 88\*, 89, 90\*, 91, 92, 105, 107\*, 222.  
 t: 220.  
 tag: 40, 41.  
 Tardy font change...: 154.  
*temp\_x*: 173, 174, 177, 178.  
*temp\_y*: 173, 174, 176, 178.  
*term\_in*: 16\*, 17\*.  
*term\_out*: 3\*.  
*terminal\_line\_length*: 5\* 16\*, 17\*, 18, 94\*.  
*test\_read\_access*: 47\*, 88\*.  
*text\_char*: 11\*, 12.  
*text\_file*: 11\*.  
*tfm\_ext*: 78, 98.  
*tfm\_file*: 46, 47\*, 50, 58.  
*TFM\_FILE\_PATH\_BIT*: 219\*.  
*TFMFILEPATH*: 47\*.  
*third\_octant*: 192, 196, 197, 198, 199.  
*thirty\_two\_cases*: 215\*.  
*thrice\_x\_height*: 183, 184, 202.  
*time\_stamp*: 172, 220, 221.  
*title\_code*: 77, 154.  
*title\_font*: 52\*, 58, 77, 78, 97, 98, 100, 115\*, 154, 172.  
*title\_head*: 160, 161, 162, 172.  
*title\_tail*: 160, 161, 162, 172.  
*toint*: 51\*, 60\*, 62\*, 66\*.  
*tol*: 173.  
 Too many labels: 74, 141.  
 Too many strings: 73, 91.  
 top: 43.  
*top\_coords*: 185, 190, 196, 197, 198, 199.  
*total\_pages*: 102, 103, 115\*, 172.  
*true*: 7, 90\*, 95, 96, 97, 122, 146, 147, 148, 151, 152, 172, 190, 194, 202, 215\*, 221.  
*twin*: 148, 149, 150, 151, 152, 192.  
*two\_to\_the*: 126, 127, 128, 129, 206.  
*typeset*: 113, 121, 179, 180, 210.  
*uexit*: 8\*, 115\*.  
*unity*: 9, 62\*, 114, 137, 168, 169, 170.  
*unsc\_slant\_ratio*: 168, 169, 170, 209.  
*unsc\_x\_ratio*: 168, 169, 170, 209, 218.  
*unsc\_y\_ratio*: 168, 169, 170, 218.  
*update\_terminal*: 16\*, 17\*.  
*use\_logo*: 172, 220, 221.  
 v: 84, 98, 218.  
 Vanishing pixel size: 169.  
*vppp*: 31.  
 WEB: 72.  
*web2c*: 52\*.  
*white*: 29.  
*widest\_row*: 5\*, 165, 211, 218.  
*width\_base*: 53, 55\*, 61, 63, 64, 69.  
*width\_index*: 40, 53.  
*write*: 3\*.  
*write\_chunk*: 107\*.  
*write\_dvi*: 107\* 108, 109.

*write\_ln*: 3\*  
**Wrong ID**: 221.  
*w0*: 21.  
*w1*: 21.  
*w2*: 21.  
*w3*: 21.  
*w4*: 21.  
*x*: 23, 110, 114, 116, 166, 167, 171.  
*x\_height*: 44, 57, 184.  
*x\_left*: 145, 146, 147.  
*x\_offset*: 154, 155, 156, 167.  
*x\_offset\_code*: 77, 154.  
*x\_ratio*: 167, 168, 169, 170, 202.  
*x\_right*: 145, 146, 147.  
*xchr*: 12, 13, 14\*, 15, 92.  
**xclause**: 7.  
*xl*: 139, 140, 145, 146, 147, 148, 158, 185, 186, 188.  
*xord*: 12, 15, 17\*, 94\*  
*xr*: 139, 140, 145, 146, 147, 148, 158, 185, 186, 188, 191.  
*xx*: 139, 140, 148, 151, 152, 158, 163, 185, 186, 187, 188, 190, 192, 194, 195, 202.  
*xxx1*: 21, 29, 30, 79, 81, 85, 154, 215\*, 219\*  
*xxx2*: 21, 29, 30, 81, 85, 215\*  
*xxx3*: 21, 29, 30, 81, 85, 215\*  
*xxx4*: 21, 29, 30, 79, 81, 85, 215\*  
*x0*: 21, 158, 159, 173.  
*x1*: 21, 158, 159, 173.  
*x2*: 21.  
*x3*: 21.  
*x4*: 21.  
*y*: 166, 167, 171.  
*y\_bot*: 145, 146, 147.  
*y\_offset*: 154, 155, 156, 167.  
*y\_offset\_code*: 77, 154.  
*y\_ratio*: 167, 168, 169, 170, 202.  
*y\_thresh*: 145, 146, 147.  
*y\_top*: 145, 146, 147.  
*yb*: 139, 140, 143, 145, 146, 147, 148, 158, 185, 186, 188.  
*yt*: 139, 140, 143, 145, 146, 147, 148, 158, 185, 186, 188.  
*yy*: 139, 140, 142, 143, 146, 147, 148, 151, 152, 163, 185, 186, 187, 188, 190, 192, 194, 195, 202.  
*yyy*: 29, 30, 32, 79, 81, 84, 85, 215\*  
*y0*: 21, 158, 159, 173.  
*y1*: 21, 158, 159, 173.  
*y2*: 21.  
*y3*: 21.  
*y4*: 21.  
*z*: 58, 166.  
*z0*: 21, 22, 179, 180.

*z1*: 21.  
*z2*: 21.  
*z3*: 21.  
*z4*: 21, 22, 179, 180.

- ⟨ Add a full set of  $k$ -bit characters 128 ⟩ Used in section 126.
- ⟨ Add more rows to  $a$ , until 12-bit entries are obtained 213 ⟩ Used in section 218.
- ⟨ Add special  $k$ -bit characters of the form  $X..XO..O$  129 ⟩ Used in section 126.
- ⟨ Adjust the maximum page width 203 ⟩ Used in section 164\*.
- ⟨ Advance to the next row that needs to be typeset; or **return**, if we're all done 217 ⟩ Used in section 218.
- ⟨ Carry out a ligature operation, updating the cursor structure and possibly advancing  $k$ ; **goto continue** if the cursor doesn't advance, otherwise **goto done** 122 ⟩ Used in section 120.
- ⟨ Compute the octant code for floating label  $p$  192 ⟩ Used in section 191.
- ⟨ Constants in the outer block 5\* ⟩ Used in section 3\*.
- ⟨ Declare the procedure called *load\_fonts* 98 ⟩ Used in section 111\*.
- ⟨ Empty the last bytes out of *dvi\_buf* 109 ⟩ Used in section 115\*.
- ⟨ Enter a dot for label  $p$  in the rectangle list, and typeset the dot 188 ⟩ Used in section 187.
- ⟨ Enter a prescribed label for node  $p$  into the rectangle list, and typeset it 190 ⟩ Used in section 189.
- ⟨ Find nearest dots, to help in label positioning 191 ⟩ Used in section 181.
- ⟨ Find non-overlapping coordinates, if possible, and **goto found**; otherwise set *place\_label*  $\leftarrow$  *false* and **return** 195 ⟩ Used in section 194.
- ⟨ Finish reading the parameters of the *boc* 165 ⟩ Used in section 164\*.
- ⟨ Finish the DVI file and **goto final\_end** 115\* ⟩ Used in section 219\*.
- ⟨ Get online special input 99 ⟩ Used in section 98.
- ⟨ Get ready to convert METAFONT coordinates to DVI coordinates 170 ⟩ Used in section 164\*.
- ⟨ Globals in the outer block 12, 16\*, 18, 37, 46, 48, 49, 53, 71, 76, 80, 86, 87, 93, 96, 102, 105, 117, 127, 134, 140, 149, 155, 158, 160, 166, 168, 174, 182, 183, 207, 211, 212, 220 ⟩ Used in section 3\*.
- ⟨ If the keyword in *buffer*[1 ..  $l$ ] is known, change  $c$  and **goto done** 83 ⟩ Used in section 82.
- ⟨ If there's a ligature or kern at the cursor position, update the cursor data structures, possibly advancing  $k$ ; continue until the cursor wants to move right 120 ⟩ Used in section 116.
- ⟨ Initialize global variables that depend on the font data 137, 169, 175, 184, 205, 206 ⟩ Used in section 98.
- ⟨ Initialize the strings 77, 78, 88\* ⟩ Used in section 219\*.
- ⟨ Initialize variables for the next character 144, 156, 161 ⟩ Used in section 219\*.
- ⟨ Look for overlaps in node  $q$  and its predecessors 147 ⟩ Used in section 145.
- ⟨ Look for overlaps in the successors of node  $q$  146 ⟩ Used in section 145.
- ⟨ Make final adjustments and **goto done** 69 ⟩ Used in section 59.
- ⟨ Move the cursor to the right and **goto continue**, if there's more work to do in the current word 123 ⟩ Used in section 116.
- ⟨ Move to column  $j$  in the DVI output 209 ⟩ Used in section 208.
- ⟨ Output a horizontal rule 177 ⟩ Used in section 173.
- ⟨ Output a vertical rule 176 ⟩ Used in section 173.
- ⟨ Output all attachable labels 193 ⟩ Used in section 181.
- ⟨ Output all dots 187 ⟩ Used in section 181.
- ⟨ Output all labels for the current character 181 ⟩ Used in section 164\*.
- ⟨ Output all overflow labels 200 ⟩ Used in section 181.
- ⟨ Output all prescribed labels 189 ⟩ Used in section 181.
- ⟨ Output all rules for the current character 173 ⟩ Used in section 164\*.
- ⟨ Output the equivalent of  $k$  copies of character  $v$  210 ⟩ Used in section 208.
- ⟨ Output the font name whose internal number is  $f$  112 ⟩ Used in section 111\*.
- ⟨ Output the *bop* and the title line 172 ⟩ Used in section 164\*.
- ⟨ Override the offsets 157 ⟩ Used in section 154.
- ⟨ Paint  $k$  bits and read another command 216 ⟩ Used in section 215\*.
- ⟨ Process a character 164\* ⟩ Used in section 219\*.
- ⟨ Process a no-op command 154 ⟩ Used in section 219\*.
- ⟨ Process the preamble 221 ⟩ Used in section 219\*.
- ⟨ Put the bits for the next row, times  $l$ , into  $a$  214 ⟩ Used in section 213.

- ⟨ Read and check the font data; *abend* if the TFM file is malformed; otherwise **goto done** 59 ⟩  
Used in section 58.
- ⟨ Read and process GF commands until coming to the end of this row 215\* ⟩ Used in section 214.
- ⟨ Read box dimensions 64 ⟩ Used in section 59.
- ⟨ Read character data 63 ⟩ Used in section 59.
- ⟨ Read extensible character recipes 67 ⟩ Used in section 59.
- ⟨ Read font parameters 68 ⟩ Used in section 59.
- ⟨ Read ligature/kern program 66\* ⟩ Used in section 59.
- ⟨ Read the next *k* characters of the GF file; change *c* and **goto done** if a keyword is recognized 82 ⟩  
Used in section 81.
- ⟨ Read the TFM header 62\* ⟩ Used in section 59.
- ⟨ Read the TFM size fields 60\* ⟩ Used in section 59.
- ⟨ Remove all rectangles from list, except for dots that have labels 201 ⟩ Used in section 200.
- ⟨ Replace *z* by *z'* and compute  $\alpha, \beta$  65 ⟩ Used in section 64.
- ⟨ Scan the file name in the buffer 95 ⟩ Used in section 94\*.
- ⟨ Search buffer for valid keyword; if successful, **goto found** 100 ⟩ Used in section 99.
- ⟨ Search for the nearest dot in nodes following *p* 151 ⟩ Used in section 150.
- ⟨ Search for the nearest dot in nodes preceding *p* 152 ⟩ Used in section 150.
- ⟨ Set initial values 13, 14\*, 15, 54, 97, 103, 106, 118\*, 126, 142 ⟩ Used in section 3\*.
- ⟨ Store a label 163 ⟩ Used in section 154.
- ⟨ Store a rule 159 ⟩ Used in section 154.
- ⟨ Store a title 162 ⟩ Used in section 154.
- ⟨ Try the first choice for label direction 196 ⟩ Used in section 195.
- ⟨ Try the fourth choice for label direction 199 ⟩ Used in section 195.
- ⟨ Try the second choice for label direction 197 ⟩ Used in section 195.
- ⟨ Try the third choice for label direction 198 ⟩ Used in section 195.
- ⟨ Try to output a diagonal rule 178 ⟩ Used in section 173.
- ⟨ Types in the outer block 9, 10, 11\*, 45, 52\*, 70, 79, 104, 139 ⟩ Used in section 3\*.
- ⟨ Typeset a space in font *f* and advance *k* 119 ⟩ Used in section 116.
- ⟨ Typeset an overflow label for *p* 202 ⟩ Used in section 200.
- ⟨ Typeset character *cur\_l*, if it exists in the font; also append an optional kern 121 ⟩ Used in section 116.
- ⟨ Typeset the pixels of the current row 208 ⟩ Used in section 218.
- ⟨ Update the font name or area 101 ⟩ Used in section 99.
- ⟨ Use size fields to allocate font information 61 ⟩ Used in section 59.
- ⟨ Vertically typeset *p* copies of character *k* + 1 180 ⟩ Used in section 178.
- ⟨ Vertically typeset *q* copies of character *k* 179 ⟩ Used in section 178.

# The Gf<sub>t</sub>oDVI processor

(Version 3.0, October 1989)

	Section	Page
Introduction .....	1	302
The character set .....	10	304
Device-independent file format .....	19	305
Generic font file format .....	27	305
Extensions to the generic format .....	34	305
Font metric data .....	36	305
Input from binary files .....	45	305
Reading the font information .....	52	306
The string pool .....	70	309
File names .....	86	309
Shipping pages out .....	102	310
Rudimentary typesetting .....	116	311
Gray fonts .....	124	312
Slant fonts .....	134	312
Representation of rectangles .....	139	313
Doing the labels .....	153	313
Doing the pixels .....	204	314
The main program .....	219	315
System-dependent changes .....	222	316
Index .....	223	316

The preparation of this report was supported in part by the National Science Foundation under grants IST-8201926, MCS-8300984, and CCR-8610181, and by the System Development Foundation. 'T<sub>E</sub>X' is a trademark of the American Mathematical Society. 'METAFONT' is a trademark of Addison-Wesley Publishing Company.