University of Edinburgh

Department of Computer Science

LAYOUT

by

P. McLellan

Internal Report

CSR-21-78

James Clerk Maxwell Building
The King's Buildings
Mayfield Road
Edinburgh
EH9 3JZ

February, 1978

LAYOUT

A document production program

LAYOUT is a program, written by Hamish Dewar, for the production of documentation from an input consisting of the text of the document interspersed with simple commands. The program is available on many machines, in particular, the Computer Science Department's PDP9, PDP15 and Interdata systems, ERCC's EMAS and 2980 systems, and the SRC's DECsystem 10 and the PDP11 running DEIMOS or DOS.

P. McLellan
February 1978, revised September 1978.

Contents

LAYOUT is a program which produces a paged formatted document from a source file. The motivations for this are twofold. Firstly, the formatting of the document can easily be altered by simple editing of the source file. Secondly, any changes or updates to the document can be effected by editing the source file, with any effects of the changes on the formatting of the document handled automatically. In order to simplify editing, LAYOUT also produces an updated version of the source file, with a line structure similar to that of the output document. This document is an example of the output from LAYOUT.

The source file

The source file consists of text interspersed with directives to LAYOUT. Directives are described in detail later on. The basic operation of the program is to read words from the source file and transfer them to the current line of the output file separated by single spaces; a new line is started when a word will not fit on the current line. If the word ends a sentence, then extra spaces are inserted. A word is any sequence of characters up to (but not including) a space, newline or directive; thus a word is usually a sequence of letters, perhaps with some punctuation at the end. If a word ends with any of ".!?" and the following word begins with a capital letter, then it is treated as ending a sentence. The spacing and line structure of the source is discarded; directives to the program are required to output extra spaces, blank lines and so on. If requested, lines can be fully justified (as these ones are).

Shift conventions

For ease of typing, especially on devices which do not support lower case letters, LAYOUT interprets the source file using a shift convention. All letters are forced to lower case, except that any letter following an "@", and the whole of any word starting with a "." is capitalised (naturally, the "@" or "." does not appear in the document). Any character following a "_", and the whole of any word starting with a "%" is underlined, though punctuation symbols at the end of words are not underlined. Combinations, such as "%@" for an underlined word starting with a capital letter are allowed; ".%" must be used (rather than "%.") for underlined capitalised words. When typing running text, these conventions soon become automatic. The shift convention can be altered or disabled, but this is not explained unt later. For example, if the source file contained the lines:

```
@MOST LETTERS ARE FORCED TO %LOWER CASE BY .LAYOUT. @SOME
ARE LEFT IN .%UPPER CASE. @SHIFT CHARACTERS CAN
APPEAR WITHIN WORDS, LIKE A@LP_H@ABET.
```

then LAYOUT would output:

> Most letters are forced to lower case by LAYOUT. Some are left in UPPER case. Shift characters can appear within words, like aLphAbet.

Directives

Directives consist of a "$" followed by a letter. A "$" followed by a character other than a letter causes that character to be treated literally, and not accorded any special significance. In particular, "$$" outputs "$"; "$ " outputs " " but does not terminate the word being read (so the space will be underlined if the word is, and the space will not be extended during justification); "$." at the end of a word inhibits recognition of the end of a sentence (useful with initials).

The effects of the various directives are detailed later, but three require special explanation.

The $A directive

The details of operation of LAYOUT are governed by a number of parameters. These can be altered at any time by a $A assign directive. A $A directive consists of a source line containing "$A" followed by a number of assignments, separated by semicolons, of the form "<parameter>=<value>". The <value> can be either a number or another parameter name. If it is an unsigned number then this is the new value of the parameter; if it is a signed ("+" or "-") then the value increments or decrements the current value of the parameter. If it is another parameter name, then the new value is the current value of that parameter. For example:

```
$A PAGENO=17
$A LEFT=4; LINE=68; NLS=2
$A LINE=-8; LEFT=+4
$A TOP=BOTTOM
```

Amongst the parameters alterable by an assign directive are those specifying the page size and margins. These will normally be set in a $A directive at the start of the source file, though they may be altered at any time. The parameters are:

TOP The number of blank lines to be left at the top of each page to form the top margin. This is initially set to 2.

BOTTOM The number of blank lines to be left at the bottom of the page to form the bottom margin; initially 4.

PAGE The number of lines in the page available for text, not counting any top and bottom margins; initially 60. PAGE+TOP+BOTTOM should add up to the number of lines in the physical page, probably 66. If set to zero, then page turns are suppressed and the document is produced as a continuous run like a galley proof.

LEFT The number of spaces to be left at the start of each line to form the left hand margin; initially zero (no left margin).

LINE The number of characters available in the line for text, not counting any left margin; initially 72. The difference between the physical line length and LEFT+LINE gives the right hand margin width.

PAGENO The page number. If non-zero (it is initially zero) then at the bottom of each page the PAGENO is printed (in the centre of the middle line of the bottom margin) and the PAGENO is incremented by one.

SECTNO The section number. If non-zero (it is initially zero) and PAGENO is non-zero, then page numbers are printed as section-page (for instance "2-13").

MARK Page marker flag. Pages are filled out to the physical page length with blank lines. If MARK is zero (as it is initially) then no further action is taken; if it is one then pages are additionally delineated by a line containing only a "=" as the first and last characters (useful if using continuous roll stationary); if set to any other value then pages are separated by formfeeds (Ascii 12).

As already explained, LAYOUT inserts words into the output line separated by single spaces, inserts extra spaces at the end of sentences and outputs the line, possibly having justified it. The details of this operation are under the control of more parameters alterable by a $A directive as follows:

SGAP The sentence gap, the number of spaces to be inserted at the end of a sentence (if it is not the last word on the line). Unless altered this is 2 spaces.

PGAP The paragraph gap, the number of spaces by which the first line of a paragraph (signalled by a $P directive, see later) is indented, initially 3 spaces.

JUST Justification flag. If non-zero (it is initially zero) then lines which are output because the next word will not fit are fully justified. Incomplete lines (such as the last line of a paragraph) are not justified unless they are output because of a $J directive (see later).

NLS Line spacing. This is initially one, so that the document is single spaced. If it set to more than one, then extra blank lines are inserted after each document line to double (or more) space the output.

The $L directive

Sometimes, particularly for headings and titles (such as those on the first page), the normal action of LAYOUT is inappropriate. To deal with this, the $L lines directive is used. This causes the document to be structured as it is in the source file. The spacing and line structure of the source file are preserved; the usual shift convention is, however, observed. A $L directive consists of a line containing "$L" followed by the number of source lines to be copied in this way; if the number is zero then all source lines up to the next directive (which must start a line) are copied. Following the number, one or more of a number of modifiers may be present. These are:

C: Capitalise all letters.
U: Underline all characters (including spaces).
M: Middle; centralise each line.

If the U modifier is used, then each line is regarded as a single word; in particular the whole word capital shift (".") is only accepted at the start of the line (when it capitalises the whole line).

The directive must always be the last item on a line. The copying begins with the following line. For example, the heading to this section could be generated by the following source lines:

```
$L1U
@THE $$@L DIRECTIVE
```

The $E directive

The $E directive is used to terminate the source file. It must occur in every source file (as the last item!). It consists of a line containing just "$E". LAYOUT fills out the current page with blank lines and stops.

The other directives

As described above, to get extra blank lines, new pages and so on, directives are used. The remaining directives are all concerned with spacing and page turning. All these directives terminate the current output line (unless it is empty), as do the $A, $L and $E directives already described. Many are followed by a number, which, if omitted, defaults to one. The directives are as follows:

$B<num> Blank lines. If there are more than <num> lines left on the current page then output <num> blank lines, otherwise start a new page. If a new page was taken just prior to the directive, then the blank lines are discarded unless the page was taken as the result of a $N or $V directive (see below). Note that $B0 can be used to terminate the current output line.

$N Newpage. The current page is filled out with blank lines and a new page is started. However, if the current page is empty (no lines, even blank ones, printed) then the directive is ignored. Because of this rule, to get a blank page use "NB$N".

$P<num> Paragraph. A new paragraph is started by outputting <num> blank lines, and indenting the next line by an extra PGAP (see the $A directive) spaces. If there are fewer than <num>+2 lines free on the current page, then a new page is started in place of the blank lines.

$V<num> Verify. A new page is started if fewer than <num> lines remain on the current page. This can be used to ensure that headings do not occur as the last line on a page, that a section is unbroken by a page boundary and so on.

$J Justify. The current output line is terminated, and justified if JUST (see the $A directive) is non-zero.

$S Section. A new section is started; a new page is started, the section number is increased by one and the page number is reset to one. For example, if the current page was numbered "2-7", then the next (automatically started) page would be numbered "3-1".

An example

The features detailed above are sufficient for the simple use of LAYOUT in preparing running text, interspersed with headings. There are further features concerned with indentation, tabulation and altering the shift conventions which are described later. As an example, suppose that the following output was required:

THE COLOURED BOWLING PINS

A wealthy man had two bowling lanes in his basement. In one lane ten dark coloured pins were used; in the other, ten light-coloured pins. The man had a mathematical turn of mind and the following problem occurred to him one evening as he was practising his delivery:

Is it possible to mix pins of both colours, then select ten pins that can be placed in the usual triangular formation in such a way that no three pins of the same colour will mark the vertices of an equilateral triangle?

If it is possible, show how to do it. Otherwise prove that it cannot be done.

then the source file would need to contain something like:

```
$A LEFT=9; LINE=54; JUST=1
$V8 $L1UMC
THE COLOURED BOWLING PINS
$P1@A WEALTHY MAN HAD TWO BOWLING LANES IN HIS
BASEMENT.  @IN ONE LANE TEN DARK-COLOURED PINS WERE
USED; IN THE
OTHER, TEN LIGHT-COLOURED PINS.  @THE MAN
HAD A MATHEMATICAL TURN OF MIND, AND
THE FOLLOWING PROBLEM OCCURRED TO HIM
ONE EVENING AS HE WAS PRACTISING HIS
DELIVERY:  $P1@IS IT POSSIBLE TO MIX
PINS OF BOTH COLOURS, THEN SELECT TEN PINS THAT
CAN BE PLACED IN THE USUAL TRIANGULAR FORMATION IN SUCH
A WAY THAT NO THREE PINS OF THE SAME COLOUR WILL
MARK THE VERTICES OF AN EQUILATERAL TRIANGLE?
$P1@IF IT IS POSSIBLE, SHOW HOW TO DO
IT.  @OTHERWISE  PROVE THAT IT CANNOT BE DONE.
```

Note the following features of the source. The spacing and line structure is not particularly systematic since it is ignored by LAYOUT. A $V directive is used before the $L directive to ensure that the heading to the section is not left at the bottom of a page. The first line is a $A directive setting up the page parameters.

Tabulation and indentation

LAYOUT maintains a vector of tab settings which can be used for indentation and tabulation. This is under control of further directives as explained below, and two more parameters to the $A directive as follows:

TAB Tab settings. Unlike other parameters, TAB takes a list of up to 25 values, separated by commas. These are the new tab settings. Initially, tabs are set every 8 characters, at columns 9, 17, 25, 33 .. The first number is tab position one; tab setting zero is always column one.

INDENT The tab setting to which all lines are automatically indented. If zero (as it is initially) then lines are not indented. It must be emphasised that the value is a tab setting and not a character position.

The tab settings can clearly be used for producing fully tabulated tables, but they are more often used to specially format paragraphs, such as those above (TAB...). The following directives control tabulation, or override the default indentation, in conjunction with the tab settings.

$I<num> Indent (for one line only). If <num> is unsigned, then a new line is started and indented to tab setting <num>, irrespective of the current value of INDENT (see above). If <num> is signed, then the new line is indented by an extra <num> tab settings from INDENT (if <num> is negative then it is indented less than usual). Thus "$I-1" means indent by one less tab setting than usual. This directive is especially useful when INDENT is non-zero, to override it for one line (for example, the first line of this paragraph).

$T<num> Tabulate. If <num> is unsigned, then tab to setting <num> (a new line is not started for this directive!). If it is signed positive, then tab forward <num> times (at least one space for each). If it is negative, then tab backwards <num> times (at least one space for each); the columns passed back over must all be blank. If justification is on (JUST non-zero) then only the text following the last $T directive is justified; thus justification will never upset columns aligned by $T.

$C<num> Column. This directive is exactly the same as the $T directive above, except that <num> is a column (character position), not a tab setting. Thus "$C+3" will leave three spaces, "$C20" moves to column 20, and so on.

$L (Again). There is a further modifier I (besides U, M and C, see earlier) to the $L directive. This causes all copied lines to be indented to the current setting of INDENT.

For example, the paragraph above on the $C directive is produced by the following source text:

```
$A INDENT=1
$I0 $$@C<NUM> $T1 @COLUMN.  @THIS DIRECTIVE IS
EXACTLY THE SAME AS THE $$@T DIRECTIVE ABOVE, EXCEPT THAT ..
```

As an example of how the tab settings can be used to produce tables, consider the follwing source:

```
$A INDENT=3
$A TAB=3,28,44
$I1@ROBINSON$T2 01-557-4356$T+1 48 @SOME @STREET, @NEASDEN,
@LONDON.
$B$I1 @SMITH $T3 @NOPHONE @HOUSE, @BIRMINGHAM.
$B1$I1 @SOMERVILLE $T2 @MUDDLESVILLE 435674 $I3
32 @HIGH @STREET, @MUDDLESVILLE, @SUMCOUNTY.
```

which would produce the following output:

Robinson	01-557-4356	48 Some Street, Neasden, London.
Smith		Nophone House, Birmingham.
Somerville	Muddlesville 435674	32 High Street, Muddlesville, Sumcounty.

When tabulating numerical data (as in the example above), a space must follow the numerical part of directives followed by a number; for example, in the table above, if "$T2 01-557.." had been typed as "$T201-557..", then it would have been read as tab 201 times.

The $A directive revisited

The $A directive described above can in fact store values of parameters for later recall. This facility depends on the assignment operator as follows:

`=` Simply assign the new value to the parameter. This is the case already described.

`<=` Save the current value of the parameter before assigning it the new value.

`>` Reinstate the last value saved for the parameter.

`<` Save the current value of the parameter but leave its value unaltered.

`>=` Reinstate the last value saved and then assign a new value; only really useful if the new value is an increment (signed number) on the old value.

The operators may be used repeatedly, so that more than one value of a parameter may be stored. These stored values are retrieved on a first in, last out basis.

The usual use for this facility is to preserve an environment and then reinstate it. For example:

```
$A INDENT<=2; JUST<=1
@THIS TEXT WILL BE INDENTED TO TAB SETTING 2
AND JUSTIFIED IRRESPECTIVE OF THE ORIGINAL
SETTINGS OF JUST AND INDENT.
$A INDENT>; JUST>
@THIS TEXT WILL BE INDENTED TO THE ORIGINAL TAB SETTING.
```

Another use for this facility is to stack a number of settings for a directive (using "<=") for later retrieval (using ">"). For example:

```
$A INDENT=3; TAB=4,8,12,16,20
$A INDENT<=2; TAB<=3,6,9,12,15
$A INDENT<=1; TAB<=4,8,20,32,40,44
```

If these directives occurred at the start of the source, then the settings stored could be instated by a statement:

```
$A INDENT>; TAB>
```

The updated source file

A version of the source file, the updated source, can be output by LAYOUT. Each line of the updated source contains either only directives, or contains the source text for a line of the document (possibly containing $C or $T directives), subject to a line length restriction:

SLINE The maximum line length for the updated source file; lines which would exceed this length are split into two.

Inhibiting output

When preparing a large source file, it is often wasteful to print the entire file, but it is not usually feasible merely to process a section of the source, since directives early in the file can have effects later on. To simplify inhibiting output, the following parameters, alterable by the $A directive are provided.

START The first page of output to be printed. LAYOUT still constructs earlier pages but these are suppressed. Note that START refers to a number of pages, and not the page number, though these will often be the same.

FINISH The last page of output to be printed. LAYOUT stops just as if a $E had been encountered before the next page.

IGNORE If non-zero, then LAYOUT ignores everything in the source file apart from $A and $E directives. Each time it is set back to zero, a new page is started and processing continues normally.

One of two methods are normally used to suppress output using the above parameters. If it is known which pages are to be output, then START is set to the first page and FINISH to the last page. If it is only known which

sections of the source file are to be output, then IGNORE is set non-zero before the first page, zero before each section to be printed, and non-zero again afterwards. Of course, one way to prematurely stop output is to edit a "$E" into the source file.

If any of these techniques are used to suppress output, then an updated source file should not be produced, since, in general, it will be incomplete.

Shift conventions revisited

As mentioned in the earlier section on shift conventions, the conventions can be altered. This is done by yet more parameters to the $A directive. As well as being able to alter the conventions for the source file, it is possible to alter the conventions for the updated source file so that it is produced to a different convention to the original source. The right hand side of some of the assignments is a character. This should be quoted (for example, '@'). The parameters are as follows:

INVERT If non-zero, as it is initially, then all upper case letters are inverted to lower case (and lower to upper). If zero, then letters are left in the case in which they are read. Thus if the letters in the source file are all in the correct case, then INVERT should be set to zero.

CAP The single character capital marker, initially (and usually) '@'.

UND The single character underline marker, initially '_'.

CAPSH The whole word capital shift, initially '.'.

UNDSH The whole word (actually rest of word) underline shift, initially '%'.

ESCAPE The directive marker, initially '$'. If followed by a character other than a letter, that character is treated literally, as explained earlier. Directives are accepted in upper or lower case.

CAPO, UNDO, CAPSHO, UNDSHO, INVO
These are the equivalent shift characters to be used for the updated source file, and have the same initial values as their corresponding parameters for the source.

ASCII If non-zero, as it is initially, then underlining will be implemented by carriage return (ascii 13) and overprinting with underscores. If it is zero, then underlining is indicated by setting the eighth bit of the code of the underlined character. Whatever the value of ASCII, LAYOUT accepts this convention in the source file.

If the parameters for the source file are different from those for the updated source (so that the shift conventions will be changed), then the $A directives defining the shift conventions will need to be altered before the updated source is suitable for reinputting to LAYOUT, since they will have been copied across verbatim and will thus refer to the old source conventions.

If any of the source shift characters is set to zero, then that shift is disabled; for example, if CAPSH is set to zero then no character will be recognised as capitalising the whole of the following word. If one of the updated source parameters is set to zero then the updated source will be produced without making use of that shift; for example, if CAPSHO is set to zero then words will be capitalised by changing their case (in particular, if INVO is zero, then capitalised words will be output in upper case). If any of the underline shifts is set to zero, then underlining is indicated by setting the eighth bit of the underlined character.

The parameters can be used completely independently; for example, to output the line:

The cat sat on 2 MATS.

all of the following lines would be acceptable:

```
   @THE %CAT SAT ON _2 .MATS.

$A CAP='*'; CAPSH='#'; UND='&'
   *THE %CAT SAT ON &2 #MATS.

$A INVERT=0; UND='_'; CAP='@'; CAPSH='.'
   @the %cat sat on _2 .mats.

$A INVERT=0; CAP=0; CAPSH=0
   The %cat sat on _2 MATS.

$A INVERT=1; CAP=0; CAPSH=0
   tHE %CAT SAT ON _2 mats.
```

Errors

In the event of errors being detected in the source then a message (self-explanatory) is printed on the report stream (normally the user's console) followed by the offending line as it appears in the updated source (note, not the original source). For example, the source line:

```
$I2@THE  CAT SAT ON $T1 THE $Z MAT.
```

Would produce the following on the report stream:

```
*OVER TEXT T
$I2 @THE CAT SAT ON $T1 THE
*UNKNOWN DIRECTIVE Z
$Z MAT.
```

Summary

The following preamble can be regarded as being on the start of every source file; it lists every parameter with its intial value.

```
$A TOP=2                    Top margin.
$A BOTTOM=4                 Bottom margin.
$A PAGE=60                  Usable page length.
$A LEFT=0                   Left margin.
$A LINE=72                  Usable line length.
$A NLS=1                    Newlines per line.
$A SGAP=2                   Spaces between sentences.
$A PGAP=3                   Paragraph indentation.
$A PAGENO=0                 Page number.
$A SECTNO=0                 Section number.
$A START=1                  First page printed.
$A FINISH=9999              Last page printed.
$A IGNORE=0                 Ignore flag.
$A JUST=0                   Justification flag.
$A MARK=0                   Page separator.
$A INDENT=0                 Indentation tab setting.
$A TAB=9,17,25,33,41,49,57,65,73,81
                            Tab settings.
$A ASCII=1                  Underlining procedure.
$A ESCAPE='$'               Directive escape character.
$A CAP='@'                  Capitalise single letter.
$A CAPSH='.'                Capitalise whole word.
$A UND='_'                  Underline single character.
$A UNDSH='%'                Underline rest of word.
$A INVERT=1                 Case inversion of source.
$A SLINE=80                 Maximum updated source line length.
$A INVO=INVERT              Corresponding
$A CAPO=CAP                   parameters
$A CAPSHO=CAPSH                 for
$A UNDO=UND                       updated
$A UNDSHO=UNDSH                     source.
```

Examples of all the directives follow, together with a note of their effects on the output.

```
$A INDENT=+2; ESCAPE<='&'; JUST=1
```
Increment INDENT by 2; save the current ESCAPE character and set it to '&'; set JUST to 1.

```
$A ESCAPE>; INDENT=-1; CAPO=CAP
```
Reinstate the saved value of ESCAPE; decrement INDENT by one; set CAPO to the current value of CAP.

```
$B3     Three blank lines.
$BO     Start a new line (do not justify old line).

$C7     Move to column 7.
$C+3    Advance 3 columns.
$C-2    Move back over 2 columns.

$E      Stop processing.

$I3     Indent to TAB(3).
$I+1    Indent to TAB(INDENT+1).
$I-2    Indent to TAB(INDENT-2).

$J      Start a new line (justify old line).

$L5UM   Copy five lines, underlining and centralising each line.
$LOIC   Copy up to the next directive, indenting and capitalising each line.

$N      Start a new page.

$P1     Start a new paragraph, preceded by one blank line.

$S      Start a new section, increasing section number by one.

$T3     Move to column TAB(3).
$T+2    Tab twice forwards.
$T-1    Tab once backwards.

$V7     Start a new page if there are less than 7 lines remaining on the
        current page.
```

Appendix 1 - use of LAYOUT

Interdata

LAYOUT is invoked on the Computer Science Department's Interdata systems by a command of the form:

```
LAYOUT source/document,usource
```

The updated source (or document) may be omitted if not required. For example:

```
LAYOUT MANUAL/MANUAL:LAY
LAYOUT MANUAL/LP
LAYOUT MANUAL/LP,MANUAL
LAYOUT MANUAL/,MANUAL:NEW
```

PDP9 or PDP15

LAYOUT is invoked on the Computer Science Department's PDP9 or PDP15s by a command of the form:

```
.LAYOUT source/document,usource
```

The updated source (or document) may be omitted if not required. For example:

```
.LAYOUT MANUAL/LP
.LAYOUT MANUAL/DT5 TEMP,NEWMAN
.LAYOUT DT3 DOC/PRETTY
```

EMAS

Before using LAYOUT on EMAS it is necessary to first append CSDEPT.LAYOLIB. It is then invoked by a command of the form:

```
LAYOUT source/document,usource
```

The updated source may be omitted if not required. For example:

```
LAYOUT MANUAL/.LP
LAYOUT MANUAL/MANUAL1
LAYOUT ECSC48.LDOC/MYCOPY
LAYOUT MANUAL/.NULL,MANUALN
```

ICL 2980

LAYOUT is invoked on the 2980 under VME/B by use of the macro LAYOUT, with parameters SOURCE, DOCUMENT and UPDATEDSOURCE (optional) nominating the files.

DECsystem 10

LAYOUT is invoked on the Science Research Council's DECsystem 10 by a command of the form:

```
R LAYOUT
```

The program will then prompt "Files:-"; type the filenames in the form:

```
output,usource=source,TTY:
```

The program will then prompt "SECTIONS (Y OR CR):"; type Y cr. For example:

```
R LAYOUT
Files:- MAN.LST,MAN.NEW=MAN.LAY,TTY:
SECTIONS (Y OR CR): Y
```

PDP11, DEIMOS

LAYOUT is invoked under DEIMOS by a command of the form:

```
LAYOUT source/document,usource
```

The updated source may be omitted if not required. For example:

```
LAYOUT FRED/GOOD,NEW
LAYOUT 3.JIM(35)/O.DOC
```

PDP11, DOS

LAYOUT is invoked under DOS by a command of the form:

```
R LAYOUT
```

The program then prompts "*"; type the filenames in the form:

```
document,usource=source
```

The updated source may be omitted if not required. For example:

```
R LAYOUT
*LP:,NEW=SOURCE
```

Other systems

LAYOUT is written in IMP and is thus easily implementable on any system which supports an IMP compiler.

Appendix 2 - use of Diablo document printer

The versions of LAYOUT on the computer science department's Interdata systems produce output suitable for Diablo document printers. This output is produced on output stream 3 in addition to the usual output on output stream 1. The exact form of the output is governed by more parameters to the $A described below. Note that none of them have any effect on the normal output - indeed, the document output and the Diablo output are always the same text in the same format, except that the printed area on the Diablo is moved across and down the page.

DCPI The number of characters per inch on the Diablo; this will normally be 10 (as it is initially) or 12 but in any case it is rounded to a multiple of 120ths of an inch.

DLPI The number of lines per inch on the Diablo; this will normally be 6 (as it is initially) but in any case it is rounded to a multiple of 48ths of an inch.

DLEFT The additional left hand margin in 100ths of an inch; this is initially 150 (1.5 inches).

DTOP The additional top margin in 100ths of an inch; this is initially 250.

DPAGE The actual physical page size on the Diablo in 100ths of an inch; initially this is 1700 (for 17 inch pages).

DHOLD The Diablo can be halted at the end of each page and waits until the HOLD button is pressed; if DHOLD is non-zero (as it is initially) then LAYOUT halts the printer at the end of each page.