The VPtoVF processor

(Version 1.6, January 2014)

	Section	Page
Introduction	1	202
Property list description of font metric data	5	203
Basic input routines	23	204
Basic scanning routines	36	205
Scanning property names	44	205
Scanning numeric data	59	205
Storing the property values	77	205
The input phase	91	206
Assembling the mappings	122	207
The checking and massaging phase	138	207
The TFM output phase	156	208
The VF output phase	175	209
The main program	180	210
System-dependent changes	182	211
Index	190	213

Editor's Note: The present variant of this C/WEB source file has been modified for use in the TEX Live system.

The following sections were changed by the change file: 1, 2, 3, 6, 22, 24, 31, 32, 33, 89, 118, 144, 152, 153, 156, 165, 175, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190.

The preparation of this program was supported in part by the National Science Foundation and by the System Development Foundation. 'TEX' is a trademark of the American Mathematical Society.

1* Introduction. The `VPtoVF` utility program converts virtual-property-list ("`VPL`") files into an equivalent pair of files called a virtual font ("`VF`") file and a TEX font metric ("`TFM`") file. It also makes a thorough check of the given `VPL` file, so that the `VF` file should be acceptable to device drivers and the `TFM` file should be acceptable to TEX.

`VPtoVF` is an extended version of the program `PLtoTF`, which is part of the standard TEXware library. The idea of a virtual font was inspired by the work of David R. Fuchs who designed a similar set of conventions in 1984 while developing a device driver for ArborText, Inc. He wrote a somewhat similar program called `PLFONT`.

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

define *my_name* ≡ ´`vptovf`´
define *banner* ≡ ´`This␣is␣VPtoVF,␣Version␣1.6`´ { printed when the program starts }

2* This program is written entirely in standard Pascal, except that it has to do some slightly system-dependent character code conversion on input. Furthermore, lower case letters are used in error messages; they could be converted to upper case if necessary. The input is read from *vpl_file*, and the output is written on *vf_file* and *tfm_file*; 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 *print*(#) ≡ *write*(*stderr*, #)
define *print_ln*(#) ≡ *write_ln*(*stderr*, #)
define *print_real*(#) ≡ *fprint_real*(*stderr*, #)

program *VPtoVF*(*vpl_file*, *vf_file*, *tfm_file*, *output*);
const ⟨ Constants in the outer block 3* ⟩
type ⟨ Types in the outer block 23 ⟩
var ⟨ Globals in the outer block 5 ⟩
⟨ Define *parse_arguments* 182* ⟩
procedure *initialize*; { this procedure gets things started properly }
var ⟨ Local variables for initialization 25 ⟩
begin *kpse_set_program_name*(*argv*[0], *my_name*); *parse_arguments*; ⟨ Set initial values 6* ⟩
end;

3* The following parameters can be changed at compile time to extend or reduce `VPtoVF`'s capacity.

⟨ Constants in the outer block 3* ⟩ ≡
buf_size = 3000; { length of lines displayed in error messages }
max_header_bytes = 1000; { four times the maximum number of words allowed in the `TFM` file header block, must be 1024 or less }
vf_size = 100000; { maximum length of *vf* data, in bytes }
max_stack = 100; { maximum depth of simulated `DVI` stack }
max_param_words = 254; { the maximum number of `fontdimen` parameters allowed }
max_lig_steps = 32510; { maximum length of ligature program, must be at most 32767 − 257 = 32510 }
max_kerns = 5000; { the maximum number of distinct kern values }
hash_size = 32579;
{ preferably a prime number, a bit larger than the number of character pairs in lig/kern steps }

This code is used in section 2*.

6:* ⟨ Set initial values 6* ⟩ ≡

```
reset(vpl_file, vpl_name);
if verbose then
  begin print(banner); print_ln(version_string);
  end;
```

See also sections 22*, 26, 28, 30, 32*, 45, 49, 68, 80, 84, and 148.

This code is used in section 2*.

22:* On some systems you may have to do something special to write a packed file of bytes.

⟨ Set initial values 6* ⟩ +≡

```
rewritebin(vf_file, vf_name); rewritebin(tfm_file, tfm_name);
```

24.* One of the things `VPtoVF` has to do is convert characters of strings to ASCII form, since that is the code used for the family name and the coding scheme in a `TFM` file. An array *xord* is used to do the conversion from *char*; the method below should work with little or no change on most Pascal systems.

```
define char ≡ 0 .. 255
define first_ord = 0   { ordinal number of the smallest element of char }
define last_ord = 127   { ordinal number of the largest element of char }
⟨ Globals in the outer block 5 ⟩ +≡
xord: array [char] of ASCII_code;   { conversion table }
```

31.* Just before each `CHARACTER` property list is evaluated, the character code is printed in octal notation. Up to eight such codes appear on a line; so we have a variable to keep track of how many are currently there.

```
⟨ Globals in the outer block 5 ⟩ +≡
chars_on_line: 0 .. 8;   { the number of characters printed on the current line }
perfect: boolean;   { was the file free of errors? }
```

32.* ⟨ Set initial values 6* ⟩ +≡

```
chars_on_line ← 0; perfect ← true;   { innocent until proved guilty }
```

33.* The following routine prints an error message and an indication of where the error was detected. The error message should not include any final punctuation, since this procedure supplies its own.

```
define err_print(#) ≡
        begin if chars_on_line > 0 then print_ln('␣');
        print(#); show_error_context;
        end
procedure show_error_context;   { prints the current scanner location }
  var k: 0 .. buf_size;   { an index into buffer }
  begin print_ln('␣(line␣', line : 1, '):');
  if ¬left_ln then print('...');
  for k ← 1 to loc do print(buffer[k]);   { print the characters already scanned }
  print_ln('␣');
  if ¬left_ln then print('␣␣␣');
  for k ← 1 to loc do print('␣');   { space out the second line }
  for k ← loc + 1 to limit do print(buffer[k]);   { print the characters yet unseen }
  if right_ln then print_ln('␣') else print_ln('...');
  chars_on_line ← 0; perfect ← false;
  end;
```

89:* When we are nearly ready to output the `TFM` file, we will set $index[p] \leftarrow k$ if the dimension in $memory[p]$ is being rounded to the kth element of its list.

define *index* ≡ *index_var*
define *class* ≡ *class_var*

⟨ Globals in the outer block 5 ⟩ +≡
index: **array** [*pointer*] **of** *byte*;
excess: *byte*; { number of words to remove, if list is being shortened }

118:* Finally we come to the part of `VPtoVF`'s input mechanism that is used most, the processing of individual character data.

```
⟨ Read character info list 118* ⟩ ≡
  begin c ← get_byte;  { read the character code that is being specified }
  if verbose then ⟨ Print c in octal notation 137 ⟩;
  while level = 1 do
    begin while cur_char = "␣" do get_next;
    if cur_char = "(" then ⟨ Read a character property 119 ⟩
    else if cur_char = ")" then skip_to_end_of_item
      else junk_error;
    end;
  if char_wd[c] = 0 then char_wd[c] ← sort_in(width, 0);  { legitimatize c }
  finish_inner_property_list;
  end
```

This code is used in section 180.

144:* **define** *round_message*(#) ≡
if *delta* > 0 **then**
begin *print*(´I␣had␣to␣round␣some␣´, #, ´s␣by␣´);
print_real((((*delta* + 1) **div** 2)/´4000000), 1, 7); *print_ln*(´␣units.´);
end

⟨ Put the width, height, depth, and italic lists into final form 144* ⟩ ≡
delta ← *shorten*(*width*, 255); *set_indices*(*width*, *delta*); *round_message*(´width´);
delta ← *shorten*(*height*, 15); *set_indices*(*height*, *delta*); *round_message*(´height´);
delta ← *shorten*(*depth*, 15); *set_indices*(*depth*, *delta*); *round_message*(´depth´);
delta ← *shorten*(*italic*, 63); *set_indices*(*italic*, *delta*); *round_message*(´italic␣correction´);

This code is used in section 139.

152:* (More good stuff from TFtoPL.)

ifdef(´notdef´)
function *f*(*h*, *x*, *y* : *indx*): *indx*;
begin end;
{ compute *f* for arguments known to be in *hash*[*h*] }
endif(´notdef´)
function *eval*(*x*, *y* : *indx*): *indx*; { compute *f*(*x*, *y*) with hashtable lookup }
var *key*: *integer*; { value sought in hash table }
begin *key* ← 256 ∗ *x* + *y* + 1; *h* ← (1009 ∗ *key*) **mod** *hash_size*;
while *hash*[*h*] > *key* **do**
if *h* > 0 **then** *decr*(*h*) **else** *h* ← *hash_size*;
if *hash*[*h*] < *key* **then** *eval* ← *y* { not in ordered hash table }
else *eval* ← *f*(*h*, *x*, *y*);
end;

153:* Pascal's beastly convention for *forward* declarations prevents us from saying **function** *f*(*h*, *x*, *y* : *indx*): *indx* here.

function *f*(*h*, *x*, *y* : *indx*): *indx*;
begin case *class*[*h*] **of**
simple: *do_nothing*;
left_z: **begin** *class*[*h*] ← *pending*; *lig_z*[*h*] ← *eval*(*lig_z*[*h*], *y*); *class*[*h*] ← *simple*;
end;
right_z: **begin** *class*[*h*] ← *pending*; *lig_z*[*h*] ← *eval*(*x*, *lig_z*[*h*]); *class*[*h*] ← *simple*;
end;
both_z: **begin** *class*[*h*] ← *pending*; *lig_z*[*h*] ← *eval*(*eval*(*x*, *lig_z*[*h*]), *y*); *class*[*h*] ← *simple*;
end;
pending: **begin** *x_lig_cycle* ← *x*; *y_lig_cycle* ← *y*; *lig_z*[*h*] ← 257; *class*[*h*] ← *simple*;
end; { the value 257 will break all cycles, since it's not in *hash* }
end; { there are no other cases }
f ← *lig_z*[*h*];
end;

156.* The TFM output phase. Now that we know how to get all of the font data correctly stored in `VPtoVF`'s memory, it only remains to write the answers out.

First of all, it is convenient to have an abbreviation for output to the `TFM` file:

define *out*(`#`) ≡ *putbyte*(`#`, *tfm_file*)

165.* When a scaled quantity is output, we may need to divide it by *design_units*. The following subroutine takes care of this, using floating point arithmetic only if *design_units* ≠ 1.0.

```
procedure out_scaled(x : fix_word); { outputs a scaled fix_word }
  var n: byte; { the first byte after the sign }
    m: 0 .. 65535; { the two least significant bytes }
  begin if fabs(x/design_units) ≥ 16.0 then
    begin print('The␣relative␣dimension␣'); print_real(x/'4000000,1,3);
    print_ln('␣is␣too␣large.'); print('␣␣(Must␣be␣less␣than␣16*designsize');
    if design_units ≠ unity then
      begin print('␣='); print_real(design_units/'200000,1,3); print('␣designunits');
      end;
    print_ln(')'); x ← 0;
    end;
  if design_units ≠ unity then x ← round((x/design_units) * 1048576.0);
  if x < 0 then
    begin out(255); x ← x + '100000000;
    if x ≤ 0 then x ← 1;
    end
  else begin out(0);
    if x ≥ '100000000 then x ← '77777777;
    end;
  n ← x div '200000; m ← x mod '200000; out(n); out(m div 256); out(m mod 256);
  end;
```

175.* The VF output phase. Output to *vf_file* is considerably simpler.

define *id_byte* = 202 { current version of `VF` format }
define *vout*(`#`) ≡ *putbyte*(`#`, *vf_file*)

⟨ Globals in the outer block 5 ⟩ +≡
vcount: *integer*; { number of bytes written to *vf_file* }

181.* Here is where `VPtoVF` begins and ends.

```
begin initialize;
name_enter;
read_input;
if verbose then print_ln('.');
corr_and_check;
⟨ Do the TFM output 157 ⟩;
vf_output;
if ¬perfect then uexit(1);
end.
```

182.* System-dependent changes. Parse a Unix-style command line.

```
define argument_is(#) ≡ (strcmp(long_options[option_index].name, #) = 0)
⟨ Define parse_arguments 182* ⟩ ≡
procedure parse_arguments;
  const n_options = 3;  { 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;
  begin ⟨ Initialize the option variables 187* ⟩;
  ⟨ Define the option table 183* ⟩;
  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 getopt_return_val = "?" then
        begin usage(my_name);  { getopt has already given an error message. }
        end
      else if argument_is(´help´) then
          begin usage_help(VPTOVF_HELP, nil);
          end
        else if argument_is(´version´) then
            begin print_version_and_exit(banner, nil, ´D.E.␣Knuth´, nil);
            end;  { Else it was a flag; getopt has already done the assignment. }
  until getopt_return_val = −1;  { Now optind is the index of first non-option on the command line. We
        must have one to three remaining arguments. }
  if (optind + 1 ≠ argc) ∧ (optind + 2 ≠ argc) ∧ (optind + 3 ≠ argc) then
    begin write_ln(stderr, my_name, ´:␣Need␣one␣to␣three␣file␣arguments.´); usage(my_name);
    end;
  vpl_name ← extend_filename(cmdline(optind), ´vpl´);
  if optind + 2 ≤ argc then
    begin  { Specified one or both of the output files. }
    vf_name ← extend_filename(cmdline(optind + 1), ´vf´);
    if optind + 3 ≤ argc then
      begin  { Both. }
      tfm_name ← extend_filename(cmdline(optind + 2), ´tfm´);
      end
    else begin  { Just one. }
      tfm_name ← make_suffix(cmdline(optind + 1), ´tfm´);
      end;
    end
  else begin  { Neither. }
    vf_name ← basename_change_suffix(vpl_name, ´.vpl´, ´.vf´);
    tfm_name ← basename_change_suffix(vpl_name, ´.vpl´, ´.tfm´);
    end;
  end;
```

This code is used in section 2*.

183.* Here are the options we allow. The first is one of the standard GNU options.

```
⟨ Define the option table 183* ⟩ ≡
  current_option ← 0; long_options[current_option].name ← ´help´;
  long_options[current_option].has_arg ← 0; long_options[current_option].flag ← 0;
  long_options[current_option].val ← 0; incr(current_option);
```

See also sections 184*, 185*, and 188*.

This code is used in section 182*.

184.* Another of the standard options.

```
⟨ Define the option table 183* ⟩ +≡
  long_options[current_option].name ← ´version´; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← 0; long_options[current_option].val ← 0; incr(current_option);
```

185.* Print progress information?

```
⟨ Define the option table 183* ⟩ +≡
  long_options[current_option].name ← ´verbose´; long_options[current_option].has_arg ← 0;
  long_options[current_option].flag ← address_of(verbose); long_options[current_option].val ← 1;
  incr(current_option);
```

186.* The global variable *verbose* determines whether or not we print progress information.

```
⟨ Globals in the outer block 5 ⟩ +≡
verbose: c_int_type;
```

187.* It starts off *false*.

```
⟨ Initialize the option variables 187* ⟩ ≡
  verbose ← false;
```

This code is used in section 182*.

188.* An element with all zeros always ends the list.

```
⟨ Define the option table 183* ⟩ +≡
  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;
```

189.* Global filenames.

```
⟨ Globals in the outer block 5 ⟩ +≡
vpl_name, tfm_name, vf_name: const_c_string;
```

190.* 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, 2, 3, 6, 22, 24, 31, 32, 33, 89, 118, 144, 152, 153, 156, 165, 175, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190.

`-help`: 183.*
`-verbose`: 185.*
`-version`: 184.*
a: 39.
`A cycle of NEXTLARGER...`: 142.
acc: 60, 61, 62, 63, 64, 65, 72, 74, 76.
address_of: 182,* 185.*
`An "R" or "D" ... needed here`: 72.
argc: 182.*
argument_is: 182.*
argv: 2,* 182.*
ASCII_code: 23, 24,* 36, 44, 46, 60.
`At most 256 VARCHAR specs...`: 120.
backup: 38, 62, 63, 64, 112.
bad_indent: 35.
banner: 1,* 6,* 182.*
basename_change_suffix: 182.*
bc: 158, 159, 160, 163, 164, 169, 177.
bchar: 77, 80, 95, 149, 154, 155, 167, 168, 171.
bchar_label: 82, 84, 112, 139, 145, 154, 168.
boolean: 29, 31,* 41, 50, 72, 77, 113, 128, 138, 150, 158, 167.
`BOT piece of character...`: 141.
both_z: 146, 150, 151, 153.*
boundary_char_code: 52, 55, 95.
buf_size: 3,* 29, 33,* 34.
buffer: 29, 33,* 34, 35, 37, 38, 40, 41, 61.
byte: 23, 39, 52, 53, 60, 66, 77, 82, 83, 89,* 90, 91, 97, 111, 124, 128, 136, 158, 165,* 167, 180.
b0: 66, 67, 68, 104, 114, 115, 116, 117, 121, 141, 145, 149, 155, 171, 172, 178.
b1: 66, 67, 68, 104, 116, 117, 121, 141, 145, 149, 151, 155, 171, 172, 178.
b2: 66, 67, 68, 104, 116, 117, 121, 141, 145, 149, 151, 155, 168, 171, 172, 178.
b3: 66, 67, 68, 104, 116, 117, 121, 141, 145, 149, 151, 155, 168, 171, 172, 178.
c: 69, 83, 91, 124, 150, 180.
`"C" value must be...`: 61.
c_int_type: 182,* 186.*
cc: 124, 127, 150, 151, 167, 169, 170.
char: 24,* 29.
char_dp: 82, 84, 119, 164.
char_dp_code: 52, 55, 119.
char_ht: 82, 84, 119, 164.
char_ht_code: 52, 55, 119.
char_ic: 82, 84, 119, 164.
char_ic_code: 52, 55, 119.
char_info: 164.
char_info_code: 52.
char_info_word: 82.
char_remainder: 82, 84, 112, 119, 120, 140, 141, 142, 149, 154, 164, 167, 169, 170.
char_tag: 82, 84, 111, 112, 119, 120, 140, 142, 154, 164, 169.
char_wd: 82, 84, 85, 118,* 119, 139, 140, 155, 159, 163, 164, 177, 179.
char_wd_code: 52, 55, 103, 119.
`Character cannot be typeset...`: 127.
character_code: 52, 55, 94, 95.
chars_on_line: 31,* 32,* 33,* 137.
check_existence: 140, 149.
check_existence_and_safety: 140, 141.
check_sum_code: 52, 55, 95.
check_sum_loc: 80, 95, 163, 177.
check_sum_specified: 77, 80, 95, 162.
check_tag: 111, 112, 119, 120.
chr: 26, 34.
class: 89,* 147, 150, 153,* 154.
class_var: 89.*
clear_lig_kern_entry: 145.
cmdline: 182.*
coding_scheme_code: 52, 55, 95.
coding_scheme_loc: 80, 95.
comment_code: 52, 55, 94, 103, 106, 110, 119, 121, 125.
const_c_string: 189.*
copy_to_end_of_item: 41, 95, 107, 108, 134.
corr_and_check: 180, 181.*
cur_bytes: 66, 67, 69, 70, 104, 106, 126.
cur_char: 36, 37, 38, 39, 40, 42, 43, 58, 60, 61, 62, 63, 64, 65, 69, 70, 72, 73, 74, 76, 92, 94, 97, 100, 102, 104, 109, 112, 118,* 120, 124, 134.
cur_code: 52, 58, 94, 95, 103, 106, 110, 116, 119, 121, 125, 130, 131, 134.
cur_font: 77, 104, 105, 106, 107, 108, 124, 126, 127, 177, 178, 180.
cur_hash: 47, 50, 51, 53.
cur_name: 46, 50, 51, 53, 54, 58.
current_option: 182,* 183,* 184,* 185,* 188.*
c0: 67, 70, 96, 163.
c1: 67, 70, 96, 163.
c2: 67, 70, 96, 163.
c3: 67, 70, 96, 163.
d: 79, 85, 87, 88, 90.
`Decimal ("D"), octal ("O"), or hex...`: 69.

decr: 4, 38, 40, 41, 50, 58, 76, 90, 97, 102, 117, 124, 128, 133, 150, 152* 159, 169, 170, 171.
delta: 143, 144*
depth: 52, 84, 119, 144* 159, 160.
design_size: 77, 80, 98, 162.
design_size_code: 52, 55, 95.
design_size_loc: 80, 162, 177.
design_units: 77, 80, 99, 106, 128, 163, 165* 179.
design_units_code: 52, 55, 95.
dict_ptr: 44, 45, 53.
dictionary: 44, 50, 53.
do_nothing: 4, 111, 140, 151, 153* 182*
`Don't push so much...`: 132.
double_check_ext: 155.
double_check_lig: 155.
double_check_rep: 155.
double_check_tail: 155.
down1: 122, 131.
ec: 158, 159, 160, 163, 164, 169, 177.
`Empty stack...`: 133.
endif: 152*
enter_name: 53, 54.
eof: 34.
eoln: 34.
equiv: 52, 53, 55, 58.
err_print: 33* 35, 38, 39, 40, 41, 42, 43, 58, 85, 92, 93, 95, 97, 98, 99, 100, 104, 107, 108, 111, 114, 115, 116, 117, 120, 124, 126, 127, 132, 133, 135.
eval: 152* 153*
excess: 88, 89* 90.
existence_tail: 140.
ext_tag: 82, 111, 120, 140.
exten: 77, 120, 121, 141, 155, 172.
extend_filename: 182*
`Extra right parenthesis`: 92.
extra_loc_needed: 167, 168, 170, 171.
f: 152* 153*
fabs: 165*
face_code: 52, 55, 95.
face_loc: 80, 95.
false: 30, 33* 34, 41, 50, 72, 80, 100, 109, 112, 114, 115, 128, 139, 150, 159, 168, 170, 187*
family_code: 52, 55, 95.
family_loc: 80, 95.
farea_length: 77, 105, 108, 178.
farea_start: 77, 105, 108, 178.
`File ended unexpectedly...`: 40.
fill_buffer: 34, 35, 37, 38, 40, 41.
finish_inner_property_list: 102, 104, 109, 118* 120, 124.
finish_the_property: 43, 94, 102, 103, 106, 110, 119, 121, 125.
first_ord: 24* 26.
fix_word: 71, 72, 77, 81, 82, 85, 86, 87, 88, 90, 91, 105, 123, 124, 128, 143, 158, 165*
flag: 183* 184* 185* 188*
flush_error: 42, 94, 103, 106, 110, 119, 121, 125.
fname_length: 77, 105, 107, 178.
fname_start: 77, 105, 107, 178.
fnt_def1: 77, 122, 178.
fnt_num_0: 122, 126.
fnt1: 77, 122, 126.
font_area_code: 52, 56, 106.
font_at: 77, 105, 106, 178.
font_at_code: 52, 56, 106.
font_checksum: 77, 105, 106, 178.
font_checksum_code: 52, 56, 106.
font_dimen_code: 52, 55, 95.
font_dsize: 77, 105, 106, 178.
font_dsize_code: 52, 56, 106.
font_name_code: 52, 56, 106.
font_number: 77, 104, 126.
font_ptr: 77, 80, 104, 105, 126, 127, 177.
`FONTAREA clipped...`: 108.
`FONTNAME clipped...`: 107.
forward: 153*
four_bytes: 66, 67, 69, 77.
fprint_real: 2*
fraction_digits: 75, 76.
frozen_du: 77, 80, 99, 106, 128.
Fuchs, David Raymond: 1*
g: 180.
get_byte: 60, 95, 101, 103, 112, 115, 116, 117, 118* 119, 121, 127.
get_fix: 72, 98, 99, 103, 106, 117, 119, 129, 130, 131.
get_four_bytes: 69, 96, 104, 106, 126.
get_hex: 39, 134.
get_keyword_char: 37, 58.
get_name: 58, 94, 103, 106, 110, 119, 121, 125.
get_next: 38, 39, 42, 43, 58, 60, 61, 62, 63, 64, 65, 69, 70, 72, 73, 74, 76, 92, 97, 100, 102, 104, 109, 112, 118* 120, 124.
getopt: 182*
getopt_long_only: 182*
getopt_return_val: 182*
getopt_struct: 182*
good_indent: 27, 28, 35.
h: 48, 85, 87, 88, 90, 124, 147, 152* 153*
has_arg: 183* 184* 185* 188*
hash: 147, 148, 150, 152* 153* 154.
hash_input: 149, 150.
hash_list: 147, 150, 154, 180.
hash_prime: 47, 48, 49, 50, 51.

hash_ptr: 147, 148, 150, 154.
hash_size: 3,* 147, 148, 150, 152,* 154, 180.
header: 10.
`HEADER indices...`: 101.
header_bytes: 77, 79, 80, 95, 96, 97, 101, 161, 162, 163, 177.
header_code: 52, 55, 95.
header_index: 77, 78, 79, 96, 97.
header_ptr: 77, 80, 101, 159, 162.
height: 52, 81, 84, 119, 144,* 159, 160.
hh: 147, 154, 180.
hstack: 123, 124, 132, 133.
`I can handle only 256...`: 104.
`I had to round...`: 144.*
`I'm out of memory...`: 41.
id_byte: 175,* 177.
ifdef: 152.*
`Illegal character...`: 38, 41.
`Illegal digit`: 70.
`Illegal face code...`: 65.
`Illegal hexadecimal digit`: 39.
incr: 4, 34, 35, 37, 38, 40, 41, 53, 58, 65, 76, 85, 87, 90, 92, 97, 101, 102, 103, 104, 105, 116, 117, 120, 126, 128, 132, 137, 145, 150, 159, 169, 170, 177, 183,* 184,* 185.*
indent: 27, 28, 35.
index: 89,* 90, 164.
index_var: 89.*
indx: 78, 147, 150, 152,* 153.*
`Infinite ligature loop...`: 154.
initialize: 2,* 181.*
input_has_ended: 29, 30, 34, 40, 92.
int_part: 72.
integer: 25, 27, 39, 40, 41, 60, 69, 71, 72, 75, 77, 87, 88, 91, 128, 150, 152,* 175,* 176, 180, 182.*
invalid_code: 26, 38, 41.
italic: 52, 81, 84, 119, 144,* 159, 160, 161, 166.
j: 50, 72, 161.
`Junk after property value...`: 43.
junk_error: 92, 93, 102, 104, 109, 118,* 120, 124.
k: 25, 33,* 50, 53, 88, 91, 97, 124, 128, 180.
kern: 77, 113, 117, 171, 180.
kern_flag: 113, 117, 149, 151, 155.
key: 150, 152.*
kpse_set_program_name: 2.*
`KRN character examined...`: 149.
krn_code: 52, 55, 110.
krn_ptr: 113, 117, 171, 180.
l: 40, 41, 87, 90.
label_code: 52, 55, 110.
label_ptr: 167, 169, 170, 171.
label_table: 167, 169, 170, 171.
last_ord: 24,* 26.
left_ln: 29, 30, 33,* 34.
left_z: 146, 151, 153.*
level: 27, 28, 35, 40, 41, 58, 102, 104, 109, 118,* 120, 124.
lf: 158, 159, 160.
lh: 158, 159, 160.
`LIG character examined...`: 149.
`LIG character generated...`: 149.
lig_code: 52, 55, 110, 116.
lig_exam: 149.
lig_gen: 149.
lig_kern: 77, 114, 115, 116, 117, 145, 147, 149, 151, 155, 168, 171, 180.
lig_ptr: 147, 149, 155, 171, 180.
lig_table_code: 52, 55, 95.
lig_tag: 82, 111, 112, 140, 154, 169.
lig_z: 147, 150, 153.*
limit: 29, 30, 33,* 34, 35, 37, 38, 40, 41.
line: 27, 28, 33,* 34.
link: 81, 82, 84, 85, 87, 90, 166.
list_tag: 82, 111, 119, 140, 142.
lk_offset: 159, 160, 167, 168, 170, 171.
lk_step_ended: 109, 112, 113, 114, 115, 116, 117.
load10: 54, 55, 56, 57.
load11: 54, 55.
load12: 54, 55, 56.
load13: 54, 57.
load14: 54.
load15: 54.
load16: 54, 55.
load17: 54.
load18: 54.
load19: 54.
load20: 54, 57.
load3: 54, 55, 56.
load4: 54, 55, 56, 57.
load5: 54, 55, 57.
load6: 54, 55, 56, 57.
load7: 54, 55, 56, 57.
load8: 54, 55, 56.
load9: 54, 55, 56.
loc: 29, 30, 33,* 34, 35, 37, 38, 40, 41, 58, 61, 92, 102.
long_options: 182,* 183,* 184,* 185,* 188.*
longest_name: 46, 50, 53, 58.
lookup: 50, 53, 58.
m: 87, 90, 165.*
make_suffix: 182.*
map_code: 52, 56, 119.
map_font_code: 52, 56, 95.
max_header_bytes: 3,* 10, 78, 101, 161.

max_kerns: 3,* 77, 113, 117, 180.
max_letters: 44, 50.
max_lig_steps: 3,* 77, 115, 116, 117, 147, 180.
max_name_index: 44, 46, 47, 52.
max_param_words: 3,* 12, 77, 103, 161.
max_stack: 3,* 123, 132.
`Maximum SKIP amount...`: 115.
mem_ptr: 82, 84, 85.
mem_size: 81, 85.
memory: 81, 82, 84, 85, 87, 88, 89,* 90, 158, 159, 160, 161, 163, 166, 179.
`Memory overflow...`: 85.
`MID piece of character...`: 141.
min_cover: 87, 88.
min_nl: 77, 80, 112, 115, 145.
`Missing POP supplied`: 124.
move_down_code: 52, 56, 125, 131.
move_right_code: 52, 56, 125, 130.
my_name: 1,* 2,* 182.*
n: 165.*
n_options: 182.*
name: 182,* 183,* 184,* 185,* 188.*
name_enter: 180, 181.*
name_length: 46, 50, 51, 53, 54, 58.
name_ptr: 46, 50, 58.
ne: 77, 80, 120, 121, 155, 159, 160, 172.
negative: 72, 73, 128.
next_d: 86, 87, 88, 98, 99.
next_larger_code: 52, 55, 119.
nhash: 47, 49, 50, 53.
nk: 77, 80, 117, 159, 160, 171.
nl: 77, 80, 112, 114, 115, 116, 117, 139, 145, 149, 154, 155, 159, 160, 168, 171.
no_tag: 82, 84, 111, 140, 142, 154.
nonblank_found: 41.
not_found: 50, 158, 159.
np: 77, 80, 103, 159, 160, 173.
numbers_differ: 104, 126.
opcode: 128.
optind: 182.*
option_index: 182.*
out: 156,* 160, 162, 164, 165,* 166, 171, 172, 174.
out_scaled: 165,* 166, 171, 173.
out_size: 160, 171.
output: 2.*
p: 85, 87, 90, 150, 161.
packet_length: 77, 80, 124, 179.
packet_start: 77, 80, 124, 179.
par_ptr: 161, 173.
param: 77, 103, 173, 174.
param_enter: 180.
`PARAMETER index must not...`: 103.
parameter_code: 52, 55, 57, 103.
parse_arguments: 2,* 182.*
pending: 146, 147, 153.*
perfect: 31,* 32,* 33,* 181.*
pointer: 81, 82, 85, 87, 88, 89,* 90, 161.
pop: 122, 124, 133.
pop_code: 52, 56, 125.
post: 122, 177.
pre: 122, 177.
print: 2,* 6,* 33,* 136, 137, 140, 142, 144,* 154, 155, 165.*
print_ln: 2,* 6,* 33,* 85, 137, 139, 140, 142, 144,* 154, 155, 165,* 181.*
print_octal: 136, 137, 140, 142, 154, 155.
print_real: 2,* 144,* 165.*
print_version_and_exit: 182.*
push: 122, 132.
push_code: 52, 56, 125.
putbyte: 156,* 175.*
q: 90, 161.
r: 69.
read: 34.
read_BCPL: 95, 97.
read_char_info: 95, 180.
read_four_bytes: 95, 96, 101.
read_input: 180, 181.*
read_lig_kern: 95, 180.
read_ln: 34.
read_packet: 119, 124.
`Real constants must be...`: 72, 74.
`REP piece of character...`: 141.
reset: 6.*
rewritebin: 22.*
right_ln: 29, 30, 33,* 34, 37.
right_z: 146, 151, 153.*
right1: 122, 130.
round: 106, 128, 163, 165,* 179.
round_message: 144.*
rr: 167, 169, 170, 171.
select_font_code: 52, 56, 125.
set_char_code: 52, 56, 125.
set_char_0: 122.
set_indices: 90, 144.*
set_rule: 122, 129.
set_rule_code: 52, 56, 125.
set1: 122, 127, 179.
seven_bit_safe_flag: 77, 80, 100, 139.
seven_bit_safe_flag_code: 52, 55, 95.
seven_flag_loc: 80, 162.
seven_unsafe: 138, 139, 140, 149, 162.
shorten: 88, 144.*
show_error_context: 33.*

simple: 146, 147, 150, 151, 153* 154.
SKIP must follow LIG or KRN: 115.
skip_code: 52, 55, 110.
skip_error: 42, 60, 61, 62, 63, 64, 65, 69, 70, 72, 74, 101.
skip_to_end_of_item: 40, 42, 43, 94, 102, 103, 104, 106, 109, 110, 118* 119, 120, 121, 124, 125.
skip_to_paren: 42, 93, 100, 112.
Sorry, I don't know...: 58.
Sorry, I haven't room...: 154.
Sorry, it's too late...: 99.
Sorry, LIGTABLE too long...: 115, 116, 117.
Sorry, the maximum...: 70.
Sorry, too many different kerns...: 117.
sort_in: 85, 118* 119, 140, 155.
sort_ptr: 167, 169, 170, 171.
Special command being clipped...: 135.
special_code: 52, 56, 125, 134.
special_hex_code: 52, 56, 125.
special_start: 124, 134, 135.
stack_ptr: 123, 124, 130, 131, 132, 133.
start: 44, 45, 46, 47, 50, 52, 53.
start_ptr: 44, 45, 53.
stderr: 2* 182*
STOP must follow LIG or KRN: 114.
stop_code: 52, 55, 110.
stop_flag: 113, 114, 145, 149.
strcmp: 182*
String is too long...: 97.
system dependencies: 2* 22* 24* 34.
t: 60, 128, 150, 167.
tail: 54.
temp_width: 158, 163.
text: 5.
tfm_file: 2* 21, 22* 156*
tfm_name: 22* 182* 189*
The character NEXTLARGER...: 140.
The design size must...: 98.
The flag value should be...: 100.
The font is not...safe: 139.
The number of units...: 99.
The relative dimension...: 165*
There's junk here...: 93.
This character already...: 111.
This HEADER index is too big...: 101.
This PARAMETER index is too big...: 103.
This property name doesn't belong...: 94, 103, 106, 110, 119, 121, 125.
This value shouldn't...: 62, 63, 64.
TOP piece of character...: 141.
true: 30, 32* 34, 41, 50, 95, 100, 106, 116, 117, 128, 140, 149, 150, 159, 168.
tt: 147, 154.
t1: 54.
t10: 54.
t11: 54.
t12: 54.
t13: 54.
t14: 54.
t15: 54.
t16: 54.
t17: 54.
t18: 54.
t19: 54.
t2: 54.
t20: 54.
t3: 54.
t4: 54.
t5: 54.
t6: 54.
t7: 54.
t8: 54.
t9: 54.
uexit: 181*
Undefined MAPFONT...: 126.
unity: 71, 72, 80, 98, 106, 128, 163, 165* 179.
UNSPECIFIED: 80.
Unused KRN step...: 155.
Unused LIG step...: 155.
Unused VARCHAR...: 155.
usage: 182*
usage_help: 182*
v: 124.
val: 183* 184* 185* 188*
var_char_code: 52, 55, 119, 121.
vcount: 175* 177, 178, 179.
verbose: 6* 118* 181* 185* 186* 187*
version_string: 6*
vf: 3* 41, 77, 134, 135, 177, 178, 179.
vf_file: 2* 21, 22* 175*
vf_fix: 128, 129, 130, 131.
vf_name: 22* 182* 189*
vf_output: 180, 181*
vf_ptr: 41, 77, 80, 95, 107, 108, 124, 134, 135.
vf_size: 3* 41, 77, 80, 105, 124, 135, 178, 179.
vf_store: 41, 124, 126, 127, 128, 129, 130, 131, 132, 133, 134.
virtual_title_code: 52, 56, 95.
vout: 175* 176, 177, 178, 179.
vout_int: 176, 178, 179.
vpl_enter: 180.
vpl_file: 2* 5, 6* 34.
vpl_name: 6* 182* 189*
VPtoVF: 2*

VPTOVF_HELP: 182*
vstack: 123, 124, 132, 133.
`VTITLE clipped...`: 95.
vtitle_length: 77, 80, 95, 177.
vtitle_start: 77, 80, 95, 177.
`Warning: Inconsistent indentation...`: 35.
`Warning: Indented line...`: 35.
width: 52, 81, 84, 85, 118* 119, 140, 144* 155, 158, 159, 160, 161, 166.
write: 2*
write_ln: 2* 182*
wstack: 123, 130.
w0: 122, 130.
w1: 122, 130.
x: 91, 124, 128, 152* 153* 176.
x_lig_cycle: 147, 153* 154.
xord: 24* 26, 34, 37, 38, 41, 61.
xstack: 123, 130.
xxx1: 122, 124, 134.
xxx4: 122, 135.
x0: 122, 130.
x1: 122, 130.
y: 150, 152* 153*
y_lig_cycle: 147, 148, 153* 154.
`You need "C" or "D" ...here`: 60.
ystack: 123, 131.
y0: 122, 131.
y1: 122, 131.
zero_bytes: 67, 68, 69, 70, 105, 120.
zstack: 123, 131.
zz: 150, 151.
z0: 122, 131.
z1: 122, 131.

⟨ Assemble a font selection 126 ⟩ Used in section 125.
⟨ Assemble a horizontal movement 130 ⟩ Used in section 125.
⟨ Assemble a rulesetting instruction 129 ⟩ Used in section 125.
⟨ Assemble a special command 134 ⟩ Used in section 125.
⟨ Assemble a stack pop 133 ⟩ Used in section 125.
⟨ Assemble a stack push 132 ⟩ Used in section 125.
⟨ Assemble a typesetting instruction 127 ⟩ Used in section 125.
⟨ Assemble a vertical movement 131 ⟩ Used in section 125.
⟨ Check for infinite ligature loops 154 ⟩ Used in section 139.
⟨ Check ligature program of c 149 ⟩ Used in sections 139 and 140.
⟨ Check the pieces of $exten[c]$ 141 ⟩ Used in section 140.
⟨ Compute the check sum 163 ⟩ Used in section 162.
⟨ Compute the command parameters y, cc, and zz 151 ⟩ Used in section 150.
⟨ Compute the hash code, *cur_hash*, for *cur_name* 51 ⟩ Used in section 50.
⟨ Compute the ligature/kern program offset 168 ⟩ Used in section 159.
⟨ Compute the twelve subfile sizes 159 ⟩ Used in section 157.
⟨ Constants in the outer block 3* ⟩ Used in section 2*.
⟨ Convert *xxx1* command to *xxx4* 135 ⟩ Used in section 134.
⟨ Correct and check the information 139 ⟩ Used in section 180.
⟨ Declare the *vf_fix* procedure 128 ⟩ Used in section 124.
⟨ Define the option table 183*, 184*, 185*, 188* ⟩ Used in section 182*.
⟨ Define *parse_arguments* 182* ⟩ Used in section 2*.
⟨ Do the TFM output 157 ⟩ Used in section 181*.
⟨ Do the VF output 177 ⟩ Used in section 180.
⟨ Doublecheck the lig/kern commands and the extensible recipes 155 ⟩ Used in section 139.
⟨ Enter all the PL names and their equivalents, except the parameter names 55 ⟩ Used in section 180.
⟨ Enter all the VPL names 56 ⟩ Used in section 180.
⟨ Enter the parameter names 57 ⟩ Used in section 180.
⟨ Find the minimum *lk_offset* and adjust all remainders 170 ⟩ Used in section 168.
⟨ For all characters g generated by c, make sure that $char_wd[g]$ is nonzero, and set *seven_unsafe* if $c < 128 \le g$ 140 ⟩ Used in section 139.
⟨ Globals in the outer block 5, 21, 24*, 27, 29, 31*, 36, 44, 46, 47, 52, 67, 75, 77, 82, 86, 89*, 91, 113, 123, 138, 143, 147, 158, 161, 167, 175*, 186*, 189* ⟩ Used in section 2*.
⟨ Initialize a new local font 105 ⟩ Used in section 104.
⟨ Initialize the option variables 187* ⟩ Used in section 182*.
⟨ Insert all labels into *label_table* 169 ⟩ Used in section 168.
⟨ Local variables for initialization 25, 48, 79, 83 ⟩ Used in section 2*.
⟨ Make sure that c is not the largest element of a charlist cycle 142 ⟩ Used in section 139.
⟨ Make sure the ligature/kerning program ends appropriately 145 ⟩ Used in section 139.
⟨ Multiply by 10, add *cur_char* − "0", and *get_next* 74 ⟩ Used in section 72.
⟨ Multiply by r, add *cur_char* − "0", and *get_next* 70 ⟩ Used in section 69.
⟨ Output a local font definition 178 ⟩ Used in section 177.
⟨ Output a packet for character c 179 ⟩ Used in section 177.
⟨ Output the character info 164 ⟩ Used in section 157.
⟨ Output the dimensions themselves 166 ⟩ Used in section 157.
⟨ Output the extensible character recipes 172 ⟩ Used in section 157.
⟨ Output the header block 162 ⟩ Used in section 157.
⟨ Output the ligature/kern program 171 ⟩ Used in section 157.
⟨ Output the parameters 173 ⟩ Used in section 157.
⟨ Output the slant ($param[1]$) without scaling 174 ⟩ Used in section 173.
⟨ Output the twelve subfile sizes 160 ⟩ Used in section 157.
⟨ Print c in octal notation 137 ⟩ Used in section 118*.

⟨ Put the width, height, depth, and italic lists into final form 144* ⟩ Used in section 139.
⟨ Read a character property 119 ⟩ Used in section 118*.
⟨ Read a font property value 94 ⟩ Used in section 92.
⟨ Read a kerning step 117 ⟩ Used in section 110.
⟨ Read a label step 112 ⟩ Used in section 110.
⟨ Read a ligature step 116 ⟩ Used in section 110.
⟨ Read a ligature/kern command 110 ⟩ Used in section 109.
⟨ Read a local font area 108 ⟩ Used in section 106.
⟨ Read a local font list 104 ⟩ Used in section 95.
⟨ Read a local font name 107 ⟩ Used in section 106.
⟨ Read a local font property 106 ⟩ Used in section 104.
⟨ Read a parameter value 103 ⟩ Used in section 102.
⟨ Read a skip step 115 ⟩ Used in section 110.
⟨ Read a stop step 114 ⟩ Used in section 110.
⟨ Read all the input 92 ⟩ Used in section 180.
⟨ Read an extensible piece 121 ⟩ Used in section 120.
⟨ Read an extensible recipe for *c* 120 ⟩ Used in section 119.
⟨ Read an indexed header word 101 ⟩ Used in section 95.
⟨ Read and assemble a list of `DVI` commands 125 ⟩ Used in section 124.
⟨ Read character info list 118* ⟩ Used in section 180.
⟨ Read font parameter list 102 ⟩ Used in section 95.
⟨ Read ligature/kern list 109 ⟩ Used in section 180.
⟨ Read the design size 98 ⟩ Used in section 95.
⟨ Read the design units 99 ⟩ Used in section 95.
⟨ Read the font property value specified by *cur_code* 95 ⟩ Used in section 94.
⟨ Read the seven-bit-safe flag 100 ⟩ Used in section 95.
⟨ Scan a face code 65 ⟩ Used in section 60.
⟨ Scan a small decimal number 62 ⟩ Used in section 60.
⟨ Scan a small hexadecimal number 64 ⟩ Used in section 60.
⟨ Scan a small octal number 63 ⟩ Used in section 60.
⟨ Scan an ASCII character code 61 ⟩ Used in section 60.
⟨ Scan the blanks and/or signs after the type code 73 ⟩ Used in section 72.
⟨ Scan the fraction part and put it in *acc* 76 ⟩ Used in section 72.
⟨ Set initial values 6*, 22*, 26, 28, 30, 32*, 45, 49, 68, 80, 84, 148 ⟩ Used in section 2*.
⟨ Set *loc* to the number of leading blanks in the buffer, and check the indentation 35 ⟩ Used in section 34.
⟨ Types in the outer block 23, 66, 71, 78, 81 ⟩ Used in section 2*.