M.I.T. Laboratory for Computer Science
Computer Systems Research Groups

Request For Comments 255

April 19, 1984

Dover printing with the IBM PC

by J. H. Saltzer

This memo describes several methods and paths by which a user with an IBM Personal Computer can obtain output from one of the Xerox Dover laser printers located at 545 Technology Square. All of the techniques assume that the user has available the PCIP network package and hardware for the building Ethernet, the proNET ring, or a serial line operating at 1200 bits/second or higher that is connected to (or can be dialed up to) the PC serial line gateway.

1) *The basic path to the Dover.* Text files on the IBM PC use the ASCII character set with standard interpretations of carriage motion characters. Therefore one can simply send a PC text file to any L.C.S. host computer that has an implementation of TFTP and also the ability to send ASCII files to the Dover. One procedure is to log in to the other host using PC/telnet and use that host's TFTP command to transfer the file from the PC to the intermediate computer. Then, use that host's standard commands for formatting an ASCII file for the dover. This technique works on MIT-XX, MIT-Multics, and any L.C.S. VAX running UNIX, using commands named "press" or "pressify". The Dover formatting commands provide a limited number of options for controlling the appearance of the file on the Dover, primarily choice of font, margins, and interline spacing.

2) *Automating the path.* It is quite easy to set up a demon process on a VAX running BSD4.2 UNIX, which process will every so often wake up and look to see if any files have appeared in its input directory. The demon can then automatically format the files for the Dover using press with some predetermined option settings, and send them on to the Dover. A C-shell script is attached that can do this job. (This C-shell script can be found on MIT-BORAX in the file */usr/dover/pcpress.*) One starts this script going as a separate task by typing the UNIX command: "pcpress start &" and then logging out. The task will continue to run as long as the VAX UNIX system stays up. Alternatively, it is possible to prime the system initialization procedures to kick off such a task automatically when the system is bootloaded, by including appropriate entries in the file */etc/rc*. Once such a demon is started, a PC user sends, with PC/tftp, the file to be printed to the directory that the demon is watching on the VAX. Because UNIX has minimal facilities for waiting for events, this shell script uses an obscure mechanism to discover the existence of a file: One sends a file named "name.fix10" to be printed, then sends a file named "name.ctl" to start the demon. This second file must be less than one tftp block (512 bytes) in length, but its contents are irrelevant—the demon is waiting for the appearance of any non-zero-length file with secondary name ".ctl". The cover page appearing at the Dover will have the notation "For: name". As soon as it has processed the file "name.fix10", the demon deletes both files, so that the name "name.fix10" can be used for another file. Since UNIX/tftp will not allow sending a file whose name already exists, there is no danger that sending a second file will overwrite an earlier file before it has been processed by the demon.

3) *Using Scribe.* One can use an editor on the PC (such as Mince or the FinalWord editor) to prepare a file containing text as well as standard Scribe format control commands. The next step would be to log in to a Scribe-implementing host, such as MIT-XX or a VAX UNIX system, get the file from the PC using tftp, and then invoke Scribe just as if the file had been prepared using the host's editor. This approach uses the PC only for editing. If one is careful to restrict formatting commands to those that lie in the overlapping subset of Scribe and FinalWord, then draft copies can be made on a printer directly attached to the PC using FinalWord's FWF and FWP commands or the FinalWord editor's "advanced print" feature. After one is satisfied with the draft version, Dover copies can then be produced with Scribe. Moving files from FinalWord to Scribe can be done easily by a person familiar with both control languages, which are in many respects identical. However, because Dover output generally looks quite different from a typical PC printer output, one almost always wants to adjust the style a bit when switching from FinalWord to Scribe.

4) *Using FinalWord as a Dover formatter.* The FinalWord formatter (FWF) will arrange that the printer output driver write its output to a file if you give it (FWF) the options "-print -o filename". The resulting file can then be sent to the Dover, using either method 1) or 2) above. The trick is to convince FinalWord that the printer has no special features whatever. This convincing is accomplished by telling FinalWord that the printer device is the one named "plain". That is, one adds the option "-dev plain" *ahead* of the "-print" option. The result will be output that is formatted for a ten-pitch (pica) typewriter.

5) *Defining a FinalWord printer device.* Much better results can be obtained if one uses the FinalWord "CONFIG" program to define a new printer device that is more closely suited to producing output for the press command and the Dover. One chooses a set of options for press that specifies fonts to be used, vertical spacing, and paper position, and then configures a FinalWord printer device with parameters that correspond to these option choices. There is a file named */usr/dover/DCONFIG.DAT* on MIT-BORAX that can be used as a "CONFIG.DAT" file for FinalWord and that has a device named "Doverfix10". This device has the proper character and line spacing for the Dover's GACHA10 fixed width font. The C-shell script "pcpress" contains the press command control options that correspond to this printer definition.

6) *Font switching.* The press command can accept font-switch control characters. If an ASCII SO character (octal 016, obtained by typing control-N on most systems) appears in the text stream, the next character is taken to be the identifier of the font to switch to. The press command allows the user to specify up to four fonts to be used in the preparation of any one file. It identifies these four fonts as "T", "K", "H", and "C", and those letters are the ones that may follow the ASCII SO character. This feature can be invoked from FinalWord by including as part of a device specification the appropriate font-switch sequences. One trap is that the FinalWord package assumes that bold italic output can be obtained by sending the printer's control sequence for bold followed by the printer's control sequence for italic. Since the Dover's fonts are switched independently, one must treat bold italic as a separate feature rather than a combination of two features. The easiest way is to use the "Greek" controls of FinalWord to output the bold italic font sequence. (The device "Doverfix10" mentioned in 5) above is configured to send font change sequences when FinalWord encounters bold, italic, and greek—for bold italic—control commands, and the press options are set accordingly in the pcpress C-shell script.) *N.B. The font-switch*

feature is available only in the VAX version of press.

7) *Using proportional-spaced fonts.* Using FinalWord to take advantage of proportional-spaced fonts on the Dover is tricky, but feasible. The first step is to identify the font you want to use, and to obtain a table of font widths for that font. The standard Dover font widths are expressed in micas, the same measure that FinalWord requires. The next step is to modify one of the FinalWord font width tables to contain these font widths. Then, configure a printer definition to use that font width table. Finally, at the host computer you use to convert the ASCII file into Dover format, you must tell the press command to use the chosen font. This path works amazingly well, with two important restrictions. First, the only spacing facility that can be invoked by using the ASCII character set is the space character, so output ends up getting right-justified only to the nearest space. (In the TIMESROMAN fonts, a space is quite small, so one has to look carefully to notice spacing errors arising from this restriction.) Also, FinalWord cannot accomplish right-justification by smoothly stretching the line, so it just throws extra spaces in here and there. Second, FinalWord can cope with only one font width table at a time, so font-switching doesn't work as well as one might hope. It is essential to stay within one font size (as in using TIMESROMAN11 in plain, bold, italic, and bold italic forms) and even then there is a small variation in widths among the forms. The primary effect of the small variation is that the right-adjustment process comes out a little wrong on lines that contain characters with multiple fonts. (If you look for large right-adjustment errors in this RFC you will note that they are correlated with use of bold and italic fonts.) Titles and other things that are not right-adjusted look just fine. The file DCONFIG.DAT mentioned above also contains specification for a device named "DoverTR11" that contains font widths for the Dover's TIMESROMAN11 font, as well as font-switching sequences for *italic,* **bold,** and ***greek (bold italic).*** The press command control options that correspond to this printer definition may be seen in the pcpress C-shell script. The C-shell script automatically chooses this set of options if it is asked to process a file with secondary name ".tr11".

Hints and kinks.

This section describes some more obscure possibilities and problems that may be encountered in using the Dover from an IBM PC.

1). In the TIMESROMAN11 font, underlining does not work because the single width of the underscore and backspace characters can not match the many widths of the printing characters. For this reason, it may be more useful to translate the underscore character to some other graphic available in the font. Reassignment is accomplished by adding to the FinalWord printer definition a translation table, and then setting up the translation table to translate most characters into themselves. The underscore character (decimal 95) may be set to translate into, for example, the one-em dash (TIMESROMAN11 font value 19, typed as control-S.) The DoverTR11 device configuration includes this translation as well as a translation of the ASCII minus sign to the TIMESROMAN11 one-en dash. In table values, decimal character 45 translates to 22, typed as control-V. (The TIMESROMAN11 representation of the ASCII minus sign is a bit narrow for some tastes.) Note that since the ASCII code values for these TIMESROMAN11 graphics are in the usual range of control characters, the press command will convert them to a graphic representation of the control character (such as ↑V or ↑S) unless it is given the parameter @NOCTL in the command line.

2) The press command performs fixed vertical spacing. When using more than one font, it determines the amount of vertical spacing by identifying the font with the greatest height above the base line and the font with the greatest descent below the base line. It adds the greatest height to the greatest descent, then adds the percentage indicated by its @LEADING control parameter (20% default.) This algorithm tends to lead to more interline spacing than one expects when using related fonts in a family, since one font may be defined as having high upward extents, while the next has long descenders. For the TIMESROMAN11 family including bold and italic, a leading of 10% leads to about the same interline spacing as is obtained with leading of 20% for TIMESROMAN11 alone, 510 micas.

3) FinalWord is not very defensively designed against the possibility that it has been given printer device definitions that are internally inconsistent. In particular, if the specification of smallest horizontal and smallest vertical movement parameters is not the same as the specification of the width of the space character and the vertical distance moved by a new-line character, FWP can go into a loop when it starts to try to produce file output. Depending on details, this loop may or may not involve filling up the disk. One must be careful.

4) There seems to be some kind of an obscure interface problem between the press command and the Dover, which causes the top half inch of press output to lie off the top of the paper. For this reason the @TOPMARGIN: parameter of press in the pcpress C-shell script is set to 1000 micas, an amount that (experimentally) makes the first line lie at the very top of the paper.

5) It is helpful to put together a PC batch script that first invokes FWF with the proper parameters, then uses PC/tftp to send the text file and the control file. Two examples of such batch scripts can be found in the files */usr/dover/doverfix.bat* and */usr/dover/dovertr.bat* on MIT-BORAX. A copy of the latter batch script is attached. However, the PC batch facility is not quite as sophisticated as the UNIX C-shell, so the resulting batch script is not nearly as user-friendly or bullet-proof as it should be.

6) The C-shell scripts that invoke press on UNIX could be replaced with C-language programs, to gain both a speedup in their operation and also allow some improvements in function. The primary improvement that might be gained would be to allow press parameters to be supplied in the control file. That feature would make the full flexibility of press directly available to the PC.

7) The Doverfix10 device mentioned above is parameterized so as to closely match the DoverTR11 device, so that it can be used for program examples that are to be pasted into text. For general printing use, it would be appropriate to reduce its interline spacing by setting the Press @LEADING parameter to 20% and the Doverfix10 device vertical height and minimum vertical movement to 415 micas.

8) It would be plausible to automate the invocation of SCRIBE on a VAX UNIX using a shell script similar in spirit to pcpress.

C-Shell file for UNIX press+dover demon

```
#! /bin/csh
#
#     Trivial press+dover demon for PC output to dover.  3/11/84.
#
if ($1 == "start" ) then
  cd /usr/spool/dover
  echo -n "Starting trivial press+dover demon:  " >>spool.log
  date >>spool.log
  set nonomatch = 1
  while (!(-e stop))     #  A file named "stop" will stop this demon.
    foreach i (*.ctl)
      if (-e $i) then
        if (!(-z ${i})) then
          echo -n "Found control file ${i}:  " >>spool.log
          date  >>spool.log
          if (-e $i:r.tr11) then
              /bin/press $i:r.tr11 @noctl @nohdr @nolit \
                     @top:1000 @bottom:0 @left:0 \
                     @shift @leading:10 \
                     @tfont:timesroman11 \
                     @cfont:timesroman11b \
                     @kfont:timesroman11i \
                     @hfont:timesroman11bi \
              |/bin/dover -user: $i:r >>spool.log
              rm -f $i:r.tr11 >>&spool.log
          else if (-e $i:r.fix10) then
              /bin/press $i:r.fix10 @noctl @nohdr @nolit \
                     @top:1000 @bottom:0 @left:0 \
                     @shift @leading:48 \
                     @tfont:gacha10 \
                     @cfont:gacha10b \
                     @kfont:gacha10i \
                     @hfont:gacha10bi \
              |/bin/dover -user: $i:r >>spool.log
              rm -f $i:r.fix10 >>&spool.log
          else
            echo "but no *.tr11 or *.fix10 file to print for $i:r "\
                                  >>&spool.log
          endif # (-e $i:r.fix10)
          rm -f $i >>&spool.log
          echo "Control file $i processed."  >>spool.log
          continue    #  Skip wait, repeat while from the start.
        else
          sleep 30              #  Wait for the rest of this file to arrive.
          if (-z ${i}) then     #  Not arriving, get rid of it.
            echo -n "Discarding zero-length file ${i}:  "  >>spool.log
            date >>spool.log
            rm $i >>&spool.log
          endif # (-z ${i})
          continue
        endif # (!(-z ${i}))
      endif # (-e $i)
    end # foreach i (*.ctl)
    sleep 100 >>&spool.log
  end # while (!(-e stop))
  echo -n "Stopping trivial press+dover demon:  " >>spool.log
  date >>spool.log
  rm stop
  exit
else if ($1 == "stop" ) then
  cp pcpress stop
  exit
else
  echo "Trivial press+dover demon.  Usage:"
  echo "          pcpress start &"
  echo "or"
  echo "          pcpress stop"
  exit
endif # ($1 = start )
```

PC Batch file to send output to a mythical UNIX named MIT-SOAPSUDS

```
echo off
echo *****  Preparing FinalWord output for Dover TimesRoman 11  ****
if exist dover.fwo erase dover.fwo
fwf %1 -dev Dovertr11 -print -o dover.fwo -q
if not exist dover.fwo goto else1
   tftp put dover.fwo MIT-SOAPSUDS /usr/spool/dover/Smith.tr11
   tftp put %0.bat MIT-SOAPSUDS /usr/spool/dover/Smith.ctl
   if not errorlevel 1 echo *****  File transmitted For:  Smith *****
   goto end1
:else1
   echo *****  FinalWord format error, nothing sent to Dover  ****
:end1
```