Photon ROFF Text Formatter (PROFF)

Richard J. Beach

Department of Computer Science
University of Waterloo

Research Report CS-76-08

April 1976

Photon Roff Text Formatter (PROFF)

Richard J. Beach
Department of Computer Science
University of Waterloo

Abstract

PROFF is a text formatting program for preparing phototypeset output of English text and computer programs. It was developed as a transition tool between the line printer output of ROFF, and the phototypeset material from a yet to be implemented photo-composition system. The formatting command structure of PROFF is taken from ROFF, to allow for the convenient typesetting of existing documents, to preserve experience with ROFF, and to enable the use of ROFF as a proofing tool. Additional commands have been defined to establish those formatting parameters unique to a phototypesetter. However, since all formatting algorithms are implemented in the Photon Econosetter, any feature which relies on pagination or line breakage cannot be supported. Such features as footnotes and running heads and feet will not be possible until the photo-composition system is ready.

PROFF interfaces with the typesetter by either paper tape or a communications line. Timesharing commands are available for converting a PROFF unformatted text file into a file of typesetting codes. This converted file may then be punched on paper tape or passed to the Photon by another timesharing command.

This report is a sample of the output quality of PROFF.

Introduction

PROFF is a program for producing phototypeset master copies of documents prepared and edited on the Math Faculty's Timesharing system. This document was produced by PROFF as a demonstration of its capabilities. PROFF output is typeset on a Photon 737 Econosetter [2] onto photographic paper, which is then developed and cut into pages. These pages may then be used as masters in a reproduction system, such as photo-offset, resulting in many high quality copies.

PROFF was designed with 2 goals in mind:

1. to provide an easy transition from the popular ROFF text formatter [1],
2. to provide a tool for the early production of typeset quality documents.

ROFF was used as a base, since there existed several documents using ROFF formatting commands, which could then be easily typeset. Therefore, the PROFF format command structure is taken from ROFF. Furthermore, considerable experience with ROFF may now be utilized in typesetting documents. Also important is the ability to use ROFF as a proof reading tool for typeset documents. By the definition of an appropriate ROFF Macro file, any PROFF file may be printed at a terminal or the line printer for proof reading, hence avoiding the wasted time and supplies on phototypesetting proof copy.

In order to produce typeset documents early in our research project, it was necessary to utilize the Photon-supplied formatting software. This software provides justification, hyphenation, multiple column output, etc. and was deemed a reasonable initial subset of the photo-composition system. However, since all the formatting algorithms run in the Photon, no feature which relies on the occurrence of line breaks or page breaks can be implemented. Thus page numbers, footnotes, running heads and feet, and sophisticated mathematical text, will have to be delayed until the development of our photo-composition system. With these limitations, PROFF is best suited for the printing of English text, and computer programs.

PROFF provides the following formatting features:

- paragraphs, with lines filled with words moved into the output line to eliminate white space
- justification, the even distribution of spacing to achieve smooth right and left margins
- hyphenation, word breakage to partially fill the output line to further eliminate white space
- centred, flushed right, or flushed left headings
- indentation, both left and right margins
- hanging indents for point form
- multiple column output
- four typefaces: Roman, Boldface, Italics, and Math
- four typesizes: 6, 10, 14, 18 point (this is 10 point)
- variable line spacing, in units of 1/144 of an inch

However, as mentioned previously PROFF **will not** support:

- running heads or feet, or page numbers
- footnotes
- widow elimination (short lines at the end of paragraphs placed on the next page or column)
- leaving white space for figures (at least, not reliably, due to page breaks)
- automatic indices or table of contents.

This report covers the use of PROFF, and assumes some familiarity with the commands supported by ROFF [1]. Topics are discussed in a tutorial style, but with sufficient detail to exhaustively define the actions of each command. Several problem areas are described and some guidelines and solutions are offered. The TSS commands which are used to produce a PROFF document are explained, and techniques for using ROFF as a proofing tool for PROFF are outlined.

PROFF Formatting Commands

PROFF requires formatting commands to be imbedded in the text file. Commands are distinguished in that they must begin a new line, and the first character must be a dot ".". (If necessary you may define this character to be any one you desire. See the **.cc** command.) Commands take effect immediately upon recognition, and therefore must be placed in the text wherever the required effect is desired. Each command is recognized by 2 characters immediately following the dot, e.g. **.ce** is the centre command. Some commands accept arguments to set a parameter. Usually these are numbers, and often may be increments or decrements. Thus the margin may be indented 5 characters by the **.in5** command, or moved over 5 characters by **.in+5.** The former is absolute, the latter is relative.

Arguments which specify sizes, such as for indenting or line length, may optionally be assigned a unit. The units accepted are

h - half points (1/144 of an inch)
p - points (1/72 of an inch)
c - characters (1/10 of an inch)
i - inches (only integers, no decimal points)
l - lines (point-size + 2 points)

Some arguments are used to set a formatting mode parameter on or off, and thus the keywords **on** or **off** will be accepted. Most commands have a reasonable default argument value and unit size. The summary in Appendix B lists all commands, their default argument value, and default units.

Character Fonts and Sizes

Our Photon typesetter can print text in four fonts: Times Roman, *Times Italic,* **Times Boldface,** and a special Mathematics font, which contains the Greek alphabet and many Math symbols. Appendix A displays the complete character set for each font. Text may be printed in four sizes: 6, 10, 14 and 18 points. These range from 1/12 to 1/4 inch in height (72 points = 1 inch). The body of this manual is printed in Times Roman at the 10 point size. The section headings are printed in Times Boldface at 14 points, and the chapter headings are Boldface at 18 points.

PROFF commands exist to change both the font and point size. Specifying a non-existent point size results in the next larger size or the maximum size of 18. Fonts may be freely mixed within text, but changing point size within a line can create undesirable output, as tall letters may overprint the descending letters of the previous lines. Some specific ways of mixing point sizes are given under the topic "Mixing Point Sizes" in the Problems section.

The standard character font for printed text is established by the **.ft** *font* command. It accepts the font name or position as its argument, and will cause all text input from that point onwards, to be printed in the named font. If no font was given, then the previous font is recalled.

The easiest way to mix character fonts is when text is being formatted in *fill mode,* when words from the input line are moved into the output line until it is full, in order to eliminate white space. Changing the font for one or more input lines will result in these words being output together with the regular font words.

For example,

```
.ft roman
These lines are set in roman text, with
some words set in the
.bf
boldface font
and
.it
italics font
on the same output line.
```

will result in the output

> These lines are set in roman text, with some words set in the **boldface font** and *italics font* on the same output line.

Notice that the *boldface* and *italics* commands affected only the one input line following, and that subsequent input lines were printed using the standard font, Times Roman.

Using the Mathematics font will require close attention to the character equivalences given in Appendix A. Future work will provide more convenient means of typesetting mathematical formulae and equations.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.pt N	10	10	no	Point size. Legal sizes are 6, 10, 14 and 18. If an invalid size is given, then the next larger or maximum size is used.
.rm N|on|off	-	-	no	Roman font. Default character font at start of PROFF.
.it N|on|off	-	-	no	Italics font.
.bf N|on|off	-	-	no	Boldface font.
.ma N|on|off	-	-	no	Mathematics font.
.ft N|C	roman	prev	no	Font. Sets fonts according font name or position: Roman=1, Italics=2, Boldface=3, Math=4. User may give either the name or number. If no argument given, then the previous font will be used. Invalid fonts or numbers are ignored.

Text Format Specifications

PROFF provides several formats for the output printed text: paragraphs with smooth margins such as the body of this report, lines flushed left for headings or point form, centred lines for titles, or lines flushed to the right margin for special effects. Paragraphs are typeset in *fill mode*, in which words from the input text are moved into the output line to eliminate white space. This mode may be turned off when each line should be typeset as input, such as lists of keywords. The smooth right margins are achieved by *justifying* the text in the output line. Justification involves adjusting the spacing between words and letters so that the first character in the line is at the left margin, the last character is at the right margin, and all characters in between are proportionally spaced to give a pleasing effect.

Words like mother-in-law, which contain hyphens, may be split across lines of text to help fill the output line. PROFF provides an automatic *hyphenation mode* in which words are split under program control to more completely fill the output line. Words are split according to a built-in dictionary of prefixes, and a set of hyphenation rules. This mode may be turned off if necessary. *Discretionary hyphens* may be used in the automatic hyphenation mode and are placed at designated hyphenation points in foreign or technical words, so that the automatic hyphenation rules are abandoned for that word. If the word will not fit in the output line, it will be split at a discretionary hyphen, otherwise if the word is not to be hyphenated, the discretionary hyphen is dropped, and will not appear in the printed text. Discretionary hyphens are typed as two hyphens within a word, e.g. photo--type--setting. Sorry, but it was a horrible trick to print two hypens together. The best way for you to do it, is to turn off hyphenation so that discretionary hyphens are ignored, or to use longer dash characters.

When in fill mode, the *break* command is used to start a new line, for example at the end of a paragraph. Any input words not yet printed will be output flush with the left margin, with no justification attempted. Many PROFF commands force a break condition since they involve a change in format, for example leaving blank space, indenting or centring. However, when necessary, the break command forces a new line.

The special formats for centring and printing lines flush with the right margin do not use the fill mode of output. The subsequent input lines of text are centred or flushed right as given. The following example indicates the three ways single lines may be printed.

```
.ce
centred text
.fr
flushed right
.nf
flushed left
```

produces

centred text

flushed right

flushed left

Note that *nofill mode* was used to flush left.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.fi on|off	on	on	yes	Fill mode. Subsequent input words are made to fill the output line.
.nf	-	-	yes	Nofill mode. Subsequent input lines are printed as is, flush with left margin.
.ju on|off	on	on	no	Justify mode. Subsequent output lines will be adjusted to obtain smooth left and right margins.
.nj	-	-	no	No justify mode. Output lines will have a ragged right margin.
.hy on|off	on	on	no	Hyphenate mode. Words which overfill output line will be automatically split, if possible. Discretionary hyphens are honoured. To be compatible with ROFF, a non-zero number will be treated as on, zero will be off.
.nh	-	-	no	No hyphenate mode. Discretionary hyphens are still honoured, if present.
.br	-	-	yes	Break. Forces output line to be printed, and begins a new output line
.ce N|on|off	-	1	yes	Centre. Centre the next N lines. On is equivalent to infinity, off or N=0 stops centring.
.fr N|on|off	-	1	yes	Flush right. The next N lines are printed flush with the right margin.

Indenting

Indenting affects the margins of the printed text, without changing the line length. The left, right, or both margins may be indented. The margins may be moved either in or out, but may never be set to exceed the line length. Margins may be established at either fixed points on the line, or relative to the current margin. Thus a margin could be established at a point 5 characters from the left edge of the line by issuing the command **.in 5.** Similarily, the margin may be moved over 3 characters from the current margin by the command **.in+3.** After this command, the margin may be returned to its previous position by the command **.in-3.** Recall that you may not exceed the line length, so PROFF considers any request to move a margin beyond the ends of the line as setting no indent. The following examples demonstrate various indented formats.

```
This text will be printed without any indents
to show you where the ends of the line are.
.ib +5
Indenting both margins is useful for setting up quotations,
point form lists, or displayed text.
.ib-5
.ir 5
The indent right command us useful to set the right margin
while varying the left margin.
```

```
.in+3
Moving the margin over 3 characters from the current margin
setting is accomplished with the +N form of the command.
.in-3
Resetting the margin to the previous setting is accomplished by
using the -N form of the command.
.ib 0
This indent both margins to zero command always
clears both left and right margins.
```

will produce the output

```
This text will be printed without any indents to show you
where the ends of the line are.
        Indenting both margins is useful for setting
        up quotations, point form lists, or dis-
        played text.
The indent right command us useful to set the right
margin while varying the left margin.
      Moving the margin over 3 characters from the
      current margin setting is accomplished with
      the +N form of the command.
Resetting the margin to the previous setting is ac-
complished by using the -N form of the command.
This indent both margins to zero command always clears
both left and right margins.
```

A *temporary indent* command is provided which affects only the first line following the command. The temporary indents affect only the left margin, and may be fixed or relative, similar to the above discussion. Moving the margin in, will leave the first line indented and rest of the paragraph printed at the margin, like the paragraphs of this report, which are set using the command: **.ti+3.** Moving the margin out may be used to associate a label with a paragraph. The number of characters that the margin is moved are taken from the next input line as the label. The remainder of the characters are set at the current margin, as are all following lines of text. The following examples shows how to set point form labels on justified text:

```
.ti+5
These lines will serve to establish the ends of the lines
for the following examples.
.in+5
.ti-3
I) the margin must first be established indented from the
left margin by using an indent command.
.ti-3
M) a negative offset on the .ti command
indicates the number of characters in the label.
.ti-3
P) using this technique aligns the paragraphs,
regardless of the variable width characters.
.in-5
```

which produces this output

These lines will serve to establish the ends of the lines for the following examples.

I) the margin must first be established indented from the left margin by using an indent command.

M) a negative offset on the .ti command indicates the number of characters in the label.

P) using this technique aligns the paragraphs, regardless of the variable width characters.

Another indented format is the *hanging indent.* This format indents all but the first line of a paragraph, unlike the temporary indent which indents only the first line. The *offset* command specifies where the margin should be set on second and subsequent output lines. At the next break in formatting, this indent will be cancelled, and the previous margin reinstated. An example of hanging indents follows:

```
These lines will serve to establish where the ends of
the output lines are located.
.of +5
Justification is the means of adjusting the
output text to fit the left and
right margins.
.of +5
Hanging indents are used to fill the first line of a paragraph
which is indented after the first line.
```

produces

These lines will serve to establish where the ends of the output lines are located.

Justification is the means of adjusting the output text to fit the left and right margins.

Hanging indents are used to fill the first line of a paragraph which is indented after the first line.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.in ±Nc	0	0	yes	Indent. Adjusts the left margin
.il ±Nc	0	0	yes	Indent left.
.ir ±Nc	0	0	yes	Indent right.
.ib ±Nc	0	0	yes	Indent both. Both left and right margins are idented.
.ti ±Nc	0	0	yes	Temporary indent. First line is indented, and subsequent lines printed at current margins. Negative indents treat characters as a label and print them separately.
.of ±Nc	0	0	yes	Offset. Second and subsequent lines are indented. Indent is cancelled at the next break.

Page Control

The output page is defined by the following parameters:

Parameter	*Command*	*Initial Value*	*Maximum Value*
page length	.pl	9i	22i
line length	.ll	6i	7.5i
number of columns	.cl	1	9
space between columns (gutter space)	.gu	none	-

These defaults produce output for an 8½×11 inch page with 1 inch margins at the right, top and bottom, and a 1½" left margin. The Photon inserts page breaks (about ¾" of white space) whenever the page length is exceeded. Since PROFF has no control over where these page breaks occur, you may either accept where they are placed, or force your own by inserting *begin page* commands where appropriate, or else print the page as a continuous strip (set page length to zero). Inserting your own page breaks requires several PROFF runs, and may waste supplies, while printing one continuous strip requires cutting and pasting pieces to make up pages.

Producing multiple column output requires defining the *line length* for each column, the *number of columns* per page and the *gutter space* between the adjacent columns across the page. Text is automatically printed in columns, moving from the bottom of one column to the top of next, resetting the left margin after providing the gutter space. When beginning a new page, the text is printed at the top of the leftmost column. To begin a new column, the *begin column* command advances to the top of the next adjacent column to the right, if there are any left on this page, or else begins a new page.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.pl ±Nl	9i	9i	yes	Page length. If no units are given, then the value N is in lines. N=0 suppresses page breaks and prints text as continuous strip.
.ll ±Nc	6i	6i	no	Line length. Default units are characters.
.cl N	1	1	no	Columns. Set number of columns per page to N.
.gu Nc	-	0	no	Gutter. Set white space left between adjacent columns of text.
.bp	-	-	yes	Begin page. Start new page, at leftmost column if more than one.
.bc	-	-	yes	Begin column. Start next column to right, if any left on this page, otherwise begin page.

Line Spacing

The control of vertical line spacing is provided in two forms: automatic spacing between output lines, and extra spacing. The automatic spacing is controlled either in units of lines, or in units of points (1/72 of an inch). The *line spacing*, *single space*, and *double space* commands set the spacing in multiples of the current vertical spacing (leading). The amount of vertical spacing is specified by the *leading* command, to either close up lines of text or expand them, as in titles.

Extra space may be inserted between lines with the *space* command. It adds extra space for figures, or spacing between headings and paragraphs.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.ss	-	-	yes	Single space. Equivalent to .ls 1.
.ds	-	-	yes	Double space. Equivalent to .ls 2.
.ls N	1	1	yes	Line spacing. Set the space between lines to N times the current vertical spacing.
.ld Np	12	(ps+2)	yes	Leading. Set vertical space between lines. If no size is given, then the vertical spacing is set to
.sp Nl	-	1	yes	Space. Insert N lines of white Space.

Command Control

PROFF takes as formatting commands, lines of input text which contain a command character in the first position. Whenever it is necessary to print lines which begin with this special character, such as the examples in this manual, there are two methods of avoiding PROFF interpreting them as command lines. The *literal* command takes the specified number of lines as text and prevents PROFF from checking for command lines. However, if there are many occasions when input lines should be taken literally, the *change command character* command allows the substitution of some other command character. From that point on in the file, PROFF commands must have this new command character as the first character in the line, including the next change command character command, otherwise they will be treated as lines of input text.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.li N	-	1	no	Literal. Takes the next N input lines without looking for commands.
.cc C	.	.	no	Change command character. All subsequent commands, including next change command character command must begin with the character C, instead of a dot.

Ligatures

Ligatures are the special print characters formed by overlapping letters, usually sequences beginning with f or t. Our typesetter is equipped with three ligatures fi, fl, ff on each of our three text fonts: Roman, Italics and Boldface. PROFF has an automatic mode in which lower case sequences of f followed by i, l, or f, are detected and converted to the special ligature characters.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.lg on|off	on	on	no	Ligature. When on, and type font is roman, italics or boldface, ligatures are inserted whenever appropriate.

File Control

It is often the case that large documents are kept in separate, smaller files, in order to make editing more convenient. The PROFF *switch source files* command allows the user to specify the insertion of the lines in a file at this point. Lines of text are read from this new file until the end of file, then lines are read from the original file, continuing from where the switch file command was located. Up to five levels of file switching are permitted. Any number of switch commands may be contained in any one file. PROFF commands will be interpreted both between switch file commands and within the named files. A typical use of this command is as follows

```
.so /title
.bp
.so /intro
.bp
.so /report
```

which formats a three part report with title, introduction and body edited in three files.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.so file	-	-	no	Switch source file. PROFF begins reading the file filename. Lines of text and formatting commands are recognized until end of file, when the previous file is continued from the line after the **.so** command.

Typesetting Programs

Computer programs are normally printed on computer terminals or line printers, which have fixed width characters (all widths are the same). On a phototypesetter, the characters were designed to look best with variable spacing, and hence, printing them as fixed width characters would make them look awkward. Furthermore, attempts to format programs by filling lines and justifying text would result in chaos. Therefore we have tried to compromise, and approximate fixed width character printing for computer programs, while avoiding this awkwardness.

PROFF has a program setting mode, which establishes various formatting mode settings automatically:

```
Fill mode:      off
Justify mode:   off
Hyphenation:    off
Ligatures:      off
```

In addition, program mode converts all spaces to fixed width spaces which are equal to the width of a digit. Fortunately, all digits have the same widths in our three text fonts, Roman, Italics, and Boldface. This allows columns of numbers to align properly, such as line numbers or statement numbers on programs. The following is a FORTRAN program typeset in the program mode.

```
C.....TEST FMIN
      FUNCTION F(X)
      DOUBLE PRECISION F,X
      F = X*(X*X - 2.) - 5.
      RETURN
      END
      EXTERNAL F
      DOUBLE PRECISION A,B,Z,TOL,FMIN
      A = 0.
      B = 1.
      TOL = 1.0E-5
      Z = FMIN(A,B,F,TOL)
      WRITE(6,1) Z
    1 FORMAT(3H Z=,F12.5)
      STOP
      END
```

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.pm on|off	off	on	yes	Program mode. Adjust formatting parameters to print computer programs.

Typesetting with TTS codes

PROFF generally will format documents without the user resorting to the low level typsetter codes (TTS codes). However, some special effects, which are possible through the Photon typesetting software, have not been made available in PROFF as PROFF commands. For users that wish to make use of these TTS codes, PROFF provides two escape mechanisms: a typeset literally command **.tl** and a TTS escape mechanism (the at-sign, @). (Normal PROFF users should skip the remainder of this section.)

The at-sign escape mechanism allows the insertion of TTS codes within a line of text. Characters are normally treated as ASCII text, and are translated into TTS equivalents, complete with case shifts, and formatting codes. However, when an at-sign is detected, the following characters up until the next at-sign will be considered as TTS codes directly, and no conversion will be attempted. For example, the registered trademark symbol is accessed by the TTS code /i and the PROFF text to generate this after some name, would be

PHOTON ECONOSETTER@/i@ Typesetter

which would print as

PHOTON ECONOSETTER® Typesetter

If many TTS codes are to be generated, the *typeset literally* command will take several input lines as TTS code directly, until a line beginning with the PROFF command character is detected. Thus the following input will pass on the TTS codes between the **.tl** and **.en** commands:

```
.tl
<p10<t01<l3000
.en
```

(which sets point size 10, typeface 1, and line length of 30 pica or 5 inches).

These PROFF features are made available for debugging and special effects, as a convenient way of controlling the typesetter directly by TTS codes. No attempt to use these codes should be made without careful study of the Photon applications book, and the TTS codes summary in Appendix C.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.tl	-	-	no	Typeset literally. Takes the following lines (until a **.en** command) as TTS codes.
.en	-	-	no	End. Terminates typeset literal input.

Tab Stops and Tables

PROFF can print text aligned to fixed postions within an output line, these points being called *tab stops*. On input, the text to be printed at a tab stop is delimited by a *tab character*, which is normally the ASCII TAB character (octal 011), but may be set to any character chosen by the user. Text is normally printed left aligned at the tab stop, but may also be right aligned or centred on the tab stop.

Tables of columnar information may easily be printed by establishing the appropriate positions and alignment for each column. Note that you only specify where the column is to be placed, and not how wide it will be. Since each tab character in the input text causes the following text (until the next tab character), to be aligned at the next tab position, any text which precedes the first tab character in the line will be printed at the left edge of the line, in a default column zero. The following example shows the use of column zero, as well as the various alignments supported.

```
.ta 10l 30c 40r
.tc #
Date#April 1#small#$1.00
Time#12:00 noon#medium#$10.00
Purpose#April Fool's#giant economy#$100.00
```

will produce

Date	April 1	small	$1.00
Time	12:00 noon	medium	$10.00
Purpose	April Fool's	giant economy	$100.00

A useful feature of tabs in PROFF exists when tables are set in *fill mode*. In this case, each table line (one containing a tab character) will always start a new print line. But subsequent input lines without tab characters will be filled and justified in the last column in the table. This permits the last column to contain a paragraph such as the command descriptions in this document. The last tab stop will remain in effect until the next *break*, caused either by a table line containing a tab character, or a formatting command that causes a break.

As an illustration of how tables are defined and printed, here are the complete set of PROFF commands required to print the command description table for this section. Note that for this example, the tab character was changed to a # character, so that it could be both printed in the example and act as a tab character to cause table formatting.

```
.tc #
.sp 2
.rule
.ta 2l 15c 21c 27c 32l
.it 2
##Initial#Default#Cause
#Command#Value#Value#Break#Description
.ta 3l 15c 21c 27c 32l
.rule
.sp
#.ta Nt Nt ...#-#-#no#Tab stop.
Sets tab stops at character postions specified.
An alignment t may optionally be coded: l=left justify,
c=centre on tab stop, r=right justify to tab stop.
If no alignment is given, left justify is assumed.
#.tc C#TAB#TAB#no#Tab character.
Sets tab character to character C.
User set tab character is in addition to the
ASCII TAB character (octal 011), which is always effective
in causing table formatting.
.rule
```

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.ta Nt Nt ...	-	-	no	Tab stop. Sets tab stops at character postions specified. An alignment t may optionally be coded: l=left justify, c=centre on tab stop, r=right justify to tab stop. If no alignment is given, left justify is assumed.
.tc C	TAB	TAB	no	Tab character. Sets tab character to character C. User set tab character is always in addition to the ASCII TAB character (octal 011), which is always effective in causing table formatting.

Text Registers

PROFF provides a means for holding frequently used text, so that it may be inserted repeatedly into the printed output without retyping. A *text register* is a named storage area within PROFF. The text register name is usually made up of letters and digits, and enclosed in parentheses. Thus (uw) could be the name of a text register containing the string "University of Waterloo". If the name is only a single character, then the enclosing parentheses can be omitted.

Text is entered into a text register by surrounding the input lines by *assign text* and *end* commands. For example,

```
.at (uw)
University of Waterloo
.en (uw)
```

would establish the text register (uw) with the string "University of Waterloo".

Text registers are used by *inserting* them into your printed output. To do this, you must define an *insertion character;* there is no default character provided.

```
.at (uw)
University of Waterloo
.en (uw)
.ic ↑
This line, written at ↑(uw), will contain
several references to the string "↑(uw)".
```

will print as

```
This line, written at University of Waterloo, will contain
several references to the string "University of Waterloo".
```

Notice that inserting a text register is identical to typing its contents at that point; no extra characters (blanks) are added. Furthermore, in order to have the current insertion character print, and not act as indicating a text register, you must type two insertion characters, using the first to neutralize the second. Thus typing "↑↑" would print as "↑".

Text registers that contain more than a single input line may produce unpleasant results when inserted into the middle of a line. (They work just fine if they are at the beginning of an input line.) The new line characters between lines in a text register are preserved, thus inserting it causes the line to be split into several lines. Therefore you cannot use the following text register to make a phrase print in boldface:

```
.at (boldly)
.bf
phrase
.en (boldly)
.ic ↑
The phrase, ↑(boldly), will not be handled
as might be expected.
```

which generates

```
The phrase, .bf phrase, will not be handled as might be expected.
```

The first line of the text register, ".bf", was appended to the line containing the insertion reference, and began a new input line with the text "phrase". The ".bf" string was not at the beginning of a new line after the insertion was complete, therefore PROFF did not notice it as a boldface command.

Combining text registers and TTS codes is a convenient way to print special characters. For example, the copyright symbol is printed by the TTS codes, /j. Thus a text register named (cpr) could be defined and used as follows:

```
.at (cpr)
@/j@
.en (cpr)
.ic ↑
.ce
↑(cpr)University of Waterloo, 1976
```

which produces

©University of Waterloo, 1976

More complicated TTS code sequences allow us to build up special characters which do not exist on the Photon disk. For example, the following text register defines a 10 point umlaut accent.

```
.at (umlaut)
@/w011<0<m9.<m2.<a011@
.en (umlaut)
Bjo↑(umlaut)rck contains an umlaut over the 'o'.
```

which prints as

Björck contains an umlaut over the 'o'.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.at reg	–	–	no	Assign text. Subsequent input lines will be accumulated into the register named reg, until a matching **.en** command with the same register name is encountered. Any previous contents of reg are lost. Reg is a name composed of letters and digits enclosed in parentheses, unless the name is only one character, when the parentheses can be omitted.
.en reg	–	–	no	End register. Terminates a matching assign text command.
.ic C	–	–	no	Insertion character. Define the character C as the insertion character for text registers. When encountered, PROFF expects a register name to follow (either a single character name, or a name in parentheses).
.un reg	–	–	no	Undefine register. The register named reg is emptied, space made available, and the name reg removed from the list of text registers.

Number Registers

An analagous feature to text registers is provided by PROFF for the handling of numbers. A *number register* is identified by a name, enclosed in parentheses unless it only one character long, and contains an integer value. The *assign number* command is used to initially set the value of the text register, and can compute an expression formed by constants, inserted number registers, and the operators for add, subtract, multiply and divide. If the expression on the assign number command begins with an operator, such as add, then it is assumed to operate on the current value of the number register.

The *assign format* command indicates the output representation of the number register, whenever it is inserted into the printed text. The default format for all number registers is as a simple decimal number, without leading blanks or zeros. Optionally, Roman Numerals (i, ii, iii, iv,...) or Theatre Numerals (a, b, c, ..., z, aa, bb,...) can be selected. Either upper or lower case numerals can be selected. A minimum width for decimal numbers can be established to force leading zeros to print when the number inserted is smaller than the field width selected. The following example shows the use of number registers, the operations on them, their output formats, and the use of single character register names.

```
.nf
.an p 0
.af p a
.ic ↑
This point form table will show the
use of theatre numerals:
.an p +1
↑p. First line.
.an p +1
↑p. Second line.
.an p +1
↑p. Third line.
.an p +1
↑p. Fourth Line.
.af p 001
There were ↑p lines in this example.
.an (year) 1976
.af (year) I
The date in roman numerals is ↑(year).
```

will produce

```
This point form table will show the
use of theatre numerals:
a. First line.
b. Second line.
c. Third line.
d. Fourth Line.
There were 004 lines in this example.
The date in roman numerals is MCMLXXVI.
```

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.an reg expr	-	-	no	Assign number. The expression expr is evaluated and the result assigned to the number register reg. Operators can be +, -, /, *. If expr starts with an operator, then it is assumed to operate on the current value of the register.
.af reg f	-	-	no	Assign format. The number register reg will print with format f when inserted. F can be i or I for roman numerals, a or A for theatre numerals, and 01, 001, 0001,... for decimal numbers with leading zeros to fill the specified width. Default format is decimal numbers with no leading zeros.

Macros

Text registers can be used to define *macro* commands, which allow the user to extend the command set of PROFF. A macro is defined by the **.at** and **.en** commands enclosing the operations to be performed by the macro. The text register name is then used as the macro name, and the macro is invoked by issuing the command: **.reg** where reg was the name of the register. Note that if the command character is redefined, then the macros must also use this new command character.

Macros may be supplied with up to 9 parameters. Within the macro definition, the user must code parameter references using a *parameter character*, and the parameter number, e.g. $1, $2, ..., $9 if the parameter character was a dollar sign. Parameters to a macro are typed on the same line as the macro command, separated by blanks. Parameter $1 will be replaced by the text typed as the first parameter, $2 the second, and so on. Missing parameters will be replaced by empty strings. If a parameter must contain a blank, then it must be enclosed in *quote characters*, the default for which is the double quote (").

Macros have been used throughout this manual to simplify the inputting of text, and to standardize the output format. Macros were defined for chapter headings, section headings, starting paragraphs and for tables. Appendix E contains a description of the macros defined for Computer Science Research Reports.

The following illustration defines a macro to automatically format sequence-numbered paragraphs. The paragraph heading will be sequentially numbered, and printed in boldface type.

```
.ic ↑
.pc $
.an (para#) 0
.at (newpara)
.sp
.an (para#) +1
.bf
↑↑(para#). $1
.en (newpara)
.newpara "First Paragraph."
Note that only the paragraph heading is set in boldface type,
the remainder of the paragraph is in roman.
.newpara "Second Paragraph."
Each paragraph is easily formatted by the macro command,
which leaves space between paragraphs, increments the
paragraph number register, (para#), sets the heading in
boldface type, and constructs the heading from the
number register (para#), and the first parameter.
```

which produces this output

1. First Paragraph. Note that only the paragraph heading is set in boldface type, the remainder of the paragraph is in roman.

2. Second Paragraph. Each paragraph is easily formatted by the macro command, which leaves space between paragraphs, increments the paragraph number register, (para#), sets the heading in boldface type, and constructs the heading from the number register (para#), and the first parameter.

Note that the macro required the insertion of the paragraph number register, (para#), to be typed with two insertion characters. This was to avoid having PROFF insert the current value of the text register (just assigned to zero) when the macro was created, but instead to insert the value when the macro is used.

The use of parameters may also be utilized with text registers when inserted from within a text line. Parameters are typed within the parentheses surrounding the name, and are separated by blanks. Note that this requires that the parentheses be used, even if the text register has a single character name. However, the user is again cautioned about the unpleasant effects which may result from invoking a multiple line text register from the middle of text lines. This example shows the use of TTS codes to implement superscripts and subscripts.

```
.ic ↑
.pc $
.at (sup)
@/w006@$1@<a006@
.en (sup)
.at (sub)
@<a006@$1@/w006@
.en (sub)
This line contains super↑(sup scripts) and sub↑(sub scripts).
Math could be typed as X↑(sup 2) and A↑(sub i).
```

which would print as

This line contains super$^{\text{scripts}}$ and sub$_{\text{scripts}}$.
Math could be typed as X^2 and A_i.

Note that each text register contains only a single line, and that the parameter, $1, was substituted outside of the area treated as TTS codes (most importantly to avoid illegal TTS codes and to permit upper/lower case shifts.).

Conditional execution of commands within a PROFF macro is possible through the use of the *if* and *ignore* commands. These commands specify a register name, and skip lines within the macro until a matching **.en** command is encountered. The register name is not used to store any text, but only to match a subsequent end command. The ignore command unconditionally skips lines; in effect it is a goto command. The if command tests an expression and does not skip lines if the expression evaluates to a non-zero value. Thus you might consider the if command and its matching end command as the brackets around statements which will be executed only when the expression is non-zero.

The following example shows a macro for handling chapter headings. The if command is used to skip to the top of a new page, only if it is not the first chapter.

```
.an (chap#) 0
.at (chap)
.if (notfirst) ↑↑(chap#)
.bp
.en (notfirst)
.an (chap#) +1
.bf
↑↑(chap#). $1
.en (chap)
```

Initially, the number register (chap#) will be zero. Thus the **.chap** macro will test this by the if command, and since the expression is zero, lines will be skipped until a matching **.en (notfirst)** command is encountered, thereby avoiding the begin page command. Since the chapter number is incremented within this macro, the next, and all subsequent calls will evaluate the if

expression non-zero, thus executing the begin page command. Note the use of two insertion characters to delay the insertion of the number registers until the macro was invoked, rather than when it was created.

Another example will demonstrate the use of the if and ignore commands together. The macro will be used to generate entries in a table. Each entry is separated from the previous one by a blank line, and every fifth entry is separated by several dashes to improve readability. The macro will count the number of times it has been called, and if it is the fifth time, it will print a line of hyphens, otherwise, it will space one line.

```
.an (#entries) 0
.an (fifth) 5
.at (entry)
.if (notfifth) 5-↑↑(fifth)
.sp
.ig (endif)
.en (notfifth)
----------
.an (fifth) 0
.en (endif)
.an (#entries) +1
.an (fifth) +1
... remainder of macro would generate table entry
.en (entry)
```

The number register (#entries) counts the total number of calls to the **.entry** macro, and the (fifth) register counts the number of calls since the last time dashes were output. The if expression tests whether (fifth) has reached 5 yet. Initially, (fifth) is 5, and the if expression will be zero, thus the statements until the **.en (notfifth)** will be skipped, since the first call is considered a fifth call. The dashes are printed, and the **.en (endif)** command passed over, since no condition (endif) was in effect. The value of (fifth) is reset to 0 to count to the next fifth entry. Upon the next entry, the expression will be non-zero, since (fifth) is not yet 5, therefore the **.sp** and **.ig (endif)** commands are executed, not skipped. The ignore command forces the macro to skip the dashes and thereby producing the desired output for a non-fifth table entry.

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.pc C	-	-	no	Parameter character. Macro parameters within text registers will be recognized as C1, C2,..., C9. When the macro is called, parameters on the command line separated by blanks will replace occurences of these references.
.qc C	"	"	no	Quote character. Parameters to macros that must contain a blank can be surrounded by the quote character.
.reg p1 ... p9	-	-	?	Macro invocation. The text register reg will be executed with parameters p1,...,p9. The parameters are separated by blanks, and if they are to contain any blanks, they must be enclosed in quote characters.

PROFF TSS Commands

All TSS commands for PROFF are stored under userid PHOTON, which has general read permission on the command files. There are three major commands:

PROFF	Format text file and convert to TTS codes
PASS	Transfer TTS file to typesetter via communications line.
L	List TTS codes (mainly for debugging PROFF itself, and typeset literally commands).

The command syntax for the PROFF command is

PHOTON/PROFF input-file proff-file

If either filename is missing, then it will be requested from the user at the terminal. The input file contains the formatting commands and text; the output file will contain the TTS codes needed by the phototypsetter. Illegal commands that are not recognized by PROFF will be listed on your timesharing terminal.

For users who are working at the TTS level, or who are debugging PROFF, there are options which are recognized (normal users should skip this paragraph):

PHOTON/PROFF [-DUIPT] [input-file] [proff-file]

where the input and output files are now optional, and the options are

- -D Debug mode. Output ASCII form of TTS codes, identical to the PHOTON/L output
- -U Uppercase terminal. Output terminal is upper case only; lower case TTS letters are displayed as letters, and upper case TTS codes are displayed with a leading % sign, eg. R is the lower case letter r, but %R is a TTS return code.
- -I Input from upper case terminal. The % escapes are honoured for upper case only terminals.
- -P Paginate output for CRT. After 20 lines of output the user must press carriage return to continue. This stops the scrolling action of CRTs at convenient times.
- -T Terminal mode. Missing filenames on the command line are assumed to be the terminal. This allows for the convenient input of PROFF commands from the terminal, or output of TTS codes at the terminal for debugging.

When the formatted text is to be typeset, the PROFF output file, containing the TTS codes, must be transferred to the phototypesetter. With the communications interface the following command is required:

PHOTON/PASS proff-file

where the proff-file is the same one generated by PROFF. Prior to transferring any TTS codes, this message will appear on the TSS terminal attached to the phototypesetter:

Press Reset and Restart

This indicates that the phototypesetter must have these switches pressed, in that order, to begin typesetting. TTS codes will then be transmitted to the typesetter without any apparent activity on the terminal. This is because the phototypesetter's communication interface does not send the TTS codes o to the terminal, to avoid any strange behaviour by the terminal acting on the TTS codes. In normal operation, the phototypesetter will stop (the STOP light will go on), and the terminal will respond, at about the same time. If not, the BREAK (or INT or ATTN) key on the terminal will clear the interface, stop the typesetter, and allow the terminal to be used with TSS again. If the typesetter stops prematurely, then the BREAK key will reactivate the terminal, and action described in the typesetter operation instructions should be followed (usually, "call for help!").

If you are curious, or if you are working at the typesetting code level, then the L command will display the TTS codes generated by PROFF on your timesharing terminal:

```
PHOTON/L  proff-file
```

These TTS codes are summarized in Appendix C. If you desire the output to be directed to a file, use the standard redirection of terminal output:

```
PHOTON/L  proff-file  >output-file
```

which will place the ASCII interpretation of the TTS codes into output-file.

Using ROFF for Proofreading

Since PROFF accepts many of the commands used by ROFF, and since ROFF has macro facilities to define new commands, it is possible to format PROFF documents using ROFF, and quickly obtain line printer or terminal output for proofreading. The special features of the phototypesetter have been handled by commands which are different from ROFF commands, e.g. .pt, .bf, etc, which have no meaning in ROFF. Hence, ROFF macros for these commands have been built, and are stored in the file PHOTON/ROFF. The variable line spacing, and different character fonts are not possible to duplicate, but the general format of paragraphs, headings, and tables are. Thus the following ROFF file will format your PROFF document:

```
.so PHOTON/ROFF
.so your-PROFF-file
```

The first file will define the macros to handle the PROFF unique commands, and the second file is your PROFF unformatted input file.

Appendix A
Photon Character Set

TTS Char	Roman			Italics			Boldface			Mathematics		
	D	U	/	D	U	/	D	U	/	D	U	/
1	1	¼	fi	*1*	*¼*	*fi*	**1**	**¼**	**fi**	→	⌠	£
2	2	½	fl	*2*	*½*	*fl*	**2**	**½**	**fl**	←	⌡	..
3	3	¾	ff	*3*	*¾*	*ff*	**3**	**¾**	**ff**	↓	⎨	═
4	4	%	°	*4*	*%*	°	**4**	**%**	°	↑	⎩	°
5	5	–	÷	*5*	*–*	÷	**5**	**–**	÷	√	⎫	÷
6	6	—	×	*6*	*—*	×	**6**	**—**	×	‾	⎬	×
7	7	-	−	*7*	*-*	−	**7**	**-**	−	≥	⎭	″
8	8	—	¶	*8*	*—*	¶	**8**	**—**	¶	≤	⎧	′
9	9	&	§	*9*	*&*	*§*	**9**	**&**	**§**	≡	⎩	^
0	0	?	O	*0*	*?*	□	**0**	**?**	□	≈	⎭	˜
a	a	A		*a*	*A*		**a**	**A**		κ	Ω	
b	b	B		*b*	*B*		**b**	**B**		ψ	‿	
c	c	C		*c*	*C*		**c**	**C**		φ	¢	
d	d	D		*d*	*D*		**d**	**D**		μ	¬	
e	e	E		*e*	*E*		**e**	**E**		γ	Θ	
f	f	F		*f*	*F*		**f**	**F**		ν	∫	
g	g	G		*g*	*G*		**g**	**G**		ο	∅	
h	h	H		*h*	*H*		**h**	**H**		π	∈	
i	i	I	®	*i*	*I*	®	**i**	**I**	®	η	ϒ	∴
j	j	J	©	*j*	*J*	©	**j**	**J**	©	ρ	∉	∞
k	k	K	@	*k*	*K*	@	**k**	**K**	@	σ	◯	∂
l	l	L		*l*	*L*		**l**	**L**		ς	℘	
m	m	M		*m*	*M*		**m**	**M**		‚	′	
n	n	N		*n*	*N*		**n**	**N**		ω	—	
o	o	O	¢	*o*	*O*	¢	**o**	**O**	¢	θ	Φ	∝
p	p	P	†	*p*	*P*	‡	**p**	**P**	†	ι	Ψ	~
q	q	Q		*q*	*Q*		**q**	**Q**		α	Γ	
r	r	R		*r*	*R*		**r**	**R**		δ	Λ	
s	s	S		*s*	*S*		**s**	**S**		λ	∇	
t	t	T		*t*	*T*		**t**	**T**		ε	Ξ	
u	u	U		*u*	*U*		**u**	**U**		ξ	Σ	
v	v	V		*v*	*V*		**v**	**V**		χ	‾	
w	w	W		*w*	*W*		**w**	**W**		β	Δ	
x	x	X		*x*	*X*		**x**	**X**		υ	>	
y	y	Y		*y*	*Y*		**y**	**Y**		ζ	Π	
z	z	Z		*z*	*Z*		**z**	**Z**		τ	<	
$	$	!		*$*	*!*		**$**	**!**		⊤	⊥	
(	)	(		*)*	*(*		)	(		~	≠	
)	]	[		*]*	*[*		]	[		}	{	
,	,	'		,	'		,	'		∪	∩	
-	-	+		-	+		-	+		≃	⌋	
.	.	′		.	′		.	′		⊂	⊃	
;	;	:	=	;	:	±	;	:	=	⌈	⌉	±
>	#	*		#	*		#	*		⊆	⊇	
_	′	″		′	″		′	″		‾	‾	
A	—	|		—	|		—	|		|	
/			/			/			/			/
< /	\			\			\			\		

Appendix B
PROFF Command Summary

Command	*Initial Value*	*Default Value*	*Cause Break*	*Description*
.af reg f	–	–	no	Assign format.
.an reg expr	–	–	no	Assign number.
.at reg	–	–	no	Assign text.
.bc	–	–	yes	Begin column.
.bf N|on|off	–	–	no	Boldface font.
.bp	–	–	yes	Begin page.
.br	–	–	yes	Break.
.cc C	.	.	no	Change command character.
.ce N|on|off	–	1	yes	Centre.
.cl N	1	1	no	Columns.
.ds	–	–	yes	Double space.
.en	–	–	no	End.
.en reg	–	–	no	End register.
.fi on|off	on	on	yes	Fill mode.
.fr N|on|off	–	1	yes	Flush right.
.ft N|C	roman	prev	no	Font.
.gu Nc	–	0	no	Gutter.
.hy on|off	on	on	no	Hyphenate mode.
.ib ±Nc	0	0	yes	Indent both.
.ic C	–	–	no	Insertion character.
.il ±Nc	0	0	yes	Indent left.
.in ±Nc	0	0	yes	Indent.
.ir ±Nc	0	0	yes	Indent right.
.it N|on|off	–	–	no	Italics font.
.ju on|off	on	on	no	Justify mode.
.ld Np	12	(ps+2)	yes	Leading.
.lg on|off	on	on	no	Ligature.
.li N	–	1	no	Literal.
.ll ±Nc	6i	6i	no	Line length.
.ls N	1	1	yes	Line spacing.
.ma N|on|off	–	–	no	Mathematics font.
.nf	–	–	yes	Nofill mode.
.nh	–	–	no	No hyphenate mode.
.nj	–	–	no	No justify mode.
.of ±Nc	0	0	yes	Offset.
.pc C	–	–	no	Parameter character.
.pl ±Nl	9i	9i	yes	Page length.
.pm on|off	off	on	yes	Program mode.
.pt N	10	10	no	Point size.
.qc C	"	"	no	Quote character.
.reg p1 ... p9	–	–	?	Macro invocation.
.rm N|on|off	–	–	no	Roman font.
.so file	–	–	no	Switch source file.
.sp Nl	–	1	yes	Space.
.ss	–	–	yes	Single space.
.ta Nt Nt ...	–	–	no	Tab stop.
.tc C	TAB	TAB	no	Tab character.
.ti ±Nc	0	0	yes	Temporary indent.
.tl	–	–	no	Typeset literally.
.un reg	–	–	no	Undefine register.

Appendix C
Photon TTS Formatting Codes

Function	*Argument*	*Description*
<a□□□	halfpoints	Add Lead
<b		No Flash next char
<c□□	00 - 39	Call Format
<d□	leader char	Define Leader
<e□□	line count or 00	End Runaround
<f		Kill Word
<g□□	format	Get Stored Line
<h		Set Hyphenation Mode
<i		Insert Leader
<j□	rule char	Set Vertical Rule
<k		Kill Line (normal mode)
		Kill Word (hyph mode)
<l□□□□	pica/points	Line Length
<m□	halfunits	Mortice
<n		Set Normal Mode (unjustified)
<o□□	format	Store Line
<p□□	points	Point Size
<q		Quad Right
<r□□	line count	Start Runaround
<s□□	format	Store Format
<t□□	typeface	Typeface
<u		Allow Flash
<v□□□	halfpoints	Leading
<w□□	halfunits	Leader Width
<x		Cancel Indents
<y		Reverse Amount of Leading Count
<z		Tab
<0		Zero Width
<8		Set Lens Table 1
<9		Set Lens Table 2
<-		Zero Lead
<.		Cancel Flash
<F		Stop Code
< □□	halfunits	Change Set
<,		Divide Leader
</		Reverse Slant \
/a		Force Carriage Return (force justify)
/b□□□□	pica/points	Indent Both
/c		Quad Centre
/e□□□□	pica/points	Indent Left
/f□□□□	pica/points	Indent Right
/g		Indent Text
/h		Set Hyphenless Mode
/i		Registered Trademark ®
/j		Copyright ©
/k		At-sign @
/l		Quad Left

Function	*Argument*	*Description*
/m		Em Space
/n		En Space
/o		Cents ¢
/p		Dagger †
/q□	count	Set Number of Columns
/r		Quad Right
/s□□□□	pica/points	Set Gutter Space
/t		Thin Space
/u		Unit Space
/v□□□□		Select Variable Flash
/w□□□	halfpoints	Reverse Lead Immediate
/x		Reset Leading Counter
/y□□□□	halfpoints	Set Column Depth
/z		Lead to End of Column
/1		fi Ligature
/2		fl Ligature
/3		ff Ligature
/4		Degree Symbol °
/5		Divide Symbol ÷
/6		Multiply Symbol ×
/7		Minus Symbol −
/8		Paragraph Mark ¶
/9		Section Mark §
/0		Bullet or Box O
/-		Baseline Rule |
//		Fraction Bar /
/F		Stop Code
/;		Equal Sign =

Appendix D
ASCII-TTS Code Conversion Table

The following table is used by PROFF for generating the actual TTS codes placed in the output file. Normally, all ASCII characters are converted to equivalent Photon characters, shown in Appendix A. Thus PROFF prints ASCII braces by changing typefonts to the Mathematics font, and back again for regular text.

When the user is using TTS Codes, via .tl or @, PROFF uses this table to generate the binary codes. ASCII codes which are not in the table cause error messages.

Octal Code	*ASCII Code*	*TTS Code*	*Octal Code*	*ASCII Code*	*TTS Code*
00	F	Tape Feed	40	t	Letter t
01	T	Thin Space	41	5	Digit 5
02	e	Letter e	42	z	Letter z
03	3	Digit 3	43	(	Parenthesis
04	/	Fraction Bar	44	l	Letter l
05	[	Lower Matrix	45	_	Vertical Rule
06	a	Letter a	46	w	Letter w
07	$	Dollar Sign	47	2	Digit 2
10		Space Band	50	h	Letter h
11	A	Add Thin Space	51	>	Em Leader
12	s	Letter s	52	y	Letter y
13	M	Em Space	53	6	Digit 6
14	i	Letter i	54	p	Letter p
15	8	Digit 8	55	0	Digit 0
16	u	Letter u	56	q	Letter q
17	7	Digit 7	57	)	En Leader
20	R	Return	60	o	Letter o
21	'	Apostrophe	61	9	Digit 9
22	d	Letter d	62	b	Letter b
23	-	Hyphen	63	↑	Upper Rail
24	r	Letter r	64	g	Letter g
25	4	Digit 4	65	;	Semicolon
26	j	Letter j	66	U	Upshift
27	<	Bell Code	67	=	Lower Rail
30	n	Letter n	70	m	Letter m
31	,	Comma	71	.	Period
32	f	Letter f	72	x	Letter x
33	L	Quad Left	73	1	Digit 1
34	c	Letter c	74	v	Letter v
35	N	En Space	75	C	Quad Center
36	k	Letter k	76	D	Downshift
37	]	Upper Matrix	77	?	Ignore

Appendix E
Computer Science Research Report Macro File

PHOTON/CSRR

```
.cc.  (comment line, since change command character to dot
.cc.    if a no-operation command)
.cc.
.cc.    Follow .title macro with the name of the report.
.cc.    Each line of title is printed boldface, 14 point
.cc.       and centred.
.cc.
.at (title)
.ll 35
.pt 14
.ld 16p
.ft bold
.ce on
.en (title)
.cc.
.cc.    Follow .author macro with the author(s) name(s).
.cc.    Each author is printed in italics, 10 point, centred.
.cc.
.at (author)
.pt 10
.ld 12p
.sp
.ft italics
.en (author)
.cc.    Provide report number as parameter, e.g. .report 76-08
.cc.    The department, university, and report lines are generated.
.cc.
.at (report)
.pc $
.sp
.ft roman
Department of Computer Science
University of Waterloo
.sp
Research Report CS-$1
.en (report)
.cc.
.cc.    This macro is the last for the title page.
.cc.    Follow it by the date of the report, e.g. April 1976
.cc.
.at (date)
.sp
.en (date)
```

```
.cc.
.cc.    2 column layout for 8 1/2 by 11 inch page
.cc.
.at (layout2)
.ce off
.cl 2
.gu 4
.ll 30
.pl 90c
.en (layout2)
.cc.
.cc.    Single column layout for 8 1/2 by 11 inch page
.cc.
.at (layout1)
.ce off
.ll 55
.pl 90c
.en (layout1)
.cc.
.cc.    Perpetual column, for cutting and pasting pages
.cc.
.at (layout0)
.ce off
.ll 55
.pl 0
.en (layout0)
.cc.
.cc.    Follow .chapter with chapter names. Each line is
.cc.    printed boldface, 14 point, flush left.
.cc.
.at (chapter)
.nf
.in 0
.ft bold
.pt 14
.ld 16p
.en (chapter)
.cc.
.cc.    Follow .section with section name. Each line is
.cc.    printed boldface, 10 point, flush left.
.cc.
.at (section)
.nf
.in 0
.pt 10
.ld 12p
.sp 2
.bf
.en (section)
```

```
.cc.
.cc.    You must specify .body after a .chapter or
.cc.    .section and before you start text printing.
.cc.    Prints text 10 point, roman, filled and justified.
.cc.
.at (body)
.fi
.pt 10
.ld 12p
.sp
.ft roman
.en (body)
.cc.
.cc.    Separate paragraphs with .para. Paragraphs are
.cc.    not indented but are spaced 1 line apart.
.cc.    The first paragraph after a .body doesnt need a .para.
.cc.
.at (para)
.sp
.en (para)
.cc.
.cc.    These 3 registers are for changing typefont within lines.
.cc.    Specify ↑r, or ↑b, or ↑i where font change should occur.
.cc.    to roman, boldface, italics respectively.
.cc.
.at r
@<t01@
.en r
.at i
@<t02@
.en i
.at b
@<t03@
.en b
```