December 24, 2011 at 11:17

1. Introduction.

DVLABEL

Creates TeX source code for typesetting labels for digital video tapes
(Version 1.6 of August 10, 2005)

Written by Fredrik Jonsson

This CWEB[†] program generates TEX source code for typesetting of labels for digital video tapes (DV format, typically used for hand-held video camera recorders).

Most people have not heard of the TEX system for typesetting mathematical text, and if they have, they will most probably not use the system for creating things like labels for video tapes. However, the output generated by TEX is in many cases absolutely superior in balance and visual clarity, and in order to benefit from the compactness and beauty of text processed by TEX even for such basic things as labels for video tapes, this program generates the necessary TEX code and even compiles it into printable PostScript.

The program is primarily designed to run in interactive mode, but via command-line parameters it also supports batch-mode operation. A feature of the program is also that it is able to compile the generated TeX source code into PostScript, using the DVIPS program.[‡]

[†] For information on the CWEB programming language by Donald E. Knuth, as well as samples of CWEB programs, see `http://www-cs-faculty.stanford.edu/~knuth/cweb.html`. For general information on literate programming, see `http://www.literateprogramming.com`.

[‡] The DVIPS program is copyrighted by Radical Eye Software, but is typically included in most TEX distributions under UNIX, Linux, and Microsoft Windows; see the site `http://www.radicaleye.com` for further information.

2. Revision history of the program.

2003-12-28 [v.1.0] <jonsson@uni-wuppertal.de>
First properly working version of the DVLABEL program. I got the idea of creating this program from the `audio-tape.ps` PostScript code by Jamie Zawinski. This code is a splendid example of how one can write a simple PostScript program with the help of a regular ASCII editor, and by sending the PostScript program to the printer, one gets a neat printout to be used for tape cassette labels, DAT, or video tapes. However, whenever one has a new cassette to be labeled, one has to edit the PostScript source, and for a rookie on PostScript programming this task is somewhat inconvenient. Therefore, I decided to create something similar, but with a standalone program that could be operated either in interactive mode, with the program asking for the specific information to be entered in the label, or in batch mode. However, instead of directly generating PostScript, I decided to go for a language that I know somewhat more in detail, namely plain TEX, which also has the benefit of being a language which the author, Donald E. Knuth, has decided to keep fixed in order to ensure future compatibility.

2003-12-29 [v.1.1] <jonsson@uni-wuppertal.de>
Revised the leading blocks (definitions) of the generator of the TEX-code, in order to have a more clearly structure of the labels. Included the `\boxit` example from the TEX-book to have the face, flap, and spine of the labels neatly structured. Also finished the parsing engine for the address and table of contents fields.

2004-11-26 [v.1.2] <fredrik.jonsson@nmrc.ie>
Added support for supplying DVIPS options via the command line when invoking DVLABEL. In order to parse for an arbitrary number of DVIPS options at the command line, it is important to enclose the list of DVIPS options by quotes, hence forcing the DVLABEL program to parse the set of options as one single string of characters. (The quotes are only necessary if the number of DVIPS options are two or more.) Hence, for example, to force DVIPS to generate output pages of US letter format and at a resolution of 720 dpi one could invoke DVLABEL with `dvlabel --dvipsopts "-tletter -D720" ...` Also slightly changed the way the input filename is used; now one can specify only the basic filename, to which the DVLABEL program now automatically appends the suffix `.dvl`, if necessary. Finally restructured the blocks related to scanning and saving code for individual DV labels, in order to have the structure ready for making the program generally capable of scanning one single file containing an arbitrary number of labels, as for instance if one would keep all records in one single text file.

2004-11-27 [v.1.3] <fredrik.jonsson@nmrc.ie>
The DVLABEL program now parses multiple label records from the same input file. Also added the useful feature of crop marks in the generated TEX output of the program. In its current state, the program only generates labels in one single horizontal column; this is something that I in the future versions will change into a $[3 \times 3]$ array of labels, as soon as I get the *v*`box` and *h*`box` statements of the *b*`oxit` definition right.

2004-12-28 [v.1.4] <fredrik.jonsson@nmrc.ie>
Added the `--headline` and `--linethick` options. Fixed a bug in the page output, which previously caused TEX to complain about vertical underfill of the generated pages.

2005-01-01 [v.1.5] <fredrik.jonsson@nmrc.ie>
Added the `--cropmark` and `--edgeseparation` options. Also fixed a bug in the vertical label dimensions. Added the *log*( ) way of displaying log and error messages.

2005-08-10 [v.1.6] <fj@phys.soton.ac.uk>
Back in Southampton with my family after a hot summer. Wrote the code for the *strip_away_path*( ) routine originally for the POINCARE program and immediately decided to adopt the code also into the DVLABEL and MAGBRAGG programs in order

to finally solve the problem with long path strings that appear in the program name string whenever poincare is called with an explicit path specified at the command line. The call to the *strip_away_path*( ) routine is located in the beginning of the block for command line parsing.

3. Compiling the source code. The program is written in CWEB, generating ANSI-C conforming source code and documentation as TEX-source, and is to be compiled using the enclosed Makefile, leaving an executable file `dvlabel`† and a PostScript file `dvlabel.ps` (the document you currently are reading), which contains the full documentation of the program:

```
#
# Makefile designed for use with ctangle, cweave, gcc, and plain TeX.
#
# Copyright (C) 2003, Fredrik Jonsson <jonsson@uni-wuppertal.de>
#
CTANGLE  = ctangle
CC       = gcc
CCOPTS   = -O2 -Wall -ansi -pedantic # follow ISO C89 (ANSI) strictly
LNOPTS   = -lm
CWEAVE   = cweave
TEX      = tex
DVIPS    = dvips
DVIPSOPT = -ta4 -D1200

all:  dvlabel.exe dvlabel.ps

dvlabel.exe:  dvlabel.o # generate the executable file
        $(CC) $(CCOPTS) -o dvlabel dvlabel.o $(LNOPTS)

dvlabel.o:  dvlabel.c # generate the object file
        $(CC) $(CCOPTS) -c dvlabel.c

dvlabel.c:  dvlabel.w # generate C code from the CWEB source
        $(CTANGLE) dvlabel

dvlabel.ps:  dvlabel.dvi # generate the PostScript documentation
        $(DVIPS) $(DVIPSOPT) dvlabel.dvi -o dvlabel.ps

dvlabel.dvi:  dvlabel.tex # generate the device-independent documentation
        $(TEX) dvlabel.tex

dvlabel.tex:  dvlabel.w # generate plain TeX code from the CWEB source
        $(CWEAVE) dvlabel

clean:
        -rm -Rf *.c *.o *.exe
        -rm -Rf *.tex *.aux *.log *.toc *.idx *.scn *.dvi
```

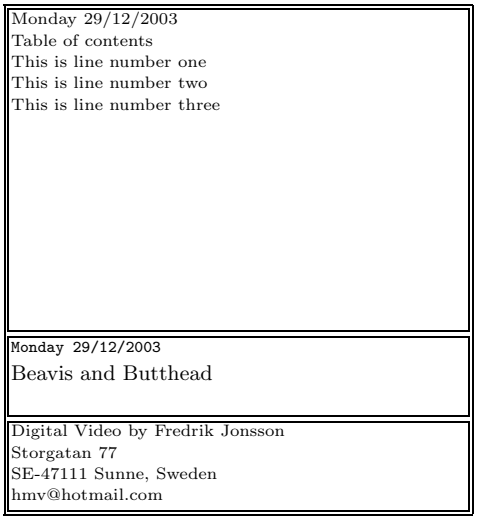
† On platforms running Windows NT, Windows 2000, or any other operating system by Microsoft, the executable file will instead automatically be called `dvlabel.exe`.

4. Running the program. The program is entirely controlled by the command line options supplied when invoking the program. To start the program in interactive mode, simply run `dvlabel` from the command line, without any command line options.

Example I: In the following excerpt from an interactive terminal session, where a leading asterisk (*) in a line indicates the user-supplied input to the program, a DV label is generated as Encapsulated PostScript, stored in a file named `fig.eps`.

```
[dvlabel> dvlabel -c -e -o fig
This is dvlabel, Version 1.4
Specify table of contents of the DV tape
[Enter text and finish with single dot on blank line]:
* Table of contents
* This is line number one
* This is line number two
* This is line number three
* .
Specify time stamp of DV tape.  This is typically
the last date that appears as time stamp in the
recorded tape.  [Press enter to use the current date]
* Monday 29/12/2003
Specify title of the DV tape [Press enter to leave blank title]:
* Beavis and Butthead
Specify author of the DV tape [Press enter to leave blank]:
* Fredrik Jonsson
Specify address of author of the DV tape
[Enter text and finish with single dot on blank line]:
* Storgatan 77
* SE-47111 Sunne, Sweden
* .
Specify email of author of the DV tape [Press enter to leave blank]:
* hmv@hotmail.com
```

The generated DV label `fig.eps` looks like this:



In this example, the `-c` command line option specifies that the program should compile the generated TEX code, `-e` that it also should attempt to generate the output as Encapsulated PostScript, and `-o fig` that the generated files all should have the base name "`fig`", that is to say, the program will use `fig.tex` for the generated TEX-code, `fig.dvi` for the generated device-independent (DVI) output, and `fig.eps` for the generated Encapsulated PostScript (EPS) image of the DV label. The command for invoking the DVLABEL program can equivalently be written in a more extensive form as `dvlabel --compile --eps --outputfile fig`.

The TEX-code that the program generates (from which the Encapsulated PostScript subsequently is generated) is stored in the file `fig.tex`, which contains

```
% File:  fig.tex
% TeX code automatically generated by dvlabel, v.1.4,
% Mon Dec 29 15:46:36 2003
% Copyright (C) Fredrik Jonsson, 2003
%
\font\eightcmssqeight=cmssq8
\font\tencmssqten=cmssq8 at 10 truept
\font\defaultfacefont=cmtt8
\font\defaultflapfont=cmtt8
\def\monthname\ifcase\month%
 \or Jan\or Feb\or Mar\or
Apr\or May\or Jun%
 \or Jul\or Aug\or Sep\or
Oct\or Nov\or Dec%
 \fi
\def\fullmonthname\ifcase\month%
 \or January\or February\or
March\or April%
 \or May\or June\or July\or
August\or September
 \or October\or November\or
```

```
December\fi
     \def\today\fullmonthname\space\number\day,%
       \space\number\year
     \def\timestampMonday 29/12/2003
     \def\titleBeavis and Butthead
     \def\authorFredrik Jonsson
     \def\emailhmv@hotmail.com
     \nopagenumbers
     \newdimen\facewidth
     \newdimen\faceheight
     \newdimen\spineheight
     \newdimen\flapheight
     \newdimen\edgeseparation
     \facewidth=60mm
     \faceheight=40mm
     \spineheight=12mm
     \flapheight=20mm
     \edgeseparation=1pt
     \def\boxit#1\vbox\hrule\hbox%
       \vrule\kern1pt\vbox\kern1pt#1\kern1pt%
     \kern1pt\vrule\hrule
     \parindent 0pt
```

```
      % Define the outline of the face of the label
      \setbox1=\hbox to \facewidth%
        \vbox to \faceheight%
          \hbox\defaultfacefont\timestamp\hfil%
          \vskip -3pt\hbox\defaultfacefont
Table of contents\hfil%
          \vskip -3pt\hbox\defaultfacefont
This is line number one\hfil%
          \vskip -3pt\hbox\defaultfacefont
This is line number two\hfil%
          \vskip -3pt\hbox\defaultfacefont
This is line number three\hfil%
        \vfil\hfil

      % Define the outline of the spine of the label
      \setbox2=\hbox to \facewidth%
        \vbox to \spineheight%
          \hbox\bf\timestamp\hfil%
          \hbox\bf\title\hfil%
        \vfil\hfil

      % Define the outline of the flap of the label
      \setbox3=\hbox to \facewidth%
        \vbox to \flapheight%
          \hbox\defaultflapfont Digital Video by
\author\hfil%
          \vskip -3pt\hbox\defaultflapfont
Storgatan 77\hfil%
          \vskip -3pt\hbox\defaultflapfont
SE-47111 Sunne, Sweden\hfil%
          \vskip -3pt\hbox\defaultflapfont %
            <\email>\hfil%
        \vfil\hfil

      \def\dvlabel\boxit\boxit\box1%
        \vskip\edgeseparation\boxit\box2%
        \vskip\edgeseparation\boxit\box3

      \dvlabel

      \bye
```

Example II: Another way of using the program is in batch mode, where the input instead is read from a text file. In the following example, the file named `fig.asc` contains the following text (compare with the interactive session on the previous pages):

```
Table of contents
This is line number one
This is line number two
This is line number three
.
Monday 29/12/2003
Beavis and Butthead
Fredrik Jonsson
Storgatan 77
SE-47111 Sunne, Sweden
.
hmv@hotmail.com
```

When the DVLABEL program is launched with the above file specified as the input to read, via the command†

```
[dvlabel> dvlabel -c -e -i fig.asc -o fig
```

the same output as in the previous example will be generated.

† In similar to the previous example, this command for invoking the DVLABEL program can similarly written in a more extensive form as `dvlabel --compile --eps --inputfile fig.asc --outputfile fig`.

5. The main program. Here follows the general outline of the main program.

```
#include <time.h>
#include <sys/time.h>
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define VERSION "1.6"  /* Program revision */
#define DEFAULT_DVIPSOPTS "-ta4␣-D1200"  /* Default DVIPS options */
#define DEFAULT_LINETHICKNESS (0.5)  /* Default line thickness in pt */
#define DEFAULT_CROPLINETHICKNESS (0.15)  /* Crop line thickness in pt */
#define DEFAULT_EDGESEPARATION (1.0)
#define MAXCHAR (128)  /* Maximum allowed number of characters on each line */
#define SUCCESS (0)  /* Return code for successful program termination */
#define FAILURE (1)  /* Return code for unsuccessful program termination */
#define INSTREAM (infile_specified ? infile : stdin)
#define OUTSTREAM (outfile_specified ? outfile : stdout)
#define log(...) log_printf (__func__, __VA_ARGS__)
  ⟨ Global variables 6 ⟩
  ⟨ Subroutines 7 ⟩
  int main(int argc, char *argv[])
  {
    ⟨ Local variables 14 ⟩
    ⟨ Parse command line 15 ⟩
    ⟨ Display banner 16 ⟩
    ⟨ Open files for reading 18 ⟩
    ⟨ Open files for writing 19 ⟩
    ⟨ Save preamble source code 29 ⟩
    num_labels = 0;
    while (newlabel(INSTREAM)) {
      num_labels++;
      ⟨ Scan text lines of one label from input stream 22 ⟩
      if (num_labels ≡ 1) {
        if (generate_crop_marks) {
          ⟨ Write vertical crop marks with width measure 32 ⟩
          ⟨ Write horizontal crop marks with height measure 33 ⟩
        }
      }
      ⟨ Save label source code 30 ⟩
      if (num_labels ≡ 2) {
        if (generate_crop_marks) {
          ⟨ Write horizontal crop marks with height measure 33 ⟩
        }
      }
      if (num_labels ≡ num_labels_per_page) {
        if (generate_crop_marks) {
          ⟨ Write horizontal crop marks with height measure 33 ⟩
          ⟨ Write vertical crop marks with width measure 32 ⟩
        }
        ⟨ Close output page 31 ⟩
        num_labels = 0;
```

```
    }
  }
  ⟨ Save closing source code 34 ⟩
  ⟨ Close files 20 ⟩
  ⟨ Compile source code 35 ⟩
  return (SUCCESS);
}
```

6. Declaration of global variables. The only global variables allowed in my programs are *optarg*, which is the string of characters that specified the call from the command line, and *progname*, which simply is the string containing the name of the program, as it was invoked from the command line.

```
⟨ Global variables 6 ⟩ ≡
  extern char *optarg;
  char *progname;
```

This code is used in section 5.

7. Declarations of subroutines used by the program.

```
⟨ Subroutines 7 ⟩ ≡
  ⟨ Routine for logging and error messaging 8 ⟩
  ⟨ Routine for displaying help message 9 ⟩
  ⟨ Scan for beginning of new label 11 ⟩
  ⟨ Routine for removing preceding path of filenames 12 ⟩
```

This code is used in section 5.

8. The **void** *log_printf*(**const char** ∗*function_name*, **const char** ∗*format*, ...) routine writes formatted entries to standard output, displaying time and calling routine in a coherent manner. Notice that although the *log_printf*( ) routine is the one which performs the actual messaging, the *log*( ) macro (defined in the header file) is the preferred way of accessing this routine, as it provides a more compact notation and automatically takes care of supplying the reference to the name of the calling function.

Also notice that the **const char** type of the last two input pointer arguments here is absolutely essential in order to pass strict pedantic compilation with GCC.

The routine accepts two input parameters. First, *function_name* which should be the name of the calling function. This is to ensure that any displayed error messages are properly matched to the issuing routines. Notice, however, that the *log*( ) macro (which is the preferred way of displaying error messages) automatically takes care of supplying the proper function name. Second, *format*, which simply is the format and message string to be displayed, formatted in the C-standard *printf*( ) or *fprintf*( ) syntax.

⟨ Routine for logging and error messaging 8 ⟩ ≡

```
void log_printf(const char *function_name, const char *format, ...)
{
  va_list args;
  time_t time0;
  struct tm lt;
  struct timeval tv;
  char logentry[1024];
  gettimeofday(&tv, Λ);
  time(&time0);
  lt = *localtime(&time0);
  sprintf(logentry, "%02u%02u%02u␣%02u:%02u:%02u.%03d␣", lt.tm_year − 100, lt.tm_mon + 1,
      lt.tm_mday, lt.tm_hour, lt.tm_min, lt.tm_sec, tv.tv_usec/1000);
  sprintf(logentry + strlen(logentry), "(%s)␣", function_name);
  va_start(args, format);      /* Initialize args by the va_start( ) macro */
  vsprintf(logentry + strlen(logentry), format, args);
  va_end(args);      /* Terminate the use of args by the va_end( ) macro */
  sprintf(logentry + strlen(logentry), "\n");      /* Always append newline */
  fprintf(stdout, "%s", logentry);
  return;
}
```

This code is used in section 7.

9. Routines for displaying help message.

⟨ Routine for displaying help message 9 ⟩ ≡

```
⟨ Routine for displaying a single line of the help message 10 ⟩
void showsomehelp(void)
{
  hl("Usage: %s [options]", progname);
  hl("When invoked without any command line options, this program enters");
  hl("interactive mode.");
  hl("Options:");
  hl(" -i, --inputfile <str>");
  hl("     Specifies the file where to read the label text strings, entered in");
  hl("     a format corresponding to the input order as would have been");
  hl("     entered in interactive mode.");
  hl(" -o, --outputfile <str>");
  hl("     Specifies the file where to save the generated TeX source code for");
  hl("     the label. Whenever this option is omitted, the generated source");
  hl("     code will be written to standard terminal output instead. Notice");
  hl("     that the name specified using this option is used as the *base");
  hl("     name* for the generated output; this means that the TeX file");
  hl("     automatically will get .tex as suffix, the DVIfile .dvi, etc.");
  hl("     For example, if you want to have your TeX output stored into a file");
  hl("     named, say, 'foo.tex', just specify '-o foo'.");
  hl(" -H, --headline");
  hl("     Toggle display of generation time and input source in page header");
  hl("     of the generated TeX code. Default: off");
  hl(" -C, --cropmarks");
  hl("     Toggle display of crop marks (alignment marks) in the generated TeX");
  hl("     code. Default: off");
  hl(" -t, --linethick <f>");
  hl("     Use linethickness of <f> typographic points in generating the boxes");
  hl("     of the label. (One point, or pt, equals to 1/72.27 inch.)");
  hl(" -s, --edgeseparation <f>");
  hl("     Insert extra space of <f> typographic points between the text boxes");
  hl("     at the folded edges of the label. This is useful when, for example,");
  hl("     thick paper is used. By changing the line thickness, the internal");
  hl("     margins of the label are also changed, since the inner edge-to-edge");
  hl("     distance between the lines of the inner and outer boxes are linked");
  hl("     to be twice the value of the line thickness. The correction of");
  hl("     inner bounding boxes of the label text is automatically adjusted so");
  hl("     as to ensure that the overall outer dimensions of the label are");
  hl("     unchanged.");
  hl(" -c, --compile");
  hl("     Try to compile the generated TeX code. This requires DVIPS to be");
  hl("     installed on your system, see www.radicaleye.com for further info.");
  hl(" -e, --eps");
  hl("     When compiling the generated TeX code, generate Encapsulated");
  hl("     PostScript (EPS) instead of the default regular PostScript.");
  hl(" -d, --dvipsopts <str>");
  hl("     When compiling the generated TeX code, use <str> as options to be");
  hl("     supplied to DVIPS. In order to parse for an arbitrary number of");
  hl("     DVIPS options at the command line, it is important to enclose the");
  hl("     list of DVIPS options by quotes. Hence, for example, to force DVIPS");
```

```
  hl("␣␣␣␣to␣generate␣output␣pages␣of␣US␣letter␣format␣and␣at␣a␣resolution␣of");
  hl("␣␣␣␣720␣dpi␣one␣could␣invoke␣DVLABEL␣with");
  hl("␣␣␣␣␣␣␣␣␣␣dvlabel␣--dvipsopts␣\"-tletter␣-D720\"");
  hl("␣␣␣␣The␣quotes␣are␣only␣necessary␣if␣the␣number␣of␣DVIPS␣options␣are");
  hl("␣␣␣␣two␣or␣more.␣Default:␣-D1200␣-ta4␣(1200␣DPI␣on␣A4␣paper).");
  hl("␣-v,␣--verbose");
  hl("␣␣␣␣Toggle␣verbose␣mode.␣Default:␣off");
  hl("␣-h,␣--help");
  hl("␣␣␣␣Display␣this␣help␣message␣and␣exit␣clean.");
  hl("\nCopyright␣(C)␣2003-2011␣Fredrik␣Jonsson␣<http://jonsson.eu>");
}
```

This code is used in section 7.

10. In order to simplify the messaging, the *hl*(**const char** ∗*format*, ...) routine acts as a simple front-end merely for compactifying the code by successive calls to *hl*(...) rather than the full *fprintf*(*stderr*, ...), still maintaining all the functionality of string formatting in the regular *printf*( ) or *fprintf*( ) syntax.

⟨ Routine for displaying a single line of the help message 10 ⟩ ≡

```
void hl(const char *format, ...){ va_list args; char
    line [1024] ;
    va_start(args, format);      /* Initialize args by the va_start( ) macro */
    vsprintf ( line , format, args ) ;
    va_end(args);      /* Terminate the use of args by the va_end( ) macro */
    sprintf ( line +strlen ( line ) , "\n" ) ;      /* Always append newline */
    fprintf (stdout, "%s", line ) ;
    return; }
```

This code is used in section 9.

11. Scan the input stream for a statement on what to do next, that is to say, either to start generating a new label (if the character 'n' is present, in which case the *newlabel* routine returns 1) or quit the program (if the character 'q' is present, in which case the *newlabel* routine returns 0). The *newlabel* routine also takes care of scanning away any trailing blanks or other characters from the input stream until a carriage return character is found, also removing this before return.

```
⟨ Scan for beginning of new label 11 ⟩ ≡
  short newlabel(FILE *instream)
  {
    char ch;
    fprintf(stdout, "Create␣new␣label␣record␣or␣quit?\n");
    fprintf(stdout, "['n'=new␣label␣/␣'q'=quit]:\n");
    fprintf(stdout, "**");
    while ((ch = fgetc(instream)) ≡ '␣') ;      /* get rid of leading blanks */
    if (ch ≡ 'n') {
      while ((ch = fgetc(instream)) ≠ '\n') ;      /* get rid of trailing characters */
      return (1);
    }
    else if (ch ≡ 'q') {
      while ((ch = fgetc(instream)) ≠ '\n') ;      /* get rid of trailing characters */
      return (0);
    }
    else {
      log("%s:␣Error:␣Unrecognized␣input␣stream␣control␣character␣'%c'.\n", progname, ch);
      exit(FAILURE);
    }
  }
```

This code is used in section 7.

12. Routines for removing preceding path of filenames. In this block all routines related to removing preceding path strings go. Not really fancy programming, and no contribution to any increase of numerical efficiency or precision; just for the sake of keeping a tidy terminal output of the program. The *strip_away_path*( ) routine is typically called when initializing the program name string *progname* from the command line string *argv*[0], and is typically located in the blocks related to parsing of the command line options. The *strip_away_path*( ) routine takes a character string *filename* as argument, and returns a pointer to the same string but without any preceding path segments.

```
⟨ Routine for removing preceding path of filenames 12 ⟩ ≡
  ⟨ Routine for checking for a valid path character 13 ⟩
  char *strip_away_path(char filename[])
  {
    int j, k = 0;
    while (pathcharacter(filename[k])) k++;
    j = (−−k);      /* this is the uppermost index of the full path+file string */
    while (isalnum((int)(filename[j]))) j−−;
    j++;      /* this is the lowermost index of the stripped file name */
    return (&filename[j]);
  }
```

This code is used in section 7.

13. In this program, valid path characters are any alphanumeric character or '`.`', '`/`', '`\`', '`_`', '`-`', or '`+`'.

```
⟨ Routine for checking for a valid path character 13 ⟩ ≡
  short pathcharacter(int ch)
  {
    return (isalnum(ch) ∨ (ch ≡ '.') ∨ (ch ≡ '/') ∨ (ch ≡ '\\') ∨ (ch ≡ '_') ∨ (ch ≡ '-') ∨ (ch ≡ '+'));
  }
```

This code is used in section 12.

14. Declaration of local variables of the main program. Here *num_labels* is a counter that is used for determining whenever a page brake in the generated TEX code is to appear, in order to give a nice output with a $[3 \times 1]$ array of labels.

```
⟨ Local variables 14 ⟩ ≡
  time_t now = time(Λ);
  long j, k, n, num_address_lines, num_toc_lines, num_labels, num_labels_per_page = 4;
  int no_arg, tmpch;
  FILE *infile = Λ, *outfile = Λ;
  char inputfilename[MAXCHAR] = "", outputfilename[MAXCHAR] = "", tmpstr[MAXCHAR];
  char dvipsopts[MAXCHAR] = DEFAULT_DVIPSOPTS;
  char timestamp[MAXCHAR] = "\today", title[MAXCHAR] = "";
  char author[MAXCHAR] = "", email[MAXCHAR] = "";
  char address[10 * MAXCHAR] = "", toc[50 * MAXCHAR] = "";
  short done, verbose = 0, compile = 0, eps_output = 0;
  short headline = 1, generate_crop_marks = 1;
  short infile_specified = 0, outfile_specified = 0, dvipsopts_specified = 0;
  float linethickness = DEFAULT_LINETHICKNESS;
  float clthick = DEFAULT_CROPLINETHICKNESS;
  float edgeseparation = DEFAULT_EDGESEPARATION;
  float facewidth, faceheight, spineheight, flapheight;
  float ifacewidth, ifaceheight, ispineheight, iflapheight;
```

This code is used in section 5.

15. Parsing command line options. All input parameters are passed to the program through command line options and arguments to the program. The syntax of command line options is listed whenever the program is invoked with the `--help` option at startup, or whenever an error occurs in the input.

```
⟨ Parse command line 15 ⟩ ≡
  {
    progname = strip_away_path(argv[0]);
    no_arg = argc;
    while (--argc) {
      if (¬strcmp(argv[no_arg − argc], "-o") ∨ ¬strcmp(argv[no_arg − argc], "--outputfile")) {
        --argc;
        strcpy(outputfilename, argv[no_arg − argc]);
        outfile_specified = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "-i") ∨ ¬strcmp(argv[no_arg − argc], "--inputfile")) {
        --argc;
        strcpy(inputfilename, argv[no_arg − argc]);
        infile_specified = 1;
      }
      else if (¬strcmp(argv[no_arg − argc], "-v") ∨ ¬strcmp(argv[no_arg − argc], "--verbose")) {
        verbose = (verbose ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-h") ∨ ¬strcmp(argv[no_arg − argc], "--help")) {
        showsomehelp( );
        exit(SUCCESS);
      }
      else if (¬strcmp(argv[no_arg − argc], "-H") ∨ ¬strcmp(argv[no_arg − argc], "--headline")) {
        headline = (headline ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-C") ∨ ¬strcmp(argv[no_arg − argc], "--cropmarks")) {
        generate_crop_marks = (generate_crop_marks ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-t") ∨ ¬strcmp(argv[no_arg − argc], "--linethick")) {
        --argc;
        if (¬sscanf(argv[no_arg − argc], "%f", &linethickness)) {
          log("Error␣in␣linethickness␣argument.");
          exit(1);
        }
      }
      else if (¬strcmp(argv[no_arg − argc], "-s") ∨ ¬strcmp(argv[no_arg − argc], "--edgeseparation")) {
        --argc;
        if (¬sscanf(argv[no_arg − argc], "%f", &edgeseparation)) {
          log("Error␣in␣edgeseparation␣argument.");
          exit(1);
        }
      }
      else if (¬strcmp(argv[no_arg − argc], "-c") ∨ ¬strcmp(argv[no_arg − argc], "--compile")) {
        compile = (compile ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-e") ∨ ¬strcmp(argv[no_arg − argc], "--eps")) {
        eps_output = (eps_output ? 0 : 1);
      }
      else if (¬strcmp(argv[no_arg − argc], "-d") ∨ ¬strcmp(argv[no_arg − argc], "--dvipsopts")) {
```

```
        −− argc;
        strcpy(dvipsopts, argv[no_arg − argc]);
        dvipsopts_specified = 1;
      }
      else {
        log("Error: Unknown option '%s'.", argv[no_arg − argc]);
        showsomehelp( );
        exit(FAILURE);
      }
    }
    if (verbose ∧ ¬outfile_specified) {
      fprintf(stdout, "You have specified verbose mode without any specification o\
          f\n""a file where to save the generated TeX source code. This me\
          ans\n""that any program comments that appear, due to the verbose m\
          ode,\n""will be mixed with the source code. It is highly recommende\
          d that\n""you either turn off verbose mode or specify a file where to\
          \ save\n""the generated output (using the -o or --outputfile option).\n");
      fprintf(stdout, "In order to make a clean output, the TeX code is separately\
          \n""enclosed in the blocks below.\n");
    }
    if (verbose ∧ dvipsopts_specified) {
      fprintf(stdout, "Specified options for later use with DVIPS: ");
      fprintf(stdout, "%s\n", dvipsopts);
    }
  }
```

This code is used in section 5.

16. Display a banner at start-up of the program.

⟨ Display banner 16 ⟩ ≡

```
  {
    fprintf(stdout, "This is %s, Version %s\n", progname, VERSION);
  }
```

This code is used in section 5.

17. Opening and closing files for data output.

18. Open files for reading. In order to read the text to typeset from an external file rather than the default standard input stream `stdin`, the name of the input file is specified with the `-i` or `--inputfile` command line options. If the filename as specified on the command line does not exist, then the program will instead try to open a file with the suffix `.dvl` concatenated to the name. This way, the program will accept short-hand filenames as well, in a way analogous to the input syntax rules of, for example, the TEX program. If no input file can be opened, an error message is displayed and the program will exit with return value `FAILURE`.

```
⟨ Open files for reading 18 ⟩ ≡
  {
    if (infile_specified) {
      if ((infile = fopen(inputfilename, "r")) ≡ Λ) {
        if ((infile = fopen(strcat(inputfilename, ".dvl"), "r")) ≡ Λ) {
          log("Could␣not␣open␣file␣%s␣for␣reading!", inputfilename);
          exit(FAILURE);
        }
      }
      fseek(infile, 0L, SEEK_SET);
      if (verbose) {
        log("Opened␣input␣file␣%s␣for␣reading.", inputfilename);
      }
    }
    else {
      if (verbose) log("No␣input␣filename␣specified␣(Entering␣interactive␣mode).");
    }
  }
```

This code is used in section 5.

19. Open files for writing.

```
⟨ Open files for writing 19 ⟩ ≡
  {
    if (outfile_specified) {
      sprintf(tmpstr, "%s.tex", outputfilename);
      if ((outfile = fopen(tmpstr, "w")) ≡ Λ) {
        log("Could␣not␣open␣file␣%s␣for␣writing!", tmpstr);
        exit(FAILURE);
      }
      fseek(outfile, 0L, SEEK_SET);
      if (verbose) {
        log("Opened␣output␣file␣%s␣for␣writing.", outputfilename);
      }
    }
    else {
      if (verbose) {
        log("No␣output␣filename␣specified.");
        log("Will␣write␣generated␣TeX␣output␣to␣standard␣terminal␣output.");
      }
    }
  }
```

This code is used in section 5.

20. Close all open files.

```
⟨ Close files 20 ⟩ ≡
  {
    if (infile_specified) fclose(infile);
    if (outfile_specified) fclose(outfile);
  }
```

This code is used in section 5.

21. Reading input as supplied by the user.

22. Parsing the input as supplied by the user. When reading the input, the DVLABEL program makes use of either the *stdin* stream (in the case of interactive mode), or the file pointed out by the *infile* pointer (in the case of processing of an input text file). In the latter case, the syntax of the supplied file is identical to the syntax as if operating the program in interactive mode, with every newline character (`/n`) interpreted as an end-of-line of a corresponding interactive input.

Example: [TO BE INSERTED]

As the individual lines of text generally contain different number of characters, the feedline character `/n` of standard C is throughout the program used to keep track of the end of line. (This is more convenient than keeping track of the indvidual line widths in, for example, and array of integers.)

⟨ Scan text lines of one label from input stream 22 ⟩ ≡

```
{
  ⟨ Read table of contents 23 ⟩
  ⟨ Read time stamp 24 ⟩
  ⟨ Read title 25 ⟩
  ⟨ Read author name 26 ⟩
  ⟨ Read address 27 ⟩
  ⟨ Read email address 28 ⟩
}
```

This code is used in section 5.

23. Parse for the table of contents of the tape. The syntax for specifying the table of contents is similar to that of the address, with arbitrary number of lines of text. The input is ended by a single dot on a new line, in similar to the way of entering, for example, email messages in terminal mode using the basic engine of `sendmail` on regular UNIX and Linux systems.

```
⟨ Read table of contents 23 ⟩ ≡
  {
    fprintf(stdout, "Specify␣table␣of␣contents␣of␣the␣DV␣tape\n");
    fprintf(stdout, "[Enter␣text␣and␣finish␣with␣single␣dot␣on␣blank␣line]:\n");
    done = 0;
    n = 0;
    num_toc_lines = 0;
    while (¬done) {
      fprintf(stdout, "**");
      while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading blanks */
      ungetc(tmpch, INSTREAM);
      k = 0;
      while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
      if (k > 1) {      /* If more text than just spaces and a linefeed character */
        if ((tmpstr[0] ≡ '.') ∧ (tmpstr[1] ≡ '\n')) {
          done = 1;
        }
        else {
          for (j = 0; j < k; j++) toc[n + j] = tmpstr[j];
          if (1 ≡ 0) toc[n + j] = '\n';
          n = n + j;      /* Keep track of last index of the toc array */
          num_toc_lines++;
        }
      }
      else {      /* If just spaces and a linefeed character */
        toc[n] = '\n';      /* Blank line of table-of-contents field */
      }
    }
    if (verbose) {
      fprintf(stdout, "%s:␣Counted␣%ld␣number␣of␣table-of-contents␣lines.\n", progname,
          num_toc_lines);
      fprintf(stdout, "Table␣of␣contents:\n");
      n = 0;
      for (k = 1; k ≤ num_toc_lines; k++) {
        for (j = n; toc[j] ≠ '\n'; j++) fprintf(stdout, "%c", toc[j]);
        fprintf(stdout, "\n");
        n = j + 1;
      }
    }
  }
```

This code is used in section 22.

24. Parse for the time stamp to be used.

```
⟨ Read time stamp 24 ⟩ ≡
  {
    fprintf(stdout, "Specify␣time␣stamp␣of␣DV␣tape.␣This␣is␣typically\n");
    fprintf(stdout, "the␣last␣date␣that␣appears␣as␣time␣stamp␣in␣the\n");
    fprintf(stdout, "recorded␣tape.␣[Press␣enter␣to␣use␣the␣current␣date]\n");
    fprintf(stdout, "**");
    while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading blank spaces */
    ungetc(tmpch, INSTREAM);
    k = 0;
    while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
    if (k > 1)      /* If more text than just spaces and a linefeed character */
      for (j = 0; j < k; j++) timestamp[j] = tmpstr[j];
    else      /* If just spaces and a linefeed character */
      for (j = 0; j < 7; j++) timestamp[j] = "\\today\n"[j];
    if (verbose) {
      fprintf(stdout, "Time␣stamp:␣");
      for (j = 0; timestamp[j] ≠ '\n'; j++) fprintf(stdout, "%c", timestamp[j]);
      fprintf(stdout, "\n");
    }
  }
```

This code is used in section 22.

25. Parse for the title to be used.

```
⟨ Read title 25 ⟩ ≡
  {
    fprintf(stdout, "Specify␣title␣of␣the␣DV␣tape␣[Press␣enter␣to␣leave␣blank␣title]:\n");
    fprintf(stdout, "**");
    while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading blank spaces */
    ungetc(tmpch, INSTREAM);
    k = 0;
    while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
    if (k > 1)      /* If more text than just spaces and a linefeed character */
      for (j = 0; j < k; j++) title[j] = tmpstr[j];
    else      /* If just spaces and a linefeed character */
      title[0] = '\n';      /* Blank title */
    if (verbose) {
      fprintf(stdout, "Title:␣");
      for (j = 0; title[j] ≠ '\n'; j++) fprintf(stdout, "%c", title[j]);
      fprintf(stdout, "\n");
    }
  }
```

This code is used in section 22.

26. Parse for the author name to be used.

```
⟨ Read author name 26 ⟩ ≡
  {
    fprintf(stdout, "Specify␣author␣of␣the␣DV␣tape␣[Press␣enter␣to␣leave␣blank]:\n");
    fprintf(stdout, "**");
    while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading blank spaces */
    ungetc(tmpch, INSTREAM);
    k = 0;
    while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
    if (k > 1)      /* If more text than just spaces and a linefeed character */
      for (j = 0; j < k; j++) author[j] = tmpstr[j];
    else      /* If just spaces and a linefeed character */
      author[0] = '\n';      /* Blank author field */
    if (verbose) {
      fprintf(stdout, "Author:␣");
      for (j = 0; author[j] ≠ '\n'; j++) fprintf(stdout, "%c", author[j]);
      fprintf(stdout, "\n");
    }
  }
```

This code is used in section 22.

27. Parse for the address to be used. The syntax for specifying the address is similar to that of the table of contents, with arbitrary number of lines of text. The input is ended by a single dot on a new line, in similar to the way of entering, for example, email messages in terminal mode using the basic engine of `sendmail` on regular UNIX and Linux systems.

```
⟨ Read address 27 ⟩ ≡
  {
    fprintf(stdout, "Specify␣address␣of␣author␣of␣the␣DV␣tape\n");
    fprintf(stdout, "[Enter␣text␣and␣finish␣with␣single␣dot␣on␣blank␣line]:\n");
    fprintf(stdout, "**");
    done = 0;
    n = 0;
    num_address_lines = 0;
    while (¬done) {
      while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading spaces */
      ungetc(tmpch, INSTREAM);
      k = 0;
      while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
      if (k > 1) {      /* If more text than just spaces and a linefeed character */
        if ((tmpstr[0] ≡ '.') ∧ (tmpstr[1] ≡ '\n')) {
          done = 1;
        }
        else {
          for (j = 0; j < k; j++) address[n + j] = tmpstr[j];
          if (1 ≡ 0) address[n + j] = '\n';
          n = n + j;      /* Keep track of last index of the address array */
          num_address_lines++;
        }
      }
      else {      /* If just spaces and a linefeed character */
        address[n] = '\n';      /* Blank line of address field */
      }
      fprintf(stdout, "**");
    }
    if (verbose) {
      fprintf(stdout, "%s:␣Counted␣%ld␣address␣lines.\n", progname, num_address_lines);
      fprintf(stdout, "Address:\n");
      n = 0;
      for (k = 1; k ≤ num_address_lines; k++) {
        for (j = n; address[j] ≠ '\n'; j++) fprintf(stdout, "%c", address[j]);
        fprintf(stdout, "\n");
        n = j + 1;
      }
    }
  }
```

This code is used in section 22.

28. Parse for the email address to be used.

```
⟨ Read email address 28 ⟩ ≡
  {
    fprintf(stdout, "Specify␣email␣of␣author␣of␣the␣DV␣tape␣[Press␣enter␣to␣leave␣blank]:\n");
    fprintf(stdout, "**");
    while ((tmpch = fgetc(INSTREAM)) ≡ '␣') ;      /* Get rid of leading blank spaces */
    ungetc(tmpch, INSTREAM);
    k = 0;
    while ((tmpstr[k++] = fgetc(INSTREAM)) ≠ '\n') ;      /* Read line */
    if (k > 1)      /* If more text than just spaces and a linefeed character */
      for (j = 0; j < k; j++) email[j] = tmpstr[j];
    else      /* If just spaces and a linefeed character */
      email[0] = '\n';      /* Blank email field */
    if (verbose) {
      fprintf(stdout, "Email:␣");
      for (j = 0; email[j] ≠ '\n'; j++) fprintf(stdout, "%c", email[j]);
      fprintf(stdout, "\n");
    }
  }
```

This code is used in section 22.

29. Generating the preamble of output TEX source code for the labels. The physical outer of the Mini Digital Label dimensions are based on the inset of TDK cassettes, with *flapheight* of 14.5 mm (41.3 pt), spineheight of 12.5 mm (35.6 pt), *faceheight* of 47 mm (133.7 pt), and *facewidth* of 67 mm (190.6 pt), hence summing up to overall outer dimensions of 190.6 × 210.6 pt of the label.

```
⟨ Save preamble source code 29 ⟩ ≡
  {
    facewidth = 190.6 + 3.0 * 72.27/25.4;
    faceheight = 133.7;
    spineheight = 35.6;
    flapheight = 41.3;
    ifacewidth = facewidth − 14.0 * linethickness;
    ifaceheight = faceheight − 11.0 * linethickness − 0.5 * edgeseparation;
    ispineheight = spineheight − 8.0 * linethickness − edgeseparation;
    iflapheight = flapheight − 11.0 * linethickness − 0.5 * edgeseparation;
    if (¬outfile_specified)
      fprintf(OUTSTREAM, "%%--------␣TEX␣CODE␣BEGINS␣HERE␣-----------------\n");
    fprintf(OUTSTREAM, "%%␣File:␣%s.tex\n", outputfilename);
    fprintf(OUTSTREAM, "%%␣TeX␣code␣generated␣by␣%s,␣v.%s,␣%s", progname, VERSION, ctime(&now));
    fprintf(OUTSTREAM, "%%␣Copyright␣(C)␣Fredrik␣Jonsson,␣2003-2005\n%%\n");
    fprintf(OUTSTREAM, "\\hoffset=-35pt\\voffset=-25pt\n");
    fprintf(OUTSTREAM, "\\hsize=175mm\\vsize=254mm\n");
    fprintf(OUTSTREAM, "\\font\\eightcmssqeight=cmssq8\n");
    fprintf(OUTSTREAM, "\\font\\sevencmtt=cmtt7\n");
    fprintf(OUTSTREAM, "\\font\\tencmssqten=cmssq8␣at␣10␣truept\n");
    fprintf(OUTSTREAM, "\\font\\deffacefont=cmr7\n");
    fprintf(OUTSTREAM, "\\font\\deftimestampfont=cmtt8␣at␣7␣truept\n");
    fprintf(OUTSTREAM, "\\font\\defspinefont=cmr9\n");
    fprintf(OUTSTREAM, "\\font\\defflapfont=cmr7\n");
    fprintf(OUTSTREAM, "\\def\\monthname{\\ifcase\\month%%\n");
    fprintf(OUTSTREAM, "␣\\or␣Jan\\or␣Feb\\or␣Mar\\or␣Apr\\or␣May\\or␣Jun%%\n");
    fprintf(OUTSTREAM, "␣\\or␣Jul\\or␣Aug\\or␣Sep\\or␣Oct\\or␣Nov\\or␣Dec%%\n");
    fprintf(OUTSTREAM, "␣\\fi}\n");
    fprintf(OUTSTREAM, "\\def\\fullmonthname{\\ifcase\\month%%\n");
    fprintf(OUTSTREAM, "␣\\or␣January\\or␣February\\or␣March\\or␣April%%\n");
    fprintf(OUTSTREAM, "␣\\or␣May\\or␣June\\or␣July\\or␣August\\or␣September\n");
    fprintf(OUTSTREAM, "␣\\or␣October\\or␣November\\or␣December\\fi}\n");
    fprintf(OUTSTREAM, "\\def\\today{\\fullmonthname\\space\\number\\day,%%\n");
    fprintf(OUTSTREAM, "␣␣\\space\\number\\year}\n");
    fprintf(OUTSTREAM, "\\def\\dvby{Digital␣Video␣by\\␣}\n");
    fprintf(OUTSTREAM, "\\parindent␣0pt\n");
    fprintf(OUTSTREAM, "\\newdimen\\facewidth\n");
    fprintf(OUTSTREAM, "\\newdimen\\faceheight\n");
    fprintf(OUTSTREAM, "\\newdimen\\spineheight\n");
    fprintf(OUTSTREAM, "\\newdimen\\flapheight\n");
    fprintf(OUTSTREAM, "\\newdimen\\linethick\n");
    fprintf(OUTSTREAM, "\\newdimen\\spacethick\n");
    fprintf(OUTSTREAM, "\\newdimen\\edgeseparation\n");
    fprintf(OUTSTREAM, "\\newdimen\\cropthick\n");
    fprintf(OUTSTREAM, "\\facewidth=%1.3fpt\n", facewidth);
    fprintf(OUTSTREAM, "\\faceheight=%1.3fpt\n", faceheight);
    fprintf(OUTSTREAM, "\\spineheight=%1.3fpt\n", spineheight);
    fprintf(OUTSTREAM, "\\flapheight=%1.3fpt\n", flapheight);
```

```
    fprintf(OUTSTREAM, "\\linethick=%1.3fpt\n", linethickness);
    fprintf(OUTSTREAM, "\\spacethick=%1.3fpt\n", 2.0 * linethickness);
    fprintf(OUTSTREAM, "\\edgeseparation=%1.3fpt\n", edgeseparation);
    fprintf(OUTSTREAM, "\\cropthick=%1.3fpt\n", clthick);
    if (headline) {
      fprintf(OUTSTREAM, "\\headline={\\hfill{\\tt␣dvlabel␣output");
      fprintf(OUTSTREAM, "␣%s", ctime(&now));
      fprintf(OUTSTREAM, "␣[%s]}}\n", (infile_specified ? inputfilename : "stdin"));
      fprintf(OUTSTREAM, "\\footline={\\hfill\\folio\\hfill}\n");
    }
    else {
      fprintf(OUTSTREAM, "\\headline={\\hfil}\n");
      fprintf(OUTSTREAM, "\\nopagenumbers\n");
    }
    fprintf(OUTSTREAM, "\\def\\boxit#1{\\vbox{\\hrule␣height\\linethick%%\n");
    fprintf(OUTSTREAM, "␣␣\\hbox{\\vrule␣width\\linethick\\kern\\spacethick%%\n");
    fprintf(OUTSTREAM, "␣␣\\vbox{\\kern\\spacethick#1\\kern\\spacethick}%%\n");
    fprintf(OUTSTREAM, "␣␣\\kern\\spacethick\\vrule␣width\\linethick}%%\n");
    fprintf(OUTSTREAM, "␣␣\\hrule␣height\\linethick}}%%\n");
  }
```

This code is used in section 5.

30. Generating the output TEX source code for the labels.

⟨ Save label source code 30 ⟩ ≡

```
{
  fprintf(OUTSTREAM, "\\def\\timestamp{");
  for (j = 0; timestamp[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", timestamp[j]);
  fprintf(OUTSTREAM, "}%%\n");
  fprintf(OUTSTREAM, "\\def\\title{");
  for (j = 0; title[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", title[j]);
  fprintf(OUTSTREAM, "}%%\n");
  fprintf(OUTSTREAM, "\\def\\author{");
  for (j = 0; author[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", author[j]);
  fprintf(OUTSTREAM, "}%%\n");
  fprintf(OUTSTREAM, "\\def\\email{");
  for (j = 0; email[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", email[j]);
  fprintf(OUTSTREAM, "}%%\n");
  fprintf(OUTSTREAM, "%% Define the outline of the face of the label\n");
  fprintf(OUTSTREAM, "\\setbox1=\\hbox to %1.3fpt{%%\n", ifacewidth);
  fprintf(OUTSTREAM, "  \\vbox to \\faceheight{%%\n");
  fprintf(OUTSTREAM, "    \\hbox{\\deffacefont\\timestamp\\hfil}%%\n");
  n = 0;
  for (k = 1; k ≤ num_toc_lines; k++) {
    fprintf(OUTSTREAM, "    \\vskip -3pt\\hbox{\\deffacefont ");
    for (j = n; toc[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", toc[j]);
    fprintf(OUTSTREAM, "\\hfil}%%\n");
    n = j + 1;
  }
  fprintf(OUTSTREAM, "  \\vfil}}\\hfil}%%\n");
  fprintf(OUTSTREAM, "%% Define the outline of the spine of the label\n");
  fprintf(OUTSTREAM, "\\setbox2=\\hbox to %1.3fpt{%%\n", ifacewidth);
  fprintf(OUTSTREAM, "  \\vbox to %1.3fpt{%%\n", ispineheight);
  fprintf(OUTSTREAM, "    \\hbox{\\deftimestampfont\\timestamp\\hfil}%%\n");
  fprintf(OUTSTREAM, "    \\hbox{\\defspinefont\\title\\hfil}%%\n");
  fprintf(OUTSTREAM, "  \\vfil}}\\hfil}%%\n");
  fprintf(OUTSTREAM, "%% Define the outline of the flap of the label\n");
  fprintf(OUTSTREAM, "\\setbox3=\\hbox to %1.3fpt{%%\n", ifacewidth);
  fprintf(OUTSTREAM, "  \\vbox to %1.3fpt{%%\n", iflapheight);
  fprintf(OUTSTREAM, "    \\hbox{\\defflapfont\\dvby\\author\\hfil}%%\n");
  n = 0;
  for (k = 1; k ≤ num_address_lines; k++) {
    fprintf(OUTSTREAM, "    \\vskip -3pt\\hbox{\\defflapfont ");
    for (j = n; address[j] ≠ '\n'; j++) fprintf(OUTSTREAM, "%c", address[j]);
    fprintf(OUTSTREAM, "\\hfil}\n");
    n = j + 1;
  }
  fprintf(OUTSTREAM, "    \\vskip -3pt\\hbox{\\defflapfont\\email\\hfil}%%\n");
  fprintf(OUTSTREAM, "  \\vfil}}\\hfil}%%\n");
  fprintf(OUTSTREAM, "\\def\\dvlabel{\\hskip\\linethick\\boxit{%%\n");
  fprintf(OUTSTREAM, "  \\boxit{\\box1}\\vskip\\edgeseparation%%\n");
  fprintf(OUTSTREAM, "  \\boxit{\\box2}\\vskip\\edgeseparation%%\n");
  fprintf(OUTSTREAM, "  \\boxit{\\box3}}\\hskip\\linethick}%%\n");
  if ((num_labels ≡ 1) ∨ (num_labels ≡ 3)) fprintf(OUTSTREAM, "\\hskip %1.3fpt", (20.0));
  fprintf(OUTSTREAM, "\\dvlabel\n");
```

```
    if ((num_labels ≡ 2) ∨ (num_labels ≡ 4) ∨ (num_labels ≡ 6)) {
      fprintf(OUTSTREAM, "\\par\\nointerlineskip\n");
    }
  }
```

This code is used in section 5.

31. Close one page of output, with each page containing *num_labels_per_page* labels, and write any preamble for initialization of a new page to the output stream.

⟨ Close output page 31 ⟩ ≡

```
  {
    fprintf(OUTSTREAM, "\\vfill\\eject\n");
  }
```

This code is used in sections 5 and 34.

32. Before writing the first row of labels to the page output, save the vertical crop marks to be used as alignment marks in vertical cutting of the final pages.

⟨ Write vertical crop marks with width measure 32 ⟩ ≡

```
  {
    fprintf(OUTSTREAM, "\\hskip␣%1.3fpt\\vrule␣height20pt␣width%1.3fpt%%\n", 20.0 − clthick/2.0,
        clthick);
    fprintf(OUTSTREAM, "\\hbox␣to%1.3fpt{\\hfil", facewidth − clthick);
    fprintf(OUTSTREAM,
        "$\\matrix{{%1.3f{\\rm\\␣pt}/%1.3f{\\rm\\␣mm}}""\\cr{}\\cr}$\\hfil}%%\n", facewidth,
        facewidth ∗ (25.4/72.27));
    fprintf(OUTSTREAM, "\\vrule␣height20pt␣width%1.3fpt%%\n", clthick);
    fprintf(OUTSTREAM, "\\hbox␣to␣%1.3fpt{\\hfil", facewidth − clthick);
    fprintf(OUTSTREAM,
        "$\\matrix{{%1.3f{\\rm\\␣pt}/%1.3f{\\rm\\␣mm}}""\\cr{}\\cr}$\\hfil}%%\n", facewidth,
        facewidth ∗ (25.4/72.27));
    fprintf(OUTSTREAM, "\\vrule␣height20pt␣width%1.3fpt\\par\\nointerlineskip\n", clthick);
  }
```

This code is used in sections 5 and 34.

33. The following block is analogous to the previous one, but now instead considering horizontal crop marks, to be used as alignment marks in horizontal cutting of the final pages.

⟨ Write horizontal crop marks with height measure 33 ⟩ ≡

```
  {
    fprintf(OUTSTREAM, "\\vskip\\linethick\n");
    fprintf(OUTSTREAM, "\\vrule␣height%1.3fpt␣width20pt", clthick);
    fprintf(OUTSTREAM, "\\hskip␣%1.3fpt", 2.0 ∗ facewidth);
    fprintf(OUTSTREAM, "\\vrule␣height%1.3fpt␣width20pt\\par\\nointerlineskip\n", clthick);
    fprintf(OUTSTREAM, "\\vskip\\linethick\n");
  }
```

This code is used in sections 5 and 34.

34. Generating the closing TEX source code for the labels.

```
⟨ Save closing source code 34 ⟩ ≡
  {
    if ((num_labels ≠ 2) ∧ (num_labels ≠ 4)) {
      fprintf(OUTSTREAM, "\\par\\nointerlineskip\n");
    }
    if (num_labels ≠ num_labels_per_page) {
      if (num_labels ≠ 0) {
        if (generate_crop_marks) {
          ⟨ Write horizontal crop marks with height measure 33 ⟩
          ⟨ Write vertical crop marks with width measure 32 ⟩
        }
      }
      ⟨ Close output page 31 ⟩
    }
    fprintf(OUTSTREAM, "\\bye\n");
    if (¬outfile_specified)
      fprintf(OUTSTREAM, "%%--------␣TEX␣CODE␣ENDS␣HERE␣-----------------\n");
  }
```

This code is used in section 5.

35. Compiling the output TEX source code. If the `-c` or `--compile` option was present at the command line during startup of DVLABEL, then the program will make use of DVIPS to compile the previously generated TEX source code into PostScript. If the `-c` or `--compile` option is used together with a specified output file (specified using the `-o` or `--outputfile` options), then this output file will be compiled. On the other hand, if the `-c` or `--compile` option is used while the program should send the generated source to *stdout*, then the internally stored lines of output will be compiled instead (hence eliminating the need for a temporary file for the compilation).

```
⟨ Compile source code 35 ⟩ ≡
  {
    if (compile) {
      if (verbose) {
        fprintf(stdout, "%s:␣Compiling␣the␣TeX␣source␣code␣into␣PostScript.\n", progname);
      }
      if (outfile_specified) {
        if (eps_output) {
          sprintf(tmpstr, "tex␣%s;␣dvips␣%s␣%s␣-E␣-o␣%s.eps", outputfilename, dvipsopts,
              outputfilename, outputfilename);
        }
        else {
          sprintf(tmpstr, "tex␣%s;␣dvips␣%s␣%s␣-o␣%s.ps", outputfilename, dvipsopts, outputfilename,
              outputfilename);
        }
        system(tmpstr);      /* System call to execute tmpstr */
      }
      else {
        fprintf(stdout, "Compiling␣terminal␣output:␣Not␣implemented␣yet.\n");
        exit(FAILURE);
      }
    }
  }
```

This code is used in section 5.

36. Index.

DVLABEL