The VPtoVF processor

(Version 1.6, January 2014)
**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 \textit{\LaTeX} 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 \textit{\LaTeX}.

**VPtoVF** is an extended version of the program **PLtoTF**, which is part of the standard \textit{\LaTeX}ware 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.

```pascal
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 \textit{vpl} file, and the output is written on \textit{vf} file and \textit{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 \textit{print} is used instead of \textit{write} when this program writes on the output file, so that all such output can be easily deflected.

```pascal
define print(#) ≡ write(stderr,#)
define print_ln(#) ≡ writeLn(stderr,#)
define print_real(#) ≡ fprint_real(stderr,#)
```

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

```pascal
(Constats in the outer block \textit{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;  { the 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 \textit{2}. 

\[\]
(Set initial values $6^*$) ≡

```c
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 $6^*$) +≡

```c
rewritebin(vf_file, vf_name); rewritebin(tfm_file, tfm_name);
```
24. One of the things \texttt{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 \texttt{TFM} file. An array \texttt{xord} is used to do the conversion from \texttt{char}; the method below should work with little or no change on most Pascal systems.

\begin{verbatim}
\textbf{define char} \equiv 0 \ldots 255
\textbf{define first\_ord} = 0 \quad \{ \text{ordinal number of the smallest element of char} \}
\textbf{define last\_ord} = 127 \quad \{ \text{ordinal number of the largest element of char} \}
\end{verbatim}

\begin{verbatim}
\langle \text{Globals in the outer block} \rangle +\equiv
\texttt{xord: array [\text{char}] of ASCII\_code}; \quad \{ \text{conversion table} \}
\end{verbatim}

31. Just before each \texttt{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.

\begin{verbatim}
\langle \text{Globals in the outer block} \rangle +\equiv
\texttt{chars\_on\_line: 0 \ldots 8; \quad \{ \text{the number of characters printed on the current line} \}}
\texttt{perfect: boolean; \quad \{ \text{was the file free of errors?} \}}
\end{verbatim}

32. (Set initial values \texttt{6+}) +\equiv
\begin{verbatim}
\texttt{chars\_on\_line} \leftarrow 0; \texttt{perfect} \leftarrow \texttt{true}; \quad \{ \text{innocent until proved guilty} \}
\end{verbatim}

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.

\begin{verbatim}
\textbf{define err\_print\_#} \equiv
\begin{verbatim}
\textbf{begin if chars\_on\_line} > 0 then print\_ln(\texttt{´\ldots´});
\textbf{print\_#}; \textbf{show\_error\_context};
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\textbf{procedure show\_error\_context}; \quad \{ \text{prints the current scanner location} \}
\begin{verbatim}
\textbf{var k: 0 \ldots buf\_size; \quad \{ \text{an index into buffer} \}}
\textbf{begin print\_ln(´\texttt{line\_line}, line : 1, \texttt{´\ldots´});}\texttt{print\_ln(´\texttt{...´});}
\textbf{if \texttt{!left\_ln} then print(´\texttt{...´});}
\textbf{for k \leftarrow 1 to loc do print(buffer[k]); \quad \{ \text{print the characters already scanned} \}}
\textbf{print\_ln(´\texttt{...´);}\texttt{if \texttt{!left\_ln} then print(´\texttt{...´});}
\textbf{for k \leftarrow 1 to loc do print(´\texttt{...´}); \quad \{ \text{space out the second line} \}}
\textbf{for k \leftarrow loc + 1 to limit do print(buffer[k]); \quad \{ \text{print the characters yet unseen} \}}
\textbf{if \texttt{right\_ln} then print\_ln(´\texttt{...´}) else print\_ln(´\texttt{...´});}
\texttt{chars\_on\_line} \leftarrow 0; \texttt{perfect} \leftarrow \texttt{false};
\end{verbatim}
\end{verbatim}
When we are nearly ready to output the TFM file, we will set $\text{index}[p] \leftarrow k$ if the dimension in $\text{memory}[p]$ is being rounded to the $k$th element of its list.

```
define index ≡ index_var
define class ≡ class_var
```

(Global s in the outer block 5) +≡

```
index: array [pointer] of byte;
excess: byte;    { number of words to remove, if list is being shortened }
```
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.
\[ 144^* \textbf{define } \text{round\_message}(\#) \equiv \\
\text{if } \delta > 0 \text{ then} \\
\begin{align*}
&\text{begin } \text{println}(\text{"I\_had\_to\_round\_some\_\#, \_s\_by\_"}); \\
&\quad \text{print\_real}(((\delta + 1) \text{ div } 2)/'4000000), 1, 7); \text{ println("\_units."); }
&\text{end}
\end{align*}
\]

(\text{Put the width, height, depth, and italic lists into final form} \[ 144^* \equiv \\
\delta \leftarrow \text{shorten}(\text{width}, 255); \text{ set\_indices}(\text{width}, \delta); \text{ round\_message}(\text{"width"}); \\
\delta \leftarrow \text{shorten}(\text{height}, 15); \text{ set\_indices}(\text{height}, \delta); \text{ round\_message}(\text{"height"}); \\
\delta \leftarrow \text{shorten}(\text{depth}, 15); \text{ set\_indices}(\text{depth}, \delta); \text{ round\_message}(\text{"depth"}); \\
\delta \leftarrow \text{shorten}(\text{italic}, 63); \text{ set\_indices}(\text{italic}, \delta); \text{ round\_message}(\text{"italic\_correction"});
\]

This code is used in section 139.

\[ 152^* \text{(More good stuff from TToFPL.)} \\
\textbf{ifdef (\text{"notdef"})} \\
\textbf{function } f(h, x, y : indx): indx; \\
\text{begin end;} \\
\quad \{ \text{compute } f \text{ for arguments known to be in } \text{hash}[h] \} \\
\textbf{endif (\text{"notdef"})} \\
\textbf{function } \text{eval}(x, y : indx): indx; \quad \{ \text{compute } f(x, y) \text{ with hashtable lookup} \}
\]

\begin{align*}
&\text{var } \text{key: integer}; \quad \{ \text{value sought in hash table} \}
&\text{begin key } \leftarrow 256 * x + y + 1; \quad h \leftarrow (1009 * \text{key}) \mod \text{hash\_size}; \\
&\text{while } \text{hash}[h] > \text{key} \text{ do} \\
&\quad \text{if } h > 0 \text{ then } \text{decr}(h) \text{ else } h \leftarrow \text{hash\_size}; \\
&\quad \text{if } \text{hash}[h] < \text{key} \text{ then } \text{eval } \leftarrow y \quad \{ \text{not in ordered hash table} \}
&\quad \text{else } \text{eval } \leftarrow f(h, x, y); \\
&\text{end;}
\end{align*}

\[ 153^* \text{Pascal’s beastly convention for forward declarations prevents us from saying } \textbf{function } f(h, x, y : indx): indx; \text{ indx here.} \\
\textbf{function } f(h, x, y : indx): indx; \\
\text{begin case class}[h] of \\
\text{simple: do\_nothing;} \\
\text{left\_z: begin class}[h] \leftarrow \text{pending}; \text{ lig\_z}[h] \leftarrow \text{eval}(\text{lig\_z}[h], y); \text{ class}[h] \leftarrow \text{simple}; \\
\text{end;} \\
\text{right\_z: begin class}[h] \leftarrow \text{pending}; \text{ lig\_z}[h] \leftarrow \text{eval}(x, \text{lig\_z}[h]); \text{ class}[h] \leftarrow \text{simple}; \\
\text{end;} \\
\text{both\_z: begin class}[h] \leftarrow \text{pending}; \text{ lig\_z}[h] \leftarrow \text{eval}(\text{eval}(x, \text{lig\_z}[h]), y); \text{ class}[h] \leftarrow \text{simple}; \\
\text{end;} \\
\text{pending: begin } x\_\text{lig\_cycle } \leftarrow x; \quad y\_\text{lig\_cycle } \leftarrow y; \quad \text{ lig\_z}[h] \leftarrow 257; \text{ class}[h] \leftarrow \text{simple}; \\
\text{end;} \quad \{ \text{the value 257 will break all cycles, since it’s not in } \text{hash} \}
\]

\text{end;} \quad \{ \text{there are no other cases} \}
\]

\text{f } \leftarrow \text{lig\_z}[h]; \\
\text{end;}

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:

\[
\text{define } \text{out}(\#) \equiv \text{putbyte}(\#, \text{tfm\_file})
\]

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 \(\neq 1.0\).

\[
\text{procedure } \text{out\_scaled}(x : \text{fix\_word}); \quad \{ \text{outputs a scaled fix\_word} \}
\]

\[
\text{begin}
\quad \text{var } n: \text{byte}; \quad \{ \text{the first byte after the sign} \}
\quad m: 0..65535; \quad \{ \text{the two least significant bytes} \}
\quad \text{if } \text{fabs}(x/\text{design\_units}) \geq 16.0 \text{ then}
\quad \text{begin}
\quad \quad \text{print(’The relative dimension’);} \quad \text{print\_real}(x/’4000000,1,3);
\quad \quad \text{print\_ln(’is too large.’);} \quad \text{print(’Must be less than 16*designsize’);}\n\quad \quad \text{if } \text{design\_units \neq unity} \text{ then}
\quad \quad \quad \text{begin}
\quad \quad \quad \quad \text{print(’=’);} \quad \text{print\_real(\text{design\_units}/’200000,1,3);} \quad \text{print(’designunits’);}\n\quad \quad \quad \end{end}
\quad \quad \text{print\_ln(’’);} \quad x \leftarrow 0;
\quad \quad \text{end}
\quad \quad \text{if } \text{design\_units \neq unity} \text{ then } x \leftarrow \text{round}((x/\text{design\_units}) \ast 1048576.0);
\quad \text{end}
\text{if } x < 0 \text{ then}
\quad \text{begin}
\quad \quad \text{out}(255); \quad x \leftarrow x + ’1000000000’;
\quad \quad \text{if } x \leq 0 \text{ then } x \leftarrow 1;
\quad \text{end}
\text{else begin}
\quad \text{out}(0);
\quad \quad \text{if } x \geq ’1000000000 \text{ then } x \leftarrow ’77777777’;
\quad \text{end}
\quad \text{n} \leftarrow x \text{ div } ’200000’; \quad m \leftarrow x \text{ mod } ’200000’; \quad \text{out}(n); \quad \text{out}(m \text{ div } 256); \quad \text{out}(m \text{ mod } 256);
\text{end}
\text{end}
The VF output phase. Output to \textit{vf\_file} is considerably simpler.

\begin{verbatim}
  define \textit{id\_byte} = 202 \{ current version of VF format \}
  define \textit{vout}(\#) \equiv \textit{putbyte}(\#, \textit{vf\_file})
\end{verbatim}

(Global in the outer block 5) +\equiv
\begin{verbatim}
\textit{vcount}: \textit{integer}; \{ number of bytes written to \textit{vf\_file} \}
\end{verbatim}
181* Here is where VPtoVF begins and ends.

```
begin initialize;
  name_enter;
  read_input;
  if verbose then print_in(".");
  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;
g 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 argument_is(´-help´) then
begin usage(my_name);
else if argument_is(´-version´) then
begin print_version_and_exit(banner, nil, ´D.E. Knuth´, nil);
end
else if getopt_return_val = ´?´ then
begin usage(help(VPTOVF_HELP, nil);
else
begin
{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_in(stderr, my_name, ´: Need one to three file arguments.´); usage(my_name);
end;
⟨Specify one or both of the output files.⟩
vpl.name ← extend_filename(cmdline(optind), ´.vpl´);
if optind + 2 ≤ argc then
begin {Both.}
    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
This code is used in section 2*.}
Here are the options we allow. The first is one of the standard GNU options.

(Define the option table 183*) \(\equiv\)
\[
\begin{align*}
  &\textit{current_option} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{name} \leftarrow \texttt{help}; \\
  &\textit{long_options}[\textit{current_option}].\textit{has_arg} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{flag} \leftarrow 0; \\
  &\textit{long_options}[\textit{current_option}].\textit{val} \leftarrow 0; \text{incr}(\textit{current_option});
\end{align*}
\]

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

This code is used in section 182*.

Another of the standard options.

(Define the option table 183*) \(\doteq\)
\[
\begin{align*}
  &\textit{long_options}[\textit{current_option}].\textit{name} \leftarrow \texttt{version}; \textit{long_options}[\textit{current_option}].\textit{has_arg} \leftarrow 0; \\
  &\textit{long_options}[\textit{current_option}].\textit{flag} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{val} \leftarrow 0; \text{incr}(\textit{current_option});
\end{align*}
\]

Print progress information?

(Define the option table 183*) \(\doteq\)
\[
\begin{align*}
  &\textit{long_options}[\textit{current_option}].\textit{name} \leftarrow \texttt{verbose}; \textit{long_options}[\textit{current_option}].\textit{has_arg} \leftarrow 0; \\
  &\textit{long_options}[\textit{current_option}].\textit{flag} \leftarrow \text{address_of}(\textit{verbose}); \textit{long_options}[\textit{current_option}].\textit{val} \leftarrow 1; \text{incr}(\textit{current_option});
\end{align*}
\]

The global variable \texttt{verbose} determines whether or not we print progress information.

(Global options in the outer block) \(\doteq\)
\[
\text{verbose: c_int_type;}
\]

It starts off \texttt{false}.

(Initialize the option variables) \(\equiv\)
\[
\text{verbose} \leftarrow \texttt{false};
\]

This code is used in section 182*.

An element with all zeros always ends the list.

(Define the option table 183*) \(\doteq\)
\[
\begin{align*}
  &\textit{long_options}[\textit{current_option}].\textit{name} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{has_arg} \leftarrow 0; \\
  &\textit{long_options}[\textit{current_option}].\textit{flag} \leftarrow 0; \textit{long_options}[\textit{current_option}].\textit{val} \leftarrow 0;
\end{align*}
\]

Global filenames.

(Global options in the outer block) \(\doteq\)
\[
\text{vpl_name, tfm_name, vf_name: const\_c\_string;}
\]
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.

crc: 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*


bchar: 77, 80, 95, 149, 154, 155, 167, 168, 171.

bchar_label: 82, 84, 112, 139, 145, 154, 168.


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.


c: 69, 83, 91, 124, 150, 180.

"C" value must be...: 61.

c_int_type: 182*, 186*


char: 24*, 29.

char_dp: 82, 84, 119, 164.

char_dp_code: 52, 55, 119.

char_ft: 82, 84, 119, 164.

char_ft_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_ud: 82, 84, 85, 118*, 119, 139, 140, 155, 159, 163, 164, 177, 179.

char_ud_code: 52, 55, 103, 119.

Character cannot be typeset...: 127.

cchar_code: 52, 55, 94, 95.

cchar_on_line: 31*, 32*, 33*, 137.

check_existence: 140, 149.

check_existence_and_set: 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.

cons_c_string: 189*


corr_and_check: 180, 181*


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_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.
Don't push so much...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
I had to round...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I'm out of memory...
I can handle only 256...
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.

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_larger_code: 52, 55, 119.

Real constants must be...: 72, 74.
REP piece of character...: 141.
reset: 6*.
rewritebin: 22*.
right_ln: 80, 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_parent: 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: 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: 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.
ucexit: 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.
version_string: 6*.
vf: 3* 41, 77, 134, 135, 177, 178, 179.
vf_file: 2* 21, 22* 175*.
vf_name: 22* 182* 189*.
vf_output: 180, 181*.
vf_size: 3* 41, 77, 80, 105, 124, 135, 178, 179.
virtual_title_code: 52, 56, 95.
vout: 175* 176, 177, 178, 179.
vout_int: 176, 178, 179.
vpl_enter: 180.
vpl_file: 2* 5, 34* 34.
vpl_name: 6* 182* 189*.
VPtrvVF: 2*.
VPTOVF_HELP: 182*

vstack: 123, 124, 132, 133.

VTITLE clipped...: 95.

title_length: 77, 80, 95, 177.

title_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*

vstack: 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.

gstack: 123, 131.

y0: 122, 131.

y1: 122, 131.

zero_bytes: 67, 68, 69, 70, 105, 120.

gstack: 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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
(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.
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 \textit{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 \textit{acc} \[76\]) Used in section 72.

(Scan the seven-bit-safe flag \[100\]) Used in section 95.

(Scan the blanks and/or signs after the type code \[73\]) Used in section 72.

(Scan the fraction part and put it in \textit{acc} \[76\]) Used in section 72.

(Scan the blank and/or signs after the type code \[73\]) Used in section 72.

(Read character property \[119\]) Used in section 118*.

(Read a character property \[119\]) Used in section 118*.

(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 \textit{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 \textit{acc} \[76\]) Used in section 72.

(Scan the seven-bit-safe flag \[100\]) Used in section 95.

(Scan the blanks and/or signs after the type code \[73\]) Used in section 72.

(Scan the fraction part and put it in \textit{acc} \[76\]) Used in section 72.

(Scan the seven-bit-safe flag \[100\]) Used in section 95.