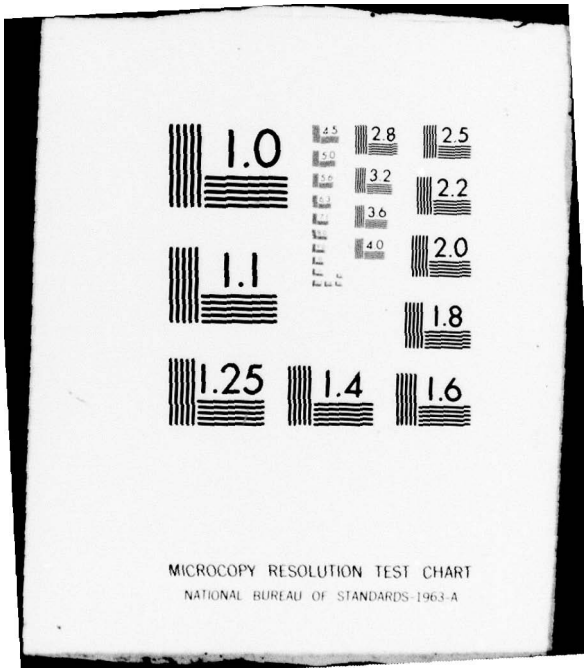
AD-A074 399 AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA) F/G 5/2
CICERO TYPESETTING MANUAL,(U)
JUL 79 R C ADAMS, G A CLEAVE
UNCLASSIFIED ARL/STRUC NOTE-453 NL

1 OF 2
AD A074399


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(14) ARL/STRUC/NOTE-453

AR-001-743

(12) B.S.

LEVEL II

AD A074399

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

MELBOURNE, VICTORIA

STRUCTURES NOTE 453

(6) CICERO TYPESETTING MANUAL

DDC RECEIVED SEP 27 1979 E

by

(10) R.C. ADAMS and G.A. CLEAVE

DDC FILE COPY

Approved for Public Release.

(12) 130p.

© COMMONWEALTH OF AUSTRALIA 1979

COPY No 21

008 650

(11) JULY 1979

79 09 26 059

UNCLASSIFIED

AR-001-743

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

STRUCTURES NOTE 453

CICERO TYPESETTING MANUAL

R.C. Adams and C.A. Cleave

(C) Copyright Commonwealth of Australia, 1979

SUMMARY

CICERO is a scheme for coding typesetting information into natural language text. Very powerful text processors may be designed to produce fully paginated typescript or photoset copy from text files containing a free format source text and CICERO format coding. Such files may be edited very easily and subsequently reformatted automatically. The separation of the editing function from formatting is one of the principal advantages of CICERO. CICERO provides means to control justification, hyphenation, pagination and tabulation, the location of figures and indexes, the setting of multi-line mathematical expressions, footnotes, sidenotes and the contents of indexes.

The manual is mainly a users guide to the coding scheme. Several computer programs are available to process files with CICERO coding. All these programs are written in PASCAL and run on a DECsystem-10 or -20 computer. One of these implements all the features described in this manual and is capable of producing TTS encoded output to drive a V-I-P phototypesetter.

This manual has been set as typescript using CICERO.

POSTAL ADDRESS: The Chief Superintendent, Aeronautical Research Laboratories, Box 4331, C.P.O., Melbourne, Victoria, 3001.

79 09 26 059

The data in this manual reflects the state of development of the A.R.L. programs CICERO, STENO and SEGMENTED STENO at the time of Software Change Order 76 (14, June 1979).

CONTENTS

	Page
CHAPTER 1. Introduction	7
1.1. Properties of Printed Prose	7
1.2. Text Processing Systems	7
1.3. Purpose of this manual	8
1.4. System hardware - Computing devices	9
1.5. System hardware - Composition and printing	11
1.6. System software	13
CHAPTER 2. Getting started with CICERO	14
2.1. The /SDEFAULT switch	14
2.2. Forcing a new line	15
2.3. New paragraphs and lists of short items	16
2.4. Local control of the left margin	16
2.5. Substitute space	17
2.6. Literal marker	17
2.7. Underlining	17
2.8. Steps for Creating Typescript	17
CHAPTER 3. Control Concepts	19
3.1. File structure	19
3.2. Location of controls	20
3.3. Specification of control action	20
3.4. Local controls	21
3.5. Control structure	22
3.6. Text processing modes	24
3.7. The /SDEFAULT switch	26
CHAPTER 4. Special character interpretation and definition	28
4.1. Changing text processing mode	28
4.2. Underlining	29
4.3. Substitute space	31
4.4. Case conversion	31
4.5. Continuation character	32
4.6. Change of typeface	32
4.7. Literal marker	33
4.8. Terminating a special action - /KILL switch	33
4.9. Ignoring a special character - /FORCET switch	33
4.10. Ignoring a compound delimiter - /NULDELIM switch	34
4.11. System defined special characters	34
CHAPTER 5. Facilities available in shiftable text	36
5.1. Line length	36
5.2. Line spacing	37
5.3. New paragraphs	37
5.4. Headings	39
5.5. Justification	39
5.6. Setting the spaceband parameters	39
5.7. Hyphenation	40
CHAPTER 6. Pagination	42
6.1. Page length	42
6.2. Margins	42
6.3. Columns	43

6.4. Tabulated columns 43
6.5. Page headings and footings 44
6.6. Contiguity 46
6.7. Page numbers 48
6.8. Chapter descriptors 48
6.9. Control of extra blank lines on the page 49
6.10. Forcing the start of a new column or new page 49

CHAPTER 7. Figures 51

7.1. The /FIGURE switch 51
7.2. Figure text 53

CHAPTER 8. Footnotes and Sidenotes 54

8.1. The /FND switch 54
8.2. The /FNOPTION switch 55
8.3. Numbering footnotes 56
8.4. Footnotes in multiple columns 56
8.5. Sidenotes 56
8.6. User control of sidenotes 57

CHAPTER 9. Indexes 58

9.1. Alphabetic indexes 58
9.2. Contents 60
8.3. The page reference line 62
9.4. References or end notes 63

CHAPTER 10. Symbols and macros 65

10.1. Name delimiters 65
10.2. Macros 65
10.3. Named tabstops 66
10.4. Counters 67
10.5. Predefined macros 67
10.6. Special Character Macros 75

CHAPTER 11. Special facilities for the Diablo 1610/1620 and Qume Sprint 5 printers 77

11.1. Controlling the pitch or character spacing 77
11.2. Controlling the vertical spacing 78
11.3. Controlling blank space 79
11.4. Controlling justification 80
11.5. Changing the ribbon colour 81
11.6. Changing typeface 81
11.7. Simulating bold printing 81
11.8. Adjusting multiple columns 82
11.9. Setting the line characteristics for the Diablo 1610/1620 or Qume Sprint 5 printers 82

CHAPTER 12. Facilities for Phototypesetting 84

12.1. The /ASSIGNFONT switch 84
12.2. Changing character size 85
12.3. Supershift and character conversion 85
12.4. Indenting new paragraphs 86
12.5. Listing photosetter output 86
12.6. Arguments for switches specifying vertical and horizontal measures 87

12.7. Setting the spaceband parameters 89
12.8. The /GALLEY switch 90
12.9. Kerning 90

CHAPTER 13. Typesetting Terminology 91

13.1. Historical Introduction 91
13.2. Measurement systems 92
13.3. Fixed spaces 92

CHAPTER 14. Good typography 95

14.1. Do not waste space 95
14.2. Make it easy to read 96
14.3. Add emphasis properly 96

CHAPTER 15. Setting mathematical expressions 97

15.1. Super and subscript operators 97
15.2. Line spacing to clear superscripts 97
15.3. Use of named tabstops 98
15.4. Use of macros for fractions 98
15.5. Use of macros for special symbols 99

CHAPTER 16. The Mergenthaler V-I-P phototypesetter 100

16.1. V-I-P specification 100
16.2. Standard character mapping 100
16.3. V-I-P dependent part of the /ASSIGNFONT switch argument 101

CHAPTER 17. DECsystem-10 implementation 102

17.1. Program implementation 102
17.2. Additional support programs 104
17.3. Program parameters 105
17.4. DECsystem-10 PASCAL 106
17.5. Special features for compatability with TYP2 input 106
17.6. Running CICERO programs on a DECsystem-20 106

APPENDIX A. Default Files 108
APPENDIX B. The /LIST Switch 115
APPENDIX C. Facilities deleted from STENO in order to make DIDOT 116

INDEX 118

DISTRIBUTION 123

DOCUMENT CONTROL DATA SHEET 127

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A

CHAPTER 1

Introduction

This manual describes CICERO, a scheme for coding format information into natural language text. CICERO was developed by the authors at the Aeronautical Research Laboratories of the Australian Defence Science and Technology Organisation.

Most of the manual deals with the details of the coding scheme. However, the function of this introductory chapter is to describe the kinds of text processing system which can make use of this coding scheme. The chapter also describes the basic characteristics of the hardware and software components of such systems.

1.1. Properties of Printed Prose

For the greater part of most printed prose the actual position of words on a line, column or page is unimportant to the meaning so long as the sequence of words remains in the conventional reading order. Even when the position on the line is significant, comparatively rarely is the position on the page. Consequently, when such text is typeset, the position of the information is determined almost completely by a set of conventional rules and logical restraints. Much saving in human effort is possible if the implementation of these can be assigned to a computer. This is most advantageous if the text is subject to revision as editing can be done without the need manually to adjust lines, paragraphs or pages.

1.2. Text Processing Systems

There are two distinct types of application where such automatic formatting is important:

1. Word Processing.

2. Automatic typesetting.

The output from a word processing system is typescript whilst an automatic typesetting system produces output of a much higher typographical quality suitable for printing. For commercially available systems, this division is more fundamental. The two classes of system perform different functions for different markets and by means that are largely incompatible. Most word processing systems implement formatting as an extension of editing. They allow the pagination of the output to be determined by the operator prior to printing. Small typesetting systems do automatic line division and hyphenation but do not paginate at all. Consequently, neither class of system takes full advantage of the properties noted above in section 1.1. The CICERO formatting scheme works automatically and compatibly for both types of system. Since most documents destined for printing experience many stages of editing and revision as typescript, for in-house use an integrated system offers significant savings by eliminating the need to retype the text for typesetting. An outline will now be presented of the forms such systems can take.

For the routine formatting operations to be done by computer an original source version of the text has to be entered into the computer and processed or stored in a machine readable form. The cheapest way to do this, at present, is to type the text in from a keyboard. During this typing operation it is necessary to provide some of the format control information (that part which is defined as local control in chapter 3). One way to do

PRECEDING PAGE BLANK

this is to provide extra keys on the keyboard to encode the control information. Although keyboards with extra function keys are available this obviously increases the cost of the hardware. It also creates difficulties when it is necessary to display or change the format information. For these reasons, the technique is most likely to be used only in dedicated systems. When general purpose keyboards must be used the format information has to be encoded using the normal character set. For correction of minor typing errors some visible record of what character the computer received is quite important. Most computer terminals provide this as either hard copy (typed record) or as an image on a VDU (Visual Display Unit, a TV type of display). An interesting innovation now employed in most commercial word processing system is to display not an echo of the input but instead the output typeset line. For some common kinds of error this is a useful checking facility. Unfortunately, it requires a more expensive VDU and cannot work when the input is from hard copy terminals.

The simplest means of operating a text processing system is to store the image of the text input from the keyboard, to edit this with conventional editors and finally to process it with a special typesetting program to produce an output file of formatted text. The output file may be stored in the system or output directly to a printing device. The advantage of this kind of system is that it is easy to implement on a general purpose computer system with simple terminals since such systems are usually provided with standard programs for transferring data from keyboard to disk storage and editing data on disk. Thus it is necessary only to write the typesetting program. Programs that implement the CICERO format control coding have been written for just such systems. It is important to realise that this is not the only way text processing can be achieved. The same format control structures could be used with systems that distribute more of the formatting function to the keyboard input stage. However, the editing function would then be more complex.

1.3. Purpose of this Manual

All the features of the CICERO formatting scheme described in this manual have been implemented in a PASCAL program that runs on a DECsystem-10 computer. Because many computers have PASCAL compilers this program is potentially transportable to other computer systems. However, differences in the conventions used for naming files will necessarily lead to minor local modifications.

The version of CICERO implemented at A.R.L. contains some features that are not logically necessary except when the program is used to process files created for an earlier typesetting program (TYP2). If the system were being installed at a new site these features might well be omitted.

The present version of the program allows three separate applications of the program:

1. Photosetter output.
2. Listing of that output on a hard copy device.
3. Direct setting on a hard copy device.

While the program was being developed it was logical to keep these three functions in the one program. However, separating the functions into three programs would reduce the size and probably the run time for each. This would reduce the cost of performing a given operation. However, none of the programs would individually implement the entire CICERO formatting scheme.

Finally, it is also possible that the same functions could be implemented more efficiently in a program written in a different language. In particular, a mini or micro system would probably be programmed in an assembly language and some of the features of the full formatting scheme might be omitted or modified.

For all these reasons this manual should not be taken as a data preparation guide for the CICERO program but as a description of a formatting system which may well be implemented differently on different systems. It is to be expected that both super and subsets of the format controls will be found. Consequently, before using the system it would be sensible to consult the installation manager to find out exactly what has been implemented.

1.4. System Hardware - Computing Devices

Because text processing programs perform a function of very wide application, as wide as printing or typing, it is probable that many of the people who use them will have no other contact with computing. This section is written for just such people. Its purpose is to describe the basic characteristics of the computer associated devices likely to be used in a text processing system. The following section describes the hardware for typesetting and printing. Obviously, neither description can be very detailed and therefore, function rather than mechanism will be emphasised.

A simple text processing system would contain these components:

1. A central processing unit.
2. An input device.
3. A storage device.
4. An output device.

In more complicated systems there might be several of each of these classes of components and possibly several different types in each class.

The central processing unit is the part of the computer that executes programs and controls devices. The most probable form of input device is a keyboard. So that the user of the keyboard can see what was typed and also receive messages from the system there is normally an output device associated with each keyboard. This device may produce hard copy or it may display a visible image on a TV screen (a VDU). The combination of keyboard and output device is called a terminal. Terminals may have other associated devices like paper tape readers, cassettes and floppy disks to allow data to be prepared in advance off-line. These will be described later in the section. The function of the keyboard is to generate and transmit a binary code to represent the typed character. This code is then used internally within the computer to represent the particular character. There are two common coding schemes in use with normal computer keyboards. One of these EBCDIC provides 88 printable characters and is used mainly on IBM computers. The other, ASCII provides 94 printable characters and is used by most other manufacturers. For typesetting work, another coding scheme, TTS is often used. It is fairly obvious that in order to use standard editing and data entry programs one must use a keyboard whose coding scheme is compatible with that software.

The most common technique for long term data storage at present is magnetic recording, though other techniques may be on the verge of commercial application. Essentially three geometrical configurations, tapes, disks and drums, are used. The advantage of the disk and drum geometries over tape is that the time to reach any piece of information is much less, often less than the time for one revolution. Magnetic tape is necessarily a

sequential medium and it may be necessary to scan an appreciable way through a tape to find the item required. Doing that can take several minutes. On the other hand tape is a cheaper medium and very large amounts of data can be stored. It is ideal for archiving data that is not likely to be required for some time and must be stored cheaply. In contrast, drums provide very rapid access to a limited and therefore very expensive data storage. Since such rapid access is unlikely to be required for text processing they will not be considered further. Disks may be demountable or fixed. Demountable disks provide a useful means of transferring information. They also allow access to a greater amount of data from the one drive than can be accomodated on the one disk pack. Currently, the majority of disk systems are demountable. Because mechanical tolerances can be controlled more precisely, fixed disks can provide greater recording densities and thence greater capacities of storage in the same space. For small systems, floppy disks may be used. These are small flexible disks about the size and shape of an EP record. Their capacity is limited and they are fairly slow. Analogously, tape cassettes similar to those used for audio recording are often used as a cheap substitute for conventional magnetic tape systems. Recorded data on tapes and disks is normally stored as files. The operating software usually allows files to be created and recalled by name.

It is most likely that hard copy is required from the text processing system so printers are the class of output device of greatest interest. However, it is often possible to drive other devices, like phototypesetters, off-line if some intermediate storage medium like paper or magnetic tape is available. Thus magnetic tape drives or paper tape punches may be required as the output device. Alternatively, a photosetter at a remote site might be driven via a modem link over a public or tied telephone line.

For text processing applications there are several things that should be said about the kinds of printer normally attached to computer systems:

1. Mostly they have been designed for other applications than text processing. Consequently,

2. They often do not provide lower case characters.

3. The quality of output is often less than splendid. For instance, the printer may have been designed for cheapness or for greater speed. These objectives can often be achieved with a dot matrix print head.

4. In most devices the character spacing is uniform. In some cases this is necessitated by the way it works. For instance, a line printer can print characters only in the positions where it has hammers. At least that is the theory. However, in most cases, there was no reason to provide other than uniform spacing for the intended application.

5. It is still true that most devices work by the impact of hammers on an inked ribbon. The need to be able to move all the type past this one character position in one print cycle means that a large range of characters is not compatible with a high printing speed. The design of the hammer system is also not compatible with a wide range of character sizes. For instance, IBM do not sell golf balls for their selectric composer systems with characters larger than 12 point. No doubt this is because it is very difficult to design a hammer system to cope with print of this size.

Even with all these limitations standard printers perform a useful function in text processing systems by permitting a cheap copy or listing of the output to be produced rapidly. The majority of errors can usually be

detected in this listing.

For high quality output only two types of impact device appear satisfactory:

1. Selectric typewriters.

2. Daisy wheel printers.

The Selectric or "golf ball" typing mechanism was first sold some twenty years ago. Many of its advantages and disadvantages stem from that fact. For instance, with current materials and technology, it is capable of typing at a maximum speed of 14.7 characters per second which is slow by modern standards but was fast at that time. It is one third the speed of modern daisy wheel printers and that mechanism can probably be designed to run faster still. Another disadvantage is that no version with proportional spacing and capable of being attached to a computer system seems to have been sold except as part of an IBM system. A significant advantage is the wide range of typefaces available.

Apart from speed daisy wheel printers offer the advantage that the horizontal and vertical spacing can be controlled quite precisely by program. This permits some pleasing typographical features. For instance, lines can be justified by increasing the interword spacing by a small amount rather than by adding whole spaces. The impression of a blank line between paragraphs can be created whilst actually spacing by rather less than a full blank line. The minimum interword spacing can be appreciably less than a normal character width whilst still allowing the word break to be evident. These techniques not only provide a pleasing appearance, they also save paper.

The special controls required to operate these features have been incorporated into the CICERO format control scheme. Thus programs written to implement CICERO should be able to drive either Selectric typewriters or daisy wheel printers. At the time this manual was prepared no proportionally spacing typewheels were available so the CICERO system does not provide a means to read width tables for this class of output device. That might form a future extension to the scheme.

Ink jet printers are now being offered by some manufacturers, notably IBM. They can operate at about twice the speed of daisy wheel printers but are significantly more expensive (about 7 times the price). The IBM 6640 Document Printer offers other advantages. For instance, it provides up to five program selectable typefaces on line and can feed automatically non-continuous stationery, even envelopes. Against these capabilities, it does not provide changes of typesize although it costs as much as many phototypesetters. Obviously, the correct choice of hardware depends on the application. For some it might be much more important to be able to address envelopes than to vary typesize. CICERO does not at present offer any particular support features for ink jet printers.

Phototypesetters will be considered in the next section. Some may be interfaced directly to computers and used as output devices.

1.5. System Hardware - Composition and Printing

The previous section described the characteristics of some common computer output devices capable of producing hard copy. The printing trade uses a different terminology. The process that has so far been called printing, that is the production of a primary image of the text is called composition. Printing refers to techniques for the reproduction of that image. Before the introduction of computer based systems composing was done largely by assembling lines of text cast in lead, the output of hot metal typesetting machines. In addition to the desirable properties

already listed and not necessarily provided by computer hard copy output devices these machines provided:

1. Proportional letter spacing.

2. A large variety of typefaces and styles.

3. A wide variety of typesizes.

Because composition of the final page was done by hand, literally clamping pieces of lead together it was possible to mix the output from several typesetting machines. Now it is evident that for a computer based system providing automatic pagination, the composing device has to be able to provide all the typefaces in all the styles and sizes required for the particular document in the one machine. The view of the trade is that photocomposition is the most promising technology capable of achieving this objective.

Phototypesetters work by exposing an image of each character required at the appropriate place on photographic paper or film. Various means are used to select and position the character. Slower machines use a Xenon flash to select the character from a mechanically movable mask. The image of the character is then magnified or reduced appropriately by passing it through a mechanically modifiable lens system and then positioned by optical deflection. Faster machines generate the character on a CRT. Usually the source image of the character is stored in a source grid. A raster scan is positioned electronically so as to illuminate the appropriate character. The signal from a photocell positioned the other side of the grid is then used to modulate the beam of the output CRT. Character size may be controlled by varying the ratio of the input and output CRT raster sizes. Some photosetters also vary the typestyle electronically by varying the input or output raster shapes. Some systems dispense with the input grid and raster scan, storing the image in digital form in memory within the setter. The important points to note are that CRT driven setters are on average somewhere between 8 and 12 times as fast as opto-mechanical setters and again on average cost between 2 and 3 times as much. Within each class there are variations in cost of about the same amount depending on the machine capabilities. The development of laser driven phototypesetters has now reached the production stage. These generate the characters by electronically modulating a mechanically scanned beam of light from a laser. The character is produced effectively as a raster scan.

Most phototypesetters incorporate a small mini-computer to provide the logical processing power for automatic line division, hyphenation and justification. A few slave setters designed to be operated directly from a computer system do not. However, there is some advantage in making the typesetter intelligent since it is then not necessary to precede each character selection code with positioning and flash information. It is the belief of the authors that at present no phototypesetter is sold with a sufficiently powerful inbuilt computer to provide automatic pagination. There are in existence a number of computerised composing systems that do provide automatic pagination. However, they have all been designed to use additional computers. Programs to implement CICERO for photocomposing are intended to be run in such additional computers.

Most phototypesetters are designed to read input from 6-level TTS encoded paper tape. Unfortunately, this tape cannot be produced by the 8-level paper tape punches found on most computer systems. Conversly, TTS encoded keyboards do not seem to be available in any form that is suitable as a computer terminal. Fortunately, some typesetters will also accept 8-level TTS encoded paper tape which can be produced by the normal computer

paper tape punches. Magnetic tape drives are also available for most of the fast phototypesetters and some of the slow ones. It is also possible in many cases to provide an interface for connecting the phototypesetter directly to the host computer system. A serial interface is a particularly good one to use as it means the phototypesetter can be made to look like a terminal to the computer system. This means a minimum of modification to the operating system software in order to connect the phototypesetter to a general purpose computer system.

1.6. System Software

Programs that implement the CICERO text formatting control scheme represent only a small part of the total software requirement for text processing systems. However, except in systems designed exclusively for text processing most of the other components will already be available. They are likely to differ significantly between systems.

Probably the most important program to run on most large computer systems and many small ones is the operating system. The function of the operating system depends on the architecture of the machine and the purpose for which it is intended. For a time-sharing system, it schedules the running of jobs, processes users' commands, protects users from each other and allocates resources. A timesharing system is a rather special case of a real time computer system. By replacing the words "job" and "user" by "task" or "transaction" the description may be applied more generally.

Editors are another very important class of program. An editor is a program to allow a file, usually a text file, to be amended easily and efficiently. Unfortunately some editors used for modifying source language programs are line number orientated. That is all commands refer to line numbers. When editing files of text from a terminal a string orientated editor is much easier to use. A string is a sequence of characters. A string orientated editor allows one to find the place in the file by searching for a string of characters. Text of any length can then be added, deleted or moved about. A string orientated editor should also be able to detect every occurrence of a particular string taking repeatedly the same action each time a match is found. A facility of this kind provides a ready means to correct spelling errors or amend names.

Finally, a number of utility programs are required to transfer files between devices or control special devices. Programs to input data from a keyboard and store it in a named disk file, or to transfer files from disk to magnetic tape are examples of this kind. One program that was found quite useful in the development of the photosetting versions of CICERO on the DECsystem-10 was one that produced a listing from a TTS tape. This is much easier than trying to read a tape visually.

CHAPTER 2

Getting Started with CICERO

This chapter presents a basic subset of CICERO format control instructions. The introduction of the facilities described at this stage in the manual serves two useful purposes:

1. It provides some practical "feel" for the system. This practical background will be most useful when presenting the more abstract concepts of the next chapter.

2. It permits the casual reader to start using the system without having to read the whole book. The authors, whilst believing this to be a legitimate approach to the problem of learning CICERO, hope that users of the system will go on to read later chapters as they realise the limitations of the basic subset of instructions presented in this chapter.

Surprisingly, the seven facilities described allow a great number of tasks to be completed satisfactorily and without compromising the typographical quality of the output.

Because the control concepts underlying CICERO have not yet been described, little explanation can be provided about the action of each control. Necessarily, this chapter will present a "cook-book" approach.

Figure 2.1 shows a short demonstration source file containing text and format control information. Figure 2.2 shows the typeset output after processing with the A.R.L. program that implements CICERO. The output was printed on a Diablo 1620 printer. The various controls in the input file will now be described.

2.1. The /SDEFAULT Switch

The first line of the source file of figure 2.1 is:

```
.BASIC/SDEF
```

To use the basic subset of CICERO described in this chapter it is necessary to start the source file in this way. What this command does is to read a file containing default switch settings and control character definitions. To avoid repetition the control sequences so defined will be called the "BASIC SUBSET" throughout the rest of this manual.

The dot at the start of command line is important as it indicates that the line is a command string and not text. Similarly, the slash is a herald for a switch. Switches occur only in command strings so this does not define any action for a slash should it occur in ordinary text. SDEF is more than a sufficiently unique abbreviation for the full name of this switch which is /SDEFAULT. BASIC is used as a file name when selecting the source file from which to read the default control information. Because file naming conventions may be expected to vary between computers it is possible that this part of the command string might be installation dependent *. The commands contained within the default file set up page and line lengths as well as defining the three special control characters of the basic subset. A property of the file named BASIC read by the /SDEFAULT switch is that it sets the text processing mode to shiftable text. Text

* For installation managers, the data contained in the file BASIC.TTY which this switch reads on the DECsystem-10 implementation at A.R.L. is listed in Appendix A.

```
.BASIC/SDEF(1)
_Conclusions_(7)
      (3)With the assumptions made in this study,
we can make two important generalisations for the
efficiency of neutral deuterium injection:

(2)
&***1.**(5)In general,
      (4)independent of ion type, the component
performances with the largest leverage on the
system efficiency are, in decreasing order, the
accelerator efficiency, the trapping fraction, the
neutralizer loss direct conversion and the neutralizer
power efficiency.

&*(6)**2.**At low
      energies, independent of ion type, the
performance of the reactor thermal converter
and the source are important.
```

Fig 2.1. Source text to produce the formatted version shown in fig 2.2. The constructs referred to in the text are shown in round brackets and are

(1) /SDEF switch.

(2) Blank line forces start at left margin.

(3) Tab causes new paragraph.

(4) Leading spaces cause indentation.

(5) Substitute spaces.

(6) Literal Marker.

(7) Underline delimiter.

processing modes will be explained in chapter 3. Most files called by the /SDEFAULT switch will not change the text processing mode. The reason this one does is because the controls defined in the basic subset provide no means for changing the text processing mode. Shiftable text is the processing mode required for automatic line division by the program. The controls defined in the default file for the basic subset will also cause line justification.

2.2. Forcing a New Line

Comparatively often it is necessary to force the start of a new line at the left margin, usually separating it from the previous text with a blank line. Headings are an obvious example. There is a simple way to do this for the shiftable text processing mode implied for the basic subset of

Conclusions

With the assumptions made in this study, we can make two important generalisations for the efficiency of neutral deuterium injection:

* 1. In general, independent of ion type, the component performances with the largest leverage on the system efficiency are, in decreasing order, the accelerator efficiency, the trapping fraction, the neutralizer loss direct conversion and the neutralizer power efficiency.

* 2. At low energies, independent of ion type, the performance of the reactor thermal converter and the source are important.

Fig 2.2. Result of setting the text of fig 2.1.

commands. In shiftable text a double carriage return line feed combination i.e. a blank line, in the source causes a blank line to be output and the next line to be started at the left margin. This is useful for headings and lists of numbered items. More blank space can be added by adding further carriage return line feeds to the source if it is necessary to separate a major heading from previous text. Examples of the use of a double carriage return line feed to force the start of a new line are shown at (2) in figure 2.1. For brevity the sequence carriage return line feed will be written as CRLF throughout the rest of this manual. In many systems pressing the key marked "RETURN" causes the character sequence CRLF to be transmitted to the program reading the data in the computer.

2.3. New Paragraphs and Lists of Short Items

For the shiftable text processing mode implied for the basic subset of commands a tab at the start of a line is used to herald a new paragraph. A new paragraph is characterised by an additional blank line to separate the paragraphs and indentation of the left margin for the first line of the new paragraph. CICERO provides means to control the interparagraph space and the indentation of new paragraphs but these facilities are not part of the basic subset. Examples of new paragraphs set this way are illustrated at (3) in figure 2.1.

If successive lines also start with a tab the blank line is omitted. This facility provides a simple means to construct lists of items short enough to fit on one line. If the text extends over a line in the output, the item will be separated from the next by a blank line as CICERO will recognise a new paragraph. Better means are available for lists of longer items.

2.4. Local Control of the Left Margin

A block of text in which the first line starts at the left margin but for which subsequent lines are indented is called a "hanging indent". This particular structure provides a visually pleasing way to create lists of longer items. CICERO provides several ways to achieve this effect

efficiently, (see for instance the <LM> predefined macro). When using the basic subset a hanging indent may be created by adding spaces to the start of a line. The width of the spaces becomes the indentation of the left margin after the line currently being assembled has been output. The indentation persists until cancelled either by a double carriage return line feed or by leaving shiftable text. Starting a new paragraph does not cancel the automatic indentation. The use of leading spaces to create a hanging indent is illustrated at (4) in figure 2.1.

Whilst this facility is part of the basic subset of CICERO controls, more experienced users will probably prefer other means to achieve the same result. All users should guard against inadvertent indentation caused by stray leading spaces, the result of poor editing. Because better means are available to control indentation and because of the ease with which spurious indentation can result from this facility, some implementations of CICERO may prefer to turn this feature off.

2.5. Substitute Space

There is a need for a space that will not be interpreted as the end of a word and that will not be expanded during the justification process. For instance, such a space can be used to ensure that numerical values are not separated by a new line from the units of measurement or initials from a surname. Lack of expansion by justification is important when numbering paragraphs or lists of items as it allows the space between the number and the text to which it relates to be kept constant. Variations in this separation can be very obvious if the numbered line is followed by a hanging indent. In the basic subset an asterisk, "*" is defined as the substitute space character. In the output it is replaced by a space. Examples of the use of substitute spaces are shown at (5) in figure 2.1. CICERO provides means for defining the substitute space character but these are not part of the basic subset.

2.6. Literal Marker

It would be very inconvenient if the definition of "*" as the substitute space prevented the appearance of an asterisk anywhere in the output text. The problem is far worse when using the advanced features of the CICERO system as many more characters are required for special control. A simple solution is to define a literal marker. The special action of this character is to cause the next to be processed literally and not as a control character. In the basic subset "&" is defined as the literal marker. Then "&*" is transmitted to the output as "*" and "&&" as "&". This is illustrated at (6) in figure 2.1.

2.7. Underlining

Underlining is discussed in much greater detail in a later chapter. The problem is complicated because not all printers have an underline character and when they do the method of underlining may vary. For these reasons CICERO provides eight different classes of underlining and the DECsystem-10 implementation allows up to 16 different types chosen from these 8 classes for any particular document. All forms of underlining are turned on and off by a controlling delimiter. The basic subset defines one such delimiter, the ASCII underline, "_". There is, therefore, only one kind of underlining available when using the basic subset. The kind of underlining performed may be chosen by the installation manager to suit the equipment available. Examples of the use of the underline delimiter are shown at (7) in figure 2.1.

2.8. Steps for Creating Typescript

There are three basic steps required for the creation of typescript with CICERO. These are:

1. Create and edit the source text.

2. Format the data with the appropriate CICERO program.

3. Print the output.

It is probably useful to demonstrate these steps for a particular system. For the DECsystem-10 these stages become,

1. File creation. The input file is created by running the text editor TECO in the create mode via the MAKE command. Thus,

```
.MAKE SOURCE.TXT
*I.BASIC/SDEF
..... $$
*EX
```

   Whilst for subsequent editing TECO is called in an edit mode via the TECO command,

2. The appropriate CICERO formatting program for typescript is used to produce a formatted output file. This program is called STENO at A.R.L. Then,

```
.R STENO
*OUTPUT.LST=SOURCE.TXT/GO
```

   generates a file OUTPUT.LST containing the formatted output.

3. Suppose this output is to be printed on a Diablo 1620 printer. To drive that at its maximum speed a special protocol is required. This is not implemented by the DECsystem-10 operating system so a special program has to be used. Thus,

```
.R DIABLO
*OUTPUT.LST
```

   causes the file OUTPUT.LST to be printed on the Diablo terminal. If there were several such terminals the command string would have to specify which to use, otherwise that can be defaulted. If power to the Diablo printer has been turned off, it is necessary to set the top of form before commencing printing as that information is held in the microprocessor memory which is lost when power is turned off.

CHAPTER 3

Control Concepts

The remaining chapters in this manual describe in precise detail the controls required to produce particular formatting functions. The purpose of the present chapter is to take a more abstract view of the tools available for implementing this control function and to present some of the reasons for the design decisions.

The authors realise that while some people will gain clarity and insight from this chapter, others will find it heavy going. The following summary is for the benefit of such people who should read it and then go on to section 3.6.

Basically, this chapter is about control. One of the most important observations is that two classes of control may be identified for the automatic formatting of files containing format instructions. These are:

1. Those whose action is tied tightly to the position of the control in the input file. These are called local controls. An example would be a delimiter which controls the limits for underlining. (i.e. underline from here to here).

2. Those whose action relates to the output rather than the input and whose action is therefore, much less tightly coupled to position in the input. These are called "global controls". An example would be a control to set page length.

The first kind can be repeated frequently and must, therefore, be implemented efficiently. CICERO implements the more common controls as single characters and allows single character implementation of more complicated controls through the special macro facility (see section 10.6). Sensible use of the second kind of control will not result in their frequent repetition so they may be implemented by more verbose but explicit means. CICERO represents such controls by switches contained within command lines which may be entered either from the terminal controlling the job or in the input file. In the latter case, the command line starts with a dot. Typically, the format of a switch might be

```
./NAME:argument
```

though the exact form varies with the switch. Cursory readers may now proceed to section 3.6.

3.1. File Structure

Both the input and output files are sequential. The order of the input sequence is that in which the characters were typed with modifications resulting from subsequent editing. With minor exceptions this order is also that in which the information would be read from the output. The requirements of automatic pagination necessitate the declaration of some items at points in the text other than those where they will be printed either so that they may be referenced or a page number may be associated with them. Examples are footnotes and index entries.

The sequence of the output file is according to the order in which the characters must be typed or transmitted to the phototypesetter. This order is not necessarily the same as either the input sequence or the reading sequence. For instance, backward movement of the paper is beyond the capabilities of most paper tractor systems fitted to printers. Consequently, to print text in multiple columns it is usually necessary to print

right across the page ordering the lines according to the sequence of their base lines down the page rather than printing columns in turn and reversing between them. Similarly, superscripts must be printed before the line containing the indexed character and subscripts on a line below and printed after it.

3.2. Location of Controls

Two means are available to control the formatting function:

1. Terminal command strings.

2. Commands embedded in the input file.

Terminal command strings are entered from the terminal from which the program is being controlled. Their function is to specify the run parameters. Generally, these are the files for input and output, though what will be defined later as global controls may also be specified. Command strings of the same format as those used from a terminal may also be embedded within the input file. However, not all the commands in the input file are of this kind.

All command strings entered from the terminal are processed before the input files are read. Further some commands entered at this time may cause a preliminary pass of the input file. During this preliminary pass it is necessary to process command strings embedded in the text since they may redefine control characters. Such strings, therefore, are processed twice. Terminal command strings can never be processed more than once.

Logically the input may be thought of as one sequential file starting with the terminal command strings and followed by one or two copies of the input file(s). This file is processed sequentially. Note that sequential processing does not necessarily imply a forward action for each control, since the text can be stored and processed in a manner to be determined when a subsequent control is encountered. Equally, the flow of information is not necessarily only forward since, when information is required from input text not yet processed, the intervening text may be stored and processed subsequently or processed and stored temporarily depending on circumstances. The latter approach is the one used in the A.R.L. implementation to allow table of contents to be printed anywhere within the output document, not just at the end when the page information has been determined. When storage is limited the preliminary pass provides a simple means to collect information in advance in order to make it available out of sequence where it is needed. This technique is used for those references that have to be numbered after sorting into alphabetical order, since the number is required when the text citing the reference is processed but cannot be calculated until all references have been read.

3.3. Specification of Control Action

Two properties are needed to specify the action of a control:

1. What it does.

2. When it does it.

Usually, it is not difficult to specify what a control does. The problem comes in determining when it does it. In a sequential processing system "when" really means at what stage in the processing of the logical input file does the control have effect. As a simple example, consider a control whose action is to set a new page heading. Such a change will not be effective until the current page is output. Depending on how the control is defined the action might be to change the heading on the current page or on

the next. For logical reasons, CICERO changes it for the next page. However, the important thing to note is that the time when the control is effective depends on a future event, usually related to the setting of the output file. It is not a simple matter to predict at what stage in processing the input file that event will happen. In contrast, some controls have significance limited to the input file processing. The literal marker mentioned in the previous chapter is a good example. It is effective immediately on the next character of the input file.

Evidently, there exists a class of control whose time of action is simply determined from the input sequence and another class for which this is not so. For want of a better terminology, the first class will be called "local controls" since the time of action is tightly coupled to their location in the input file sequence. The second class will be called "global controls".

3.4. Local Controls

CICERO uses five different types of local controls which will now be described.

3.4.1. **Delimiters** are used to specify the limits of some kind of field, action or mode. A simple example from the previous chapter is the underline delimiter. Notice that in this case the same delimiter is used to terminate the action as is used to initiate it. Sometimes that is not desirable. There are two important repercussions of using one delimiter for the two functions:

(a) Recursive techniques are prevented. That is further action within the area of action is impossible. This is an effective way to prevent a user specifying unimplemented recursion. For instance, CICERO does not permit footnotes within footnotes. To implement such a facility would require a much more complicated algorithm than the one used and seems rather pointless. The way footnotes are specified is by enclosing them within delimiters at the point where they are called. By allowing only one kind of footnote delimiter which is used to initiate and terminate the footnote, CICERO removes any possibility of specifying a footnote within a footnote. For recursion to be possible it is also necessary for the text between the delimiters to be processed by the same processing code as processed the initial delimiter. If that does not happen, the user may be allowed to specify different opening and closing delimiters since further opening delimiters within the delimited string of characters will not be recognised. Recursive techniques can be useful and are particularly important for the specification and expansion of macros described in chapter 10.

Unfortunately, the limitation to one delimiter for non-recursive operations makes it logically impossible not only to specify unimplemented recursion but also to detect an attempt to do so. This is a bad side effect since no error message can be given to the user.

(b) The omission of one delimiter causes repercussions for the processing of the remainder of the file. No doubt this annoys some people but it is not necessarily bad. What it really means is that the error message is quite spectacular.

Evidently, whilst one delimiter may be logically sufficient error detection is easier if there are different opening and closing delimiters. Since the ASCII character set is finite this may be difficult to arrange. An alternative expedient would be to design a special input program for

CICERO which would make the user immediately aware of the implications for the output of what he has just typed.

3.4.2. Separators are used to separate one field from another. For instance, an index entry may contain separators to separate sub-levels within the entry (see section 9.1).

Separators may be used to separate rather than delimit modes. In this usage, they differ from delimiters in that there is no subsequent turn off marker so there is no need to remember the states before the marker. Separators of this kind are useful for setting point size or typeface where clearly there is no rational reason to assume the previous setting will necessarily occur again. Quite often, it is possible to implement a function with either delimiters or separators. CICERO implements changes of text processing mode with delimiters but it would also be possible to do it with separators. The cost would be an extra control character.

3.4.3. Local Action Markers have an effect which is limited in range often to just the next character. It is, therefore, important to remember the previous processing mode and status. An example of such a control is the literal marker introduced in the previous chapter. Others are the case and supershift markers which are effective for the next character too. Actually a repeated case shift marker suppresses case conversion until the end of the next word or the first non-alphabetic character.

3.4.4. Immediate Action Markers have an immediate effect but cause no change in processing mode. Three examples may be taken from the basic subset of controls presented in the last chapter:

1. Double CRLF to return to the left margin.
2. Leading tab to start a new paragraph.
3. Substitute space.

Actually, the text formatting program needs to take special action for some characters without the user being aware of it. For instance, CICERO will normally provide a double space in typescript after a dot used as a full stop. A full stop is recognised as a dot followed by a space or carriage return in shiftable text. Occasionally, for instance, when the dot signifies an abbreviation, this is not the right action. Then the literal marker may be used to suppress this special action.

3.5. Control Structure

Because of the weakness of the coupling between action and location within the input file, sensible use of global controls will never require their frequent repetition. Many may be expected to occur no more than once in the input file. Thus clumsy and verbose means for global specification can be tolerated if, for instance, the coding becomes more readable. The /SDEFAULT switch introduced in the previous chapter, itself an example of a global control, provides a simple means of specifying standard global control settings implicitly.

3.5.1. Command Strings and Global Control

There are significant advantages to be gained through allowing global controls to occur not only in the input file but also in command strings entered from the controlling terminal or batch job stream. One advantage is that they are more easily changed between runs. Another is that such command strings are read in one pass whereas two passes may sometimes be made when processing the input file. For the DECsystem-10 computer, command

strings consist of file specifications and switches. Switches provide a convenient mechanism for setting parameters to be used by the program. The switch mechanism seems to be a good way to implement many of the global control functions, for instance to set page and line sizes. However, specification of files may also be seen as a global control function. Consequently, a command string may be regarded as a collection of global controls. This suggests that the general way to implement global controls is to allow command strings within the input file. All that is required to implement this is a convention for delimiting such strings. For CICERO that convention is that a command string within the input file starts with a dot at the start of the line and ends at the first line feed after the command string interpreter has finished processing the command string. The literal marker, already described, provides a neat way to enter lines that are required to start in the output with a dot. This definition of a command string within text specifically allows some global controls to occupy several lines. The /ASSIGNFONT switch used for photosetting, the /HEADING and /FOOTING switches are all examples of such multiple line controls.

3.5.2. Switch Structure

The structure of the file specification part of a command string is likely to depend on the computer system. That for the DECsystem-10 is described in chapter 17. Some means has to be provided to separate switches from file components. CICERO assumes that a slash, "/" may be used for this purpose and that it is not a valid component of a file name specification. On some systems it might be necessary to use a different character.

If it is assumed that CICERO will necessarily process a switch from left to right it is sensible to identify the switch first before looking at its arguments as the number and type of these will vary for different switches. Since intelligibility and clarity are more important than brevity for global controls, it is reasonable to identify switches by name. The name must terminate with a separator. If the switch has no arguments the separator could be a space, a carriage return or a herald for another switch name. For CICERO valid switch names are constructed from alphabetic or numeric characters only and when processing the name lower case alphabetic characters are converted to upper case. If valid switch names are kept in an ordered structure in the implementing program, once having checked that a supplied switch name is not an exact match, there is usually very little overhead in checking that it is a unique abbreviation. Consequently, CICERO allows switch names to be abbreviated so long as the name remains unique. Not all the switches have arguments as some simply set a boolean flag. When arguments are specified they must be separated by separators. The arguments may be numbers, special characters or strings of characters. The latter must be enclosed within delimiters. The appropriate separators will be described for each switch. Generally, a colon, ":" is used to separate the switch name from a numeric value or a special character. For instance,

```
/LINE:56

/STD:#
```

3.5.3. Structure of Local Controls

Local controls occur much more frequently than global controls so it is important to specify them efficiently. In many instances, this may be done with a single reserved character since the control merely establishes or terminates some mode. The reservation of some characters for this purpose does not restrict the set available for output as the literal marker allows

the control function to be overruled. CICERO allows the specification of such special characters as a global control function. This means not only that the set of control characters can be modified to suit particular applications but if there is no need of a particular control, no special character need be reserved for it. Thus the basic subset of controls presented in the last chapter required only three special characters ('*', '_' and '&') and it was unnecessary to tell the user of any others because they were not defined.

It is not always convenient to use a single character as a local control. Thus in the previous chapter, the double CRLF to force the start of a new line at the left margin actually requires four characters (two key strokes on many systems) whilst starting a new paragraph requires the sequence CRLF TAB, three characters. However, these particular choices happen to be very natural ones and impose far less strain on the user's memory than would the definition of some special local control for these functions.

In some cases, single characters could be used but doing so would tie up a very large number of special characters. In these cases it is better to construct a composite local control enclosed within delimiters. Examples are the heralds used to change typeface or point size for the photosetter.

Finally, there are a few cases where the local control is used so rarely that there is no point in allocating a special character for it. The predefined macro call described in chapter 10 is a simple mechanism for arbitrarily extending the number of local controls without increasing the list of reserved characters.

3.6. Text Processing Modes

One way to reduce the amount of control information that has to be provided is to group it together into modes. By saying a processor is in a particular mode we mean that its actions are limited to a subset of the total possible. This is a common concept for computer central processors but is just as applicable to any logical processor including a text processor.

The most important processing mode which will be called shiftable text, has already been described. In this mode, words from the input file are assembled on the output line until the line length is exceeded. The last word is then either hyphenated or shifted as a whole to the next line and the output line is then set. The mode is called "shiftable" because words can be shifted without regard to the line structure of the input file, provided only that the sequence is preserved. Note that nothing has been said about justification. This is because justification is not the important characterisitic of this mode. Once the line break decision has been made by the program, the line can be justified by expanding the spaces or if preferred a ragged right margin can be set. What characterises this mode is that the division into lines in the input file is ignored, the program performing this function itself for the output. This makes editing of the text very easy since chunks of text can be added or deleted and the output line structure will adjust automatically to give a uniform or near uniform line length.

Four other text processing modes appear to be useful. Three of these correspond to the quadding functions provided in the TTS code for linecasters (see section 13.1). These are:

Free text (quad left)

Right text (quad right)

Centre text (quad centre)

The free text processing mode causes the line format of the output to follow that of the input. This mode is useful for headings, tables and more complicated structures like mathematical equations. The right text processing mode is similar to free text except that from the point of entry text is shifted to align the last character of each line against the right margin. Right text is useful for equation numbers and dates on letters. Also if a heading extends over two lines, the part on the second line can be shifted right to distinguish it from a sub-heading. Finally, the centre text processing mode is similar to right text except that text is centred instead of shifted right. The persistence of this mode saves appreciable typing of control information when setting text like title pages for books or reports where many lines are centred. In the fourth processing mode, null mode, text is processed to the stage where index entries are stored but it is not transferred to the output. This mode provides a convenient means for linking index entries to the page where the subject is mentioned without the need for the entry to appear explicitly on that page. Often it is used to set entries in a table of contents in a slightly different form to the headings on the pages referenced.

CICERO delimits these text processing modes with single character delimiters. A system based on delimiters rather than heralds must either save each processing mode on a stack each time a new one is entered, restoring the previous mode when the terminating delimiter is found or it must have an implied hierarchy to establish which of several modes current is actually operative. CICERO uses the latter method. One of the advantages is that the lowest mode in the hierarchy requires no delimiters since it happens by default. The choice of free text as the default processing mode enables the program to be used to paginate or repaginate text that has already been divided into lines. This could be useful if the input is computer output. The mode hierarchy is, in increasing order of priority:

Free text mode
Shiftable text mode
Right text mode
Centre text mode
Null mode

The definition of these controls (described in the next chapter) is a global control function since the control is not effective until the delimiter is encountered in the input text. Thus the dependence on location in the input file is usually not critical.

In addition to the text processing modes described above, it is sometimes necessary to shift a line of text, for instance a chapter heading, to the outer edge of a page where it will be most noticeable. This requirement implies differing action depending on whether the page numbering is odd or even. The normal binding convention is that odd numbered pages are on the right. CICERO handles this requirement by the introduction of two sub-modes of free text. Thus in the hierarchy of processing modes, they take precedence over free text but are ignored in all other processing modes. Because, they meet a rarely occuring requirement, they are delimited not by single character delimiters but by predefined macros. The description of these is left to chapter 10.

It is important to specify the action upon changing text processing mode. On leaving shiftable text the normal requirement is to set the current line without justifying it. In typesetting terminology, the line is quadded left. The same action is also required when in shiftable text a new paragraph is started or a start at the left margin is forced. Care has to be

taken in any of these circumstances if the line is overlength and the right action is sometimes to transfer the last word to the next line, set the balance of the line justified and then output a quadded line containing just the last word. A further problem occurs when the minimum space between words in justified text is less than the mean space used for quadded lines as the line may then be overset when the bigger value is used but underset when the line length is calculated using the minimum width. The text formatting program must handle these problems without requiring intervention from the user. On leaving the other processing modes the line should be set according to the previous processing mode. On entering all text processing modes other than right text a new line should be started. For right text it is useful not to start a new line as this permits part of the line to be quadded right, for instance, to append an equation number.

3.7. The /SDEFAULT Switch

The final section of this chapter is a fuller description of the role and action of the /SDEFAULT switch than could be given in chapter 2. The full format for this switch is,

```
File specification/SDEFAULT
```

It may be given in a terminal command string,

```
*File spec/SDEF
*
```

where "*" is the prompt character requesting data to be entered from the keyboard. Normally, in this mode of operation the user would define the default file or files first and then enter the specifications of the input and output files after the second "*". Alternatively, the default file specifications and /SDEFAULT switch can be included in a command line in the input file. Such a command line must be preceded by a "." at the start of the line.

The file specification will be system dependent. For the DECsystem-10 it takes the following form:

```
DEV:FILE.EXT[AREA]
```

where

DEV is a device or disk structure name.
FILE is a six character file name.
EXT is a three character file extension name.
AREA is a project programmer pair.

Certain defaults apply. If the .EXT field is omitted CICERO looks for an extension .SET if it is photosetting or .TTY if it is producing typescript only. The output device will normally be DSK. The AREA will be defaulted to the user's current directory path by the operating system. However, if the file is not found the program will search the SYS area or on some systems the PUB area. Essentially, the behaviour is the same for the DECsystem-20 except that users will not be used to addressing directories by project programmer numbers.

If no device, file name or extension is given, CICERO will default to DSK:CICERO.SET if it is photosetting or DSK:CICERO.TTY if it is producing typescript. The ability to change the default depending on what kind of processing is being performed is important since it means that when the /SDEFAULT specification is included in the source file a different default

will be read depending on whether typescript or photoset copy is being set without the need to modify the source file.

Several default files may be read during the single setting operation. For instance, one might have global switch specifications in one file and macro definitions in another. Each such file specification must be followed by the /SDEFAULT switch.

In practice almost all global definitions can be written into default files. Then the typist performing the primary input, merely has to specify a default file name for that class of work and enter the source text together with appropriate local controls. The definition of the default files can be done by someone more skilled. Further some effort can be expended on developing and testing the default file to the stage where it will perform reliably what is required. Some sample default files are given in Appendix A.

Clearly, there is merit in adopting a standard set of special characters as delimiters. These definitions are all global assignments controlled by switches. Those suggested in Appendix A work reasonably satisfactorily for a variety of different ASCII keyboards and perhaps should be adopted as standard.

CHAPTER 4

Special Character Interpretation and Definition

The local control function in CICERO depends on the definition and use of special characters. Nine characters are always treated as special by CICERO. They are:

BS backspace

HT horizontal tab

LF line feed

VT vertical tab

FF form feed

CR carriage return

SP space

, comma

. dot

CICERO also allows the definition of some 23 different types of special character by the user. Ten of these will be described in this chapter and the remainder later in the manual. It is relevant to observe that there are some 27 ASCII characters that may readily be used for such control purposes in most documents. These are:

```
!"#$%&'()*+-/<=>@[\]^_`{|}~
```

and although some functions require more than one special character or may have different forms each requiring a different character, the code does provide enough suitable characters for all normal applications. The assignment of special characters to local control functions is done by switches in global command strings. This chapter will describe the function of the local controls and where appropriate, the defining switches. The special action of these assigned characters may also be terminated or suspended and ways of doing this will also be described. The ordering of the chapter is probably not ideal from a reference viewpoint but has been chosen so as to introduce concepts in a logical sequence for ease of reading

4.1. Changing Text Processing Mode

Delimiters for the shiftable, right and centre text processing modes and for null mode are defined by the /STD, /RTD, /CTD and /NMD switches. Thus,

```
./STD:# /RTD:$ /CTD:@ /NMD:'
```

defines "#" as the shiftable text delimiter

"$" as the right text delimiter

"@" as the centre text delimiter

"'" as the null mode delimiter

and kills any special action that might have been associated with these particular characters before the global command was issued. Since it is

the default processing mode a delimiter for free text is not required. There is merit in adopting a standard system of delimiters and Appendix A contains listings of various standard default files which do this. The above definitions are consistent with those of Appendix A. Rarely, would one need to include the definition of these processing mode delimiters within the input text. Normally, they would be in a default file.

Defining a shiftable text delimiter causes subsequent text enclosed within shiftable text delimiters to be processed in the shiftable text processing mode. That is the line division is performed automatically. Justification may or may not take place. The control of processing in shiftable text is described in the next chapter of this manual (Chapter 5).

In most instances the action upon encountering one of the above delimiters is to output the last line in the current processing mode and to start a new line in the new mode. However, note the following exceptions:

1. On leaving shiftable text, the last line is output quadded left (ie. is not justified).

2. On moving from free text to right text, the present line is not forced out but the right text is appended to the line and spaced from the free text so as to align it against the right margin. This allows equation numbers to be appended to equations and so on.

3. If the previous line is blank it will not be output.

4. If the previous line was the number, key or printers mark at the start of a footnote, the line will not be output on changing to shiftable text. Instead the following data will be appended and the whole line treated as shiftable text.

4.2. Underlining

The hardware available for underlining leads to the requirement for four different classes of underline:

1. An underline on a separate line but with no line advance. That is, the previous line ends with a carriage return but there is no line feed. This is specified by the /UCR and /UFCR switches. It is suitable for printers that have an offset underline and separate carriage return and line feed functions.

2. An underline on a separate line with its own line advance. The previous line ends conventionally with a CRLF sequence. This is specified by the /ULF and /UFLF switches. It is suitable for printers that must underline with a minus, also for photosetters when an EM or EN rule is used for underlining.

3. Underlining on a character by character basis following each character to be underlined with a backspace (BS) and the underline. This is specified with the /UBS and /UFBS switches. It is suitable for printers that have an offset underline and implement the backspace function. Few line printers do and for them the effect is usually to print the underline after the character. Almost all VDUs store the characters to be displayed in random access memory, there being provision for one character per position in the raster scan. Backspace is implemented but the next character transmitted overwrites the previous one for that position.

4. A non-spacing underline which is transmitted before the character to be underlined. This option is specified by the /UNSP and /UFNSP switches. It is the most efficient way to implement underlining since no extra

characters are transmitted apart from the underline and the underlined character. This method of underlining used to be used on some flexowriters. It would be a possible way to implement underlining on VDUs since the underline could merely set a bit in the storage location representing the next character position. The bit would be a flag to cause underlining of that position. Typewriter emulating typefaces on the V-I-P phototypesetter are designed to underline in this mode.

Switch names commencing "UF" cause spaces to be underlined. Those that start simply with "U" do not do this. The forms of underlining that print a whole underline at the one time (/UCR, /UFCR, /ULF and /UFLF) do so by evaluating the starting and terminating positions for the line and then printing, possibly overlapped, sufficient underlines to cover the distance. The other forms print one underline per character. Underlining spaces subject to expansion during justification poses special problems. The backspacing and non-spacing underlines can only print one underline per character. Consequently, the underline will be continuous only if the justifying space does not exceed the width of the underline. The other forms of underline (/UFCR and /UFLF) can calculate the width of justified spaces only for typescript. The width of these spaces when photosetting is calculated by the photosetter itself and is consequently not available to CICERO. Therefore, underlining of justified shiftable text for the photosetter should be avoided. Emphasis is usually much better achieved by the use of italic or bold script. For typesecript, underlining of spaces in justified shiftable text should be done only with the /UFCR or /UFLF forms of underlining.

The first function of an underline switch is to specify an underline delimiter. Any prior special action allotted to this character is cancelled so the delimiter specified should be unique. Then

```
./UFCR:_
```

specifies "_" as the underline delimiter for underlining with a separate underline without line advance. As is normal for switches that specify delimiters the argument is separated from the switch name with a colon.

Optionally, the underline switch may specify an underlining character. If it does not a default will be used. For the /ULF and /UFLF switches the default underlining character is a minus. For all the others it is the normal ASCII underline "_".

```
./UFLF:-:_
```

would select "-" as an underline delimiter and also make "_" the underlining character. For some printers a tilde is a useful underlining character producing a wavy underline.

The /ULF and /UFLF switches also allow the specification of the underline offset. For the photosetter the value read is that appropriate to 10 point characters and is scaled for other sizes. The default value is an off-set of 4 points which puts the EM or EN rules in approximately the right place for underlining. For typescript, the default value is the argument of the /LFLEAD switch or if that is zero then 1. Underlining in photocomposition is much less common than in typescript. Generally, the purpose of underlining is to add emphasis and this may be achieved more effectively by a change of typeface to bold or italic or a variation in point size. Nevertheless, it is possible.

```
./UFLF:-:=,6
```

could be used to select "-" as as the underline delimiter, "=" as the underlining character and specify an offset of 6 fractional vertical spacing units for the Diablo 1620 terminal. Note the use of the comma as a separator for this field. A colon cannot be used since it must be possible to specify the offset when the underline character is defaulted.

Finally, it is possible to include typeface and size delimiters to specify the font and character size for the underline line. Suppose that the /FONT switch, (see section 4.6. below) and the /SIZE switch, (see section 12.2) define [] as the font delimiters and () as the size delimiters. Then,

```
./UFLF:~:_[HR](6)
```

would define "~" as the underline delimiter for an underline type with:

An EM rule as the underline.

4/10 the current point size as the offset (by default).

HR as the typeface

and 6 as the point size for the underline.

If no point size were specified the current size would be used and the line thickness would change wherever the point size changed on the underlined line. If no typeface were specified the current face would be used. If that were italic, the V-I-P phototypesetter would for most typefaces underline with a broken line of EN rules. (EN rules do not join up, EM rules do).

It is possible to have up to 16 different underlines all operating concurrently. However, it is relatively unlikely that most people will ever need more than 2 or 3.

4.3. Substitute Space

For typescript a substitute space prints as a space of width equal to the normal character spacing. For photosetting it prints as an EN space. All numbers have width equal to an EN space so this fixed space can be used for justifying numbers. Other means are available for entering the other fixed spaces commonly used in photosetting (see chapter 12).

The immediate action marker which is converted into a substitute space is specified by the /SUB switch. For instance,

```
./SUB:*
```

specifies "*" as the substitute space marker. The various uses of substitute spaces have been described already in Chapter 2.

4.4. Case Conversion

At one time very few computer terminals or printers provided lower case characters. Consequently, it was necessary to provide some facility for case conversion in the typesetting program. This is no longer the case. However, CICERO retains the facility. The principle of operation is that all text is read from the input file in upper case and converted automatically to lower case with the following exceptions:

1. A character preceded by a single case shift operator is left in upper case.
2. A word preceded by two case shift operators is left in upper case as far as the first non-alphabetic character. A literal marker may be used to

re-instate case conversion as it counts as a non-alphabetic character but otherwise has no effect.

The scheme is fairly economic in requirement for operators and requires only one special character to be assigned.

```
./SHIFT:^
```

would define "^" as the shift operator and cause all further input to be converted to lower case unless preceded by the operator.

4.5. Continutation Character

The function of the continuation character is to allow lines to be broken in the input file but read as one by CICERO. That is, it is transmitted as a normal character unless followed by a CRLF in which case it hides the latter from CICERO. A line of output may be terminated with the continuation character if in the input a literal marker preceded it. The continuation character is specified by the /CONCAR switch. Thus,

```
./CONCAR:-
```

specifies "-" as the continuation character.

4.6. Change of Typeface

The control coding for change of typeface works through compound separators. The separator structure consists of the name of the typeface enclosed within font delimiters. These delimiters are defined through the /FONT or /GOLF switches. These two switches actually cause execution of the same code and they are really identical. Originally, the switch was called "Golf" and was intended to define the delimiters for golf ball descriptors. When the program was extended to provide photosetting capability "Font" seemed a more appropriate name. However, for compatibility, the old name has been retained as well. Then,

```
./FONT:[]
```

defines [and] as the font name delimiters. Then, for instance, in the source file one might find

```
[COURIER]In([SYMBOL]q[COURIER]r)
```

Where the "q" from the SYMBOL wheel actually prints a gamma.

Names of typefaces are at the user's discretion and may be up to 15 characters long, though obviously it is better to keep them short. When photosetting, they have to be associated with physical font locations using the /ASSIGNFONT switch (see section 12.1). However, for typed output, the name is used only to identify the font in messages. The destination of font change messages is controlled through the /GBMSG switch which can take arguments TTY, LST or NUL. For instance,

```
./GBMSG:NUL
```

suppresses all font change messages and does not wait for type wheels or golf balls to be changed.

```
./GBMSG:LST
```

puts the font name enclosed within delimiters into the listing output. That upsets the layout but one can check where the typeface is being changed. Finally,

```
./GBMSC:TTY
```

stops output when a change of typeface is required and prints a message

'Please mount SYMBOL'

on the terminal controlling the job. (Here SYMBOL is the name selected for the user to describe the wheel or golf ball required). The output printer need not be the same terminal. When the golf ball has been changed the user can re-start the output by typing a carriage return.

4.7. Literal Marker

The literal marker may be used to suppress any user assigned special action of the character following it. In detail the action is first to clear the temporary and permanent case shift flags so that if automatic case conversion was specified it will be resumed for following alphabetic characters. The start of line flag is then cancelled so that, for instance, a line that starts with a literal character followed by a dot will not be processed as a command line. Finally, if the character following the literal marker is not one of the following:

An alphabetic character requiring case conversion,
A normal character,
A backspace,
A comma,
A carriage return,
A form feed or page mark,
A line feed,
A horizontal tab,
A vertical tab

it will be processed by the normal character code. Otherwise, the literal marker has no further effect and the character is processed by the code appropriate to it.

The literal marker is specified by using the /LITERAL switch. Thus,

```
./LIT:&
```

specifies "&" as the literal marker.

4.8. Terminating a Special Action - /KILL switch

The /KILL switch terminates the special action associated with a specially defined character and restores it to being a normal character. For instance,

```
./KILL:#
```

would cause "#" to be treated as a normal character and not as the shiftable text delimiter which it was defined to be by the switch /STD of section 4.1.

The /KILL switch will fail if the function assigned to the character was not one in the user definable set.

4.9. Ignoring a Special Character - /FORGET switch

The /FORGET switch is useful if a function must be encoded say for typescript but is to be ignored when photosetting. For instance, when photosetting the underline delimiter might be ignored by specifying,

```
./FORGET:_
```

in the appropriate default file. This switch is intended to nullify a single character used as a local control. The switch may be used repeatedly to nullify a number of such local control characters. That is there is no

single "FORGET" character. For instance,

```
./FORGET:_ /FORGET:~ /FORGET:%
```

would cause the local control characters "_", "~" and "%" to be ignored.

4.10. Ignoring a Compound Delimiter - /NULDELIM switch

Some local controls are necessarily compound controls enclosed within delimiters. For instance, changes of typeface are encoded by enclosing the typeface name within delimiters. (See section 4.6). It may be important to ignore such controls when setting typescript. The /NULDELIMIT switch allows this. Thus,

```
./NULD:[]
```

causes all strings delimited by [] to be ignored.

4.11. System Defined Special Characters

The nine characters mentioned at the start of this chapter have special action always. For reference, this action is defined here:

BS - backspace. The program scans back down the line keeping a running total of the character widths over which it has moved and stopping when the total first becomes positive. It then generates fixed width backspaces to backspace by that amount. This procedure allows multiple backspacing over characters of differing widths. It also means that characters of zero width like changes of point size do not swallow backspaces.

HT - horizontal tab. In free text a horizontal tab generates fixed spaces to move to the next uniformly spaced tabulation column. The spacing of these columns is set by the /TABSIZ switch. In shiftable text a tab at the start of a line of input signifies a new paragraph. The control of new paragraphs is described in more detail in the next chapter.

LF - line feed. Normally CICERO expects line feeds to be preceded by carriage returns. However, multiple line feeds after a carriage return will result in corresponding succession of blank lines in the output. Line feeds elsewhere will be transmitted to the output.

VT - vertical tab. Is used to force the start of a new column. (See chapter 6).

FF - form feed. Provided the present page is not empty a form feed will force the start of a new one under normal circumstances. Normal circumstances means that this action has not been turned off. Because line oriented editors tend to run out of line numbers if they are not allowed to generate new pages from time to time, two Boolean switches have been introduced,

/NOFF - causes FFs to be ignored.

/FF - re-enables FFs.

Clearly, there needs to be some means of forcing a new page when form feeds are being ignored, so a predefined macro <FF> is available to force new pages (see chapter 10).

CR - carriage return. In all processing modes other than shiftable text a carriage return causes the present line to be output. In shiftable text, single carriage returns are treated simply as word breaks. They

result in a space or end of line in the output and the following line feed is ignored. Double carriage return line feed combinations, CRLFs, cause a start at the left margin (cancelling any ruling indentation) after outputting a blank line of spacing equal to the height specified by the /BLANKSP switch. This action occurs also for the first blank line of free text after leaving shiftable text.

SP - space. Spaces at the start of a line are counted and used to set the indentation of the next line to be output. In free text, that is the current line so the effect is the same as if the spaces were transmitted directly to the output. In shiftable text, the value is used as the indentation for all subsequent lines until a line is forced out by a double CRLF or the <ZM> predefined macro is issued or the processing mode leaves shiftable text. Note the indentation persists across paragraphs. Because this means of controlling indentation is open to accidental misuse some implementations of CICERO may have it deleted.

The action for multiple spaces not at the start of a line also depends on the processing mode. In shiftable text, they are merged. In free text, they are transmitted faithfully.

, - comma. A comma at the start of a line has special significance for the program TYP2, a predecessor of CICERO. For compatability the A.R.L. implementation of CICERO will process files designed for TYP2 and for this reason a leading comma has an unpublished special action. If it is necessary to start a line with a comma, this special action should be suppressed by preceding the comma with a literal marker.

. - dot. A dot at the start of a line causes the line to be treated as a command string. Within a line, it will be interpreted as a full stop (period) if it is followed by a space and not preceded by a literal marker. The space following such a full stop may be expanded by the addition of a thin space in some implementations. That is intended as a compromise between French spacing (a double space) and the modern practice of no extra space. On some systems, the code may have been modified to result in the latter.

CHAPTER 5

Facilities Available in Shiftable Text

All the facilities described in this section are available to the user of the basic subset of commands. Consequently, for most of them, the local controls associated with their use have already been described in Chapter 2. The purpose of this chapter is, therefore, to present the associated global controls. Unfortunately, the approrpriate form for these does depend on the printing device for which the output is intended. Although there will be separate chapters later in the manual to describe facilities for photosetting and for fractionally spacing printers like the Diablo 1620, it seems sensible to group together all variants of these global controls so that this chapter becomes a better reference source.

5.1. Line Length

Line length excludes any fixed left margin but is taken over local left margin indentation so the right margin and line length remain invariant despite changes in such indentations. Line length is controlled by the /LINE switch entered in a command string, most probably in the input file. The simplest form of this switch applies for all devices with uniform character spacing. For these, line lengths may be specified in characters. Then,

```
./LINE:56
```

sets a line length of 56 characters for shiftable, right or centre text. For a photosetter or any device with proportional spacing, character counts can no longer be used as a measure of width since character widths vary. *CICERO assumes that all horizontal measures for photosetting* will be specified in units of picas and points. A dot is used to separate the pica from the point field. Units of measurement for typesetting are described in chapter 12. Basically, 12 points make a pica and a pica is approximately 1/6 th of an inch. If the line length is a whole number of picas the dot may be omitted. Thus,

```
./LINE:21
```

sets a line length of 21 picas whilst

```
./LINE:12.6
```

sets a line length of 12 picas 6 points.

For programs that produce a listing of the photosetter output, it is necessary to be able to specify separate line lengths for the setter and listing device. Then

```
./LINE:23,72
```

specifies a line length of 23 picas for the photosetter and 72 characters for the listing device. The division into lines is made on the basis of the photosetter line length but justification and column width in the listing file are determined as far as possible by the width in characters specified for the listing device.

In shiftable text changes in line length are effective after the line currently being filled has been output. Reducing the line length for the current line would cause problems if the line was already overset for the new measure. In the other text processing modes, it is not possible to

change line length mid line. The general rule is, therefore, that the change is effective on the next line.

5.2. Line Spacing

Vertical line spacing is controlled by the /SPACING switch in a command string. For instance,

```
./SPACING:1
```

sets single line spacing for printing devices capable only of uniform vertical line spacing. Some printers, for instance the Diablo 1620 terminal, can space vertically in units of 1/48th of an inch. If the /LFLEAD switch (see chapter 11) has been specified with an argument bigger than 1 then the argument of the /SPACING switch is taken to be in the appropriate vertical fractional units. Then

```
./SPACING:8
```

will set a line spacing of 6 lines to the inch.

For photosetter output, the unit of vertical spacing is a point. A dot may be used to specify tenths. Thus,

```
./SPACING:10.5
```

will set a vertical spacing of 10-1/2 points. Users should be warned that relatively few photosetters are capable of vertical spacing with increments smaller than 1/2 points.

The /SPACING switch is effective on the next line to be output as the decision whether it fits on the page cannot be made until the line has been assembled and it can be ascertained whether there are underlines or sub or superscripts. In shiftable text this means the switch will affect the spacing from the previous line of the line currently being assembled.

In the presence of sub and superscripts or underlining, the line spacing is defined as the separation of the un-offset base lines. CICERO will increase the line spacing to avoid overprinting in some instances. This happens if the line spacing is bigger than the offset of the top of the highest superscript but less than the sum of that and the lowest base line offset of the previous line. This condition covers most accidental cases but allows mathematics to be set with different symbols on very closely spaced base lines. Some people find this degree of protection against overprinting unnecessarily conservative. They are happy, for instance, to allow superscripts to protrude into the previous line especially, if it was blank. The algorithm to print the page has to process lines in sequence. However, so long as the base lines from different lines do not intermingle this condition can be met whilst still allowing some merging of superscripts. This lower level of protection is selected by specifying the /CLASHES switch which causes the character height of superscripts to be ignored and /NOCLASHES which restores the normal protective mode.

5.3. New Paragraphs

New paragraphs are signalled by a tab at the start of a line of input. The action taken is then to output a blank line and indent the start of the new paragraph. Both the spacing of the blank line and the indentation of the start of the new paragraph are globally controlled by switches as follows:

```
./PARASP:1
```

sets the inter paragraph spacing to 1 blank line (the default value) while

```
./PARINDENT:3
```

sets the paragraph indentation to 3 substitute spaces (also the default value).

For terminals with fractional vertical spacing (specified by setting the /LFLEAD switch argument greater than 1), the argument of the /PARASP switch is taken to be in fractional vertical units (normally 1/48ths of an inch). For instance,

```
./PARASP:4
```

would set an interparagraph spacing of 4/48 inches. Compared to say 8/48 inch line spacing this represents a significant saving in space whilst still giving the impression of a blank line between paragraphs. The argument of the /PARINDENT switch is still interpreted as substitute space characters.

For a photosetter, the argument of the /PARASP switch is in points, the same as for the /SPACING switch. The actual inter paragraph spacing depends on the point size of the characters being set. The value supplied by the switch is the spacing for 10 point and is scaled for other sizes. Thus if the inter paragraph spacing is specified by

```
./PARASP:4
```

and text is being set in 8 point characters, the actual inter paragraph spacing will be 8*4/10 = 3 points. Also when photosetting a new paragraph immediately after a heading set in free text or forced to the left margin by a double CRLF, the inter paragraph spacing will be set at half the specified value. This ensures that headings appear tied to the text to which they relate.

When photosetting, the original intention was to specify paragraph indentation with the /PARINDENT switch already described. However, an indentation of 1 EM space is commonly used. Normally, substitute spaces translate to EN spaces when photosetting. Unfortunately, the width of an EN space is liable to vary with typeface, since it is the width of the numerals and two EN spaces do not necessarily add up to an EM space. To accommodate these variations the specification of the /PARINDENT switch is changed for photosetting and now sets the indentation in units of 1/2 EM spaces. Further, three other switches have been introduced. The /EMPARAGRAPH switch, cancelled by the /NOEMPARAG switch, causes the indentation to adjust automatically to the line measure depending on some rule. The rule implemented as a default happens to be that recommended for Australian Government publications and is,

1 EM space for measures up to 26 picas,
1-1/2 EM spaces for measures between 27 and 35 picas
2 EM spaces for lines longer than that.

The break points are determined by the /EMLIMITS switch which takes two arguments. The first, is the maximum line length in picas for EM indentation of new paragraphs, the second is the maximum line length for 1-1/2 EM indentation. The default setting of this switch is /EMLIMITS:26,35. Whilst, this description may seem complicated, what it really amounts to is that specification of the /EMPARA switch by itself will result in an appropriate indentation for the first line of new paragraphs regardless of the line measure.

When setting in multiple columns, CICERO will attempt to even up the base lines of the last lines on each column by adding to the inter paragraph

spaces in the shorter column. This happens automatically. However, a user may specify a maximum inter paragraph spacing with the /MAXADDLEAD switch. A value of zero will allow some adjustment of columns by this means so long as the imbalance between columns can be compensated without requiring a uniform adjustment to all interparagraph spaces. (see section 11.8). The arguments of the /MAXADDLEAD switch are the same as those for the /SPACING and /PARASP switches. For the photosetter they are absolute and not scaled by the prevalent character size.

5.4. Headings

A double CRLF in shiftable text causes the output of a blank line and the next line starts at the left margin. Any automatic indentation is also cancelled. Exactly the same effect can be achieved by entering free text to output the blank line and heading. Either way the spacing of the blank line is determined by the /BLANKSP switch which takes the same arguments as the /SPACING and /PARASP switches. Thus for devices with uniform line spacing,

```
./BLANKSP:1
```

sets up the default spacing of one blank line. For the Diablo 1620 terminal

```
./BLANKSP:6
```

sets up a spacing of 6/48 inches before headings whilst for a photosetter,

```
./BLANKSP:6
```

sets up a spacing of 6 points before a heading set in 10 point characters. If the character size is different, the actual spacing is scaled accordingly. Note also that it is the character size ruling when the double CRLF or blank line at the start of free text is read that determines the blank space. However, if character size increases for the heading there will be a further increase in base line spacing so as to ensure that the blank space specified is preserved.

If more CRLFs are read the additional spacing is in units of the current line spacing specified by the /SPACING switch.

The <NOJUST> predefined macro described in section 10.5.8 may be used to suspend justification temporarily for a heading that extends over more than one line. Justification is restored, if it was in force, when the last line of the heading is forced out quadded left.

5.5. Justification

Justification is the process of evening up the right margin by expanding the inter word spaces. There is a theory that justified text is easier to read because the action of finding the start of each new line from the end of the previous one is constant. However, fashion seems also to be a major motivator and at the present time justified text is out of fashion for some types of publication. CICERO caters for all tastes and justification in shiftable text can be turned on and off with the /JUSTIFY and /NOJUSTIFY switches.

5.6. Setting the Spaceband Parameters

When setting shiftable text the inter word spaces or spacebands can be appreciably less than the mean character width without the word division becoming indistinct. For devices that can fractionally space (specified by setting the /FSPERS switch with argument greater than 1) and for photosetting, CICERO allows the minimum space to be set with the /SPBMIN switch. Thus for a Diablo 1620 terminal,

```
./SPBMIN:8
```

sets a minimum spaceband of 8/120 inches, whilst for the photosetter,

```
./SPBMIN:4
```

sets a minimum spaceband of 4 units. Units of horizontal measure for the Diablo 1620 are explained in chapter 11 and for photosetting in chapter 13.

Justification is performed by expanding the spaces on the line starting from the minimum spaceband width and stopping when the rightmost edge of the last character aligns with the right margin. A few lines will fit exactly and will, therefore, have spaces of the minimum spaceband width.

For lines that are quadded, like the last line of a paragraph or headings, a mean value is used unless the line would then be overset. This value may be specified with the /SPBMEAN switch. For instance, for the Diablo 1620 terminal printing with a 10-pitch wheel one would specify,

```
./SPBMEAN:12
```

to set a mean spaceband of 12/120 inches. Whilst for a photosetter,

```
./SPBMEAN:8
```

would set a mean spaceband of 8 units.

5.7. Hyphenation

The amount of space that must be distributed amongst the interword gaps to justify a line depends on the location of the end of line within the first word that will not fit on the line. i.e. it is the width of the characters that would have fitted on the line. Clearly the average value depends on the vocabulary used. However, the size of the justifying spaces is that amount divided by the number of spaces on a line. That value depends on the ratio of line length to point size. Newspapers, traditionally use a very short line length and consequently experience severe hyphenation problems. It is very important for them to reduce the amount of blank space on lines. Therefore, they must hyphenate wherever possible. Technical and scientific publications, for which CICERO is designed, tend to use much longer line lengths so the hyphenation requirement is much less acute. On the other hand, they probably also use longer words.

The A.R.L. implementation of CICERO uses a very simple but reliable hyphenator. It will only hyphenate at an existing hyphen, after an "x" or between certain unpronouncable pairs of consonants. It also requires a vowel in each part of the word. It has a very good chance of hyphenating long words that nearly fit on a line. However, it certainly does not hyphenate everywhere that a hyphen would be acceptable. Furthermore, it will not hyphenate the last word of a paragraph, column or page and if the word already contains an existing hyphen, it will only hyphenate there. A word that contains a substitute space will not be hyphenated as the point of using the substitute space is to prevent separation across lines. It is possible sometimes for it to produce a hyphen trailing across columns but this only happens if lines are moved between columns so as to even them up on the last page or when the start of a new page is forced.

Hyphenation is turned on by the /HYPHEN switch and off by the /NOHYPHEN switch. These switches are effective when the next line break decision is made in shiftable text.

Hyphenation is also turned on and off on a temporary basis by the predefined macros <HYPHEN> and <NOHYPHEN> and by the underline delimiters. An underline delimiter that starts underlining or the <NOHYPHEN> predefined macro supress hyphenation. The terminating delimiter or the predefined macro <HYPHEN> restore hyphenation after the next word break and if the /HYPHEN switch was on. These facilities are intended to allow headings to be set in shiftable text without their being hyphenated. In particular, when photosetting, the delimiter used for underlining control for typescript can be used as a special macro character (see section 10.6) and the macros called can themselves turn hyphenation on and off through the predefined macros as well as changing typeface or pointsize.

CHAPTER 6

Pagination

CICERO provides totally automatic page division or pagination for a fairly wide range of formats suitable for books and journals. In its present form it would not be suitable for newspapers. The purpose of the present chapter is to describe the controls directly associated with pagination. The following three chapters cover the related topics of figure placement, footnotes and indexes.

Pagination, the process of division into pages, is fundamental to the operation of CICERO and no facilities are provided for producing unpaged output.

6.1. Page Length

The logical page length for output is controlled by the /PAGE switch. For devices with uniform character size the argument is the page height in single spaced lines. Thus,

```
./PAGE:58
```

sets a page length of 58 lines. In the case of fractionally spacing printers like the Diablo 1620, this height is multiplied by the argument of the /LFLEAD switch to obtain the value in vertical fractional line feeds. However, the argument is still specified in lines. Page height does not depend on the /SPACING switch though obviously, the number of lines that actually fit on the page will.

Since the character size can vary, when photosetting a different means must be used to specify page height. CICERO assumes that large vertical dimensions, such as page length and the height of blank figures, will be specified in units of picas and points. As with the /LINE switch the pica field is separated from the point field by a dot which may be omitted if the value is a whole number of picas. Thus

```
./PAGE:54
```

for the photosetter will set a 54 pica page length (about 9 inches) whilst

```
./PAGE:36.6
```

would set a 36 pica 6 point page length.

When using fan folded paper, the physical page size will be determined by the folds in the paper. However, the logical page size is likely to be determined by the application. If the output device has hardware form feed, the starts of logical and physical pages may be aligned. If this is not provided, it is useful to be able to set a separate physical page length to achieve this. The /RPAGE switch does this. For instance,

```
./RPAGE:58
```

sets a physical page length of 58 lines. The arguments of the /RPAGE and /PAGE switches have the same forms and significances. However, the /RPAGE switch is unlikely to be used when photosetting.

6.2. Margins

A constant left margin may be specified with the /MARGIN switch. Thus

```
./MARGIN:8
```

sets a left margin of 8 characters width for a typescript device. The /MARGIN switch is effective when the next page text block is output. Except when changing the numbers of columns or commencing tabulation, this does not occur until the page is turned. Thus the /MARGIN switch is not a good way to control local indentation of the left margin. This is much more readily done with the <LM> predefined macro or by counting leading spaces as described in Chapter 2.

Sometimes it is necessary to have different sized left margins on odd and even numbered pages. For instance, with typescript output one might wish to do this in order to facilitate binding. Then,

```
./MARGIN:12,4
```

sets a left margin of 12 spaces on odd numbered pages and 4 on even pages. The default action when no separate argument is specified for even pages is to assign it the value for odd pages.

For photosetting, the margin size is specified in picas and points. For instance,

```
./MARGIN:2.8
```

would set a left margin of 2 picas 8 points. The use of this means to set a uniform left margin throughout a publication would be unusual, since the photoset area is kept as small as possible and generally covers only the printed area of the final copy. Margin sizes can be left to the platemaker. However, the switch might be used when setting a small amount of text in a single column at the start of a publication printed predominantly in two columns. Abstracts are often set this way.

6.3. Columns

When setting in multiple columns each column starting with the leftmost is filled in turn with text from the input source. A start at a new column may be forced by including a vertical tab in the input. A form feed will force the start of a new page. When a page is forced out in this way and also for the last page, the columns of text on the page are evened up by redistributing the surplus space. As was mentioned in the last chapter, this process can lead to a hyphen trailing between columns. Otherwise, text that must be kept contiguous is not broken up.

The number of columns to be set per page is specified with the /COLUMN switch. Thus,

```
./COLUMN:2
```

specifies a double column format. Changes of column format happen immediately upon exit from the command decoder.

The maximum number of columns specifiable is likely to be implementation dependent. That at A.R.L. allows 10 though the authors have never had cause to use more than three.

The intercolumn margin or gutter may be set with the /GUTTER switch. For instance,

```
./GUTTER:5
```

would set an intercolumn margin of 5 spaces. Whilst for photosetting,

```
./GUTTER:1.6
```

would set a gutter of 1 pica 6 points.

6.4. Tabulated Columns

Sometimes, it is necessary to set formatted data into multiple columns, text in some of the columns being justified and entries being related horizontally as well as vertically. CICERO allows this. The /TABULATE switch is used to specify individual column and gutter widths. Thus,

```
./TABULATE:44,4,44,4,44
```

would set up 3 columns each of 44 characters width and with equal gutters of 4 spaces. Each of the columns and gutters could have had different widths. Unlike the /COLUMN switch, this switch does not start tabulation. That action is controlled by four predefined macros. Predefined macros are described in Chapter 10. Essentially, their function is to act as special local controls when the frequency of occurrence of the control does not justify the specification of special characters. The four are:

<TON> to turn tabulation on.

<TOFF> to turn it off.

<TNEXT> to advance to the next column.

<TSTART> to start again at the left margin.

In specifying the above, it has been assumed that "<" and ">" are the macro delimiters.

The method of use is as follows. For each entry in the table the whole message for each column is entered and terminated with a <TNEXT> except for the last column where <TSTART> is used. The text processing mode is continuous between columns. Each entry is assembled separately before it is added to the page and contiguity is preserved. Thus if any column is too long to fit the entry is transferred to the next page and the previous page justified. Error messages will result if any entry exceeds the page length or if the number of <TNEXT> calls exceeds the number of gutters specified in the /TABULATE switch. Figures 10.2 and 10.3 illustrate tabulated setting using this facility.

The vertical spacing between entries may be set with the /TSPACING switch. For instance,

```
./TSPACING:6
```

would set a spacing of 6 fractional vertical line feeds when output is destined for the Diablo 1620 terminal.

Tabulated columns are an alternative format to the normal multiple column output. Mixing of the two formats is not at present permitted.

6.5. Page Headings and Footings

Besides the main text, the formats of most publications also require headings and footings on each page. Since the formatting requirements are potentially the same, the class exemplified by headings and footings will be called page titles in this manual. When formatting page titles it is necessary to allow:

1. The page number to be included or excluded.

2. The page title to be changed.

3. The page title to be different on even and odd numbered pages.

```
./HEADING:$Page :,:
:,:Page :,:
:
```

Produces headings on odd numbered pages

Page #

and on even pages,

Page #

Whilst

```
./Footing:@&*&*:,:&*&*:
```

produces a centred footing

#

Where # stands for the page number.

Fig. 6.1. Some examples of page headings and the source code to produce them.

4. Format control within page titles (so that titles can be quadded right, left or centred - even possibly justified).

5. No limit on the length or number of lines in the title (except that obviously it must not exceed the page length).

6. Changes in typeface or point size within page titles.

Provided the space required by the page title can be evaluated before the main text block is set, none of these is difficult to achieve. This requirement can be met without allocating a fixed space to the title simply by making the title effective on the next page to be started rather than the current page. It also implies that the space occupied by the title does not depend critically on page numbers. For quadded text this will be so.

CICERO implements page titles as follows. Page titles are declared in switches in command strings probably contained within the input file. Three such switches are provided:

1. /HEADING to set the page heading at the top of the page.

2. /FOOTING to set the page footing which is at the bottom of the page.

3. /XFOOTING to set an extra footing between the bottom of the text and the page reference line. Page reference lines are described in Chapter 9.

Any of these switches can take up to four arguments. The arguments are strings. The first two specify the title on odd numbered pages, the second

pair that for even numbered pages. If the titles on even pages is the same as that on odd pages the second pair may be omitted. Within each pair of arguments, the first string is the source code for the part of the title before the page number, the second is the part after it. If the second string is omitted that page title does not contain the page number. The first character after the switch name is used both to delimit the switch and the first string. It must be non-alphanumeric and cannot be a comma. Commas are used to separate the arguments. The first character of each string is taken as the string delimiter except that spaces are ignored. Commas cannot be used as string delimiters so that null strings may be recognised. Four commas after the switch name will clear all the title strings for that switch.

In order to evaluate the space that will be occupied, title strings are processed at the start of the first new page after the switch declaration. However, the title is stored as source code so that it can be reprocessed for each new page number. Whether a particular page will be even or odd cannot be determined at the time the text for the page is set if the page follows a contents or index as these will modify the page number. Consequently, CICERO will always remember the bigger value.

For page titles, the left margin is that prevailing when the switch was specified and the line length is the distance across all columns and gutters at the same time. The line spacing will be that prevailing at the time the switch was declared.

Some examples of page titles and the source code required to produce them are shown in figure 6.1.

6.6. Contiguity

Good pagination requires some constructions, for instance headings and new paragraphs, to be prevented from occuring at the bottom of a page. The avoidance of these conditions implies the recognition of contiguity. For the two conditions mentioned, the termination of the contiguous text is not linked to any particular event in the input code. Rather, it is a constraint on the output. A simple way to implement it is to specify a necessary number of contiguous lines that should be capable of being fitted on the page whenever the construction occurs. If that number is not available, the column is justified and the construction moved to the next column or page. Fairly obviously there needs to be a hierarchy of contiguity requirements. For instance, if there is a heading followed by a new paragraph, the contiguity requirement for headings must be sufficient to ensure at the time the heading is processed that there will be room for a following new paragraph. Otherwise the new paragraph will be shifted to the next page leaving the heading at the bottom of the previous page. The contiguity requirement for these constructions must also be remembered with the stored lines since the ordering of lines on a page is subject to modification if the columns have to be evened up when the page is forced out half full (eg. for the last page). Contiguity requirements must be observed during this process of adjusting the columns.

Default values for the contiguity requirements on entry to centre text, free text (and lines forced out with a double CRLF in shiftable text) and new paragraphs may be set with the /CT CNL, /FT CNL and /TAB CNL switches. For instance,

```
./CT CNL:5 /FT CNL:3 /TAB CNL:3
```

would set up the default values. Note that the units are always lines even for photosetting. This is because the contiguity requirement is necessarily expressed in lines.

Turning underlining on at the start of a line increases the contiguity requirement for that line by 2 lines. Thus an underlined heading has precedence over both a new paragraph and the start of free text. When photosetting, the predefined macro <UPGNL1> has the same effect at the start of a line. More generally, the predefined macro <UPGNL> increases the contiguity requirement for the line by one line per call regardless of its location on the line. These macros are described in section 10.5.8.

There is often the need to specify larger contiguity requirements for particular constructions, for instance, short tables referenced in line and mathematical expressions. In these cases both the start and finish of the contiguous chunk may be identified in the input text. Consequently, the text could be delimited with say "contiguous text delimiters" and the whole piece set into lines before the decision where to put it is made. However, this solution is unnecessarily complicated. It is unlikely that such tables or mathematical expressions will be set in shiftable text. Consequently, the numbers of lines they will occupy can be counted in advance. Then the /GNL switch may be used to specify a contiguous block of sufficient size. For instance,

```
./GNL:10
```

would ensure that the next 10 lines do not get split across a page.

Finally, there is one construction where the end of the contiguous region is easily identified but not the start. The last line of a paragraph should not occur at the start of a new column or page. If it does the construction is called a widow line. Besides, the location of the line, a widow is characterised by the following:

1. The text processing mode is shiftable text.

2. The line is quadded left - not filled and justified.

3. The previous line was not quadded.

When the columns are evened up on pages that are forced out the redistribution of lines might create a widow line. Consequently, contiguity conditions must be set for the previous line for all potential widow lines so as to prevent this happening.

If the previous column or page was correctly set there should be no spare room on it. Consequently, the right solution to the widow line problem is not to squeeze it into the previous page or column but instead to provide it with company by transferring over the last line of the previous column. Even when a new page is being started, this solution is possible since the widow line is always the first not to fit on the page. Therefore, it can be recognised before the previous page is disposed of. However, this means of solution may be blocked by a contiguity requirement. For instance, for the default values specified above, a paragraph of 4 lines with the last a widow could not be set without either violating the contiguity requirement for a new paragraph or retaining the widow line. The A.R.L. implementation of CICERO leaves the widow line in this case and prints a warning message. The user may solve the problem by making the whole paragraph contiguous with the /GNL switch at the start. In practice, widow lines occur much less frequently than a cursory analysis would suggest. This is because contiguity requirements cause headings and new paragraphs to migrate to the tops of pages. Consequently, a solution to the widow line problem which works in most cases but requires intervention from the user very occasionally is really quite adequate.

On a time-sharing system one might reasonably allow the program to ask the user what should be done if the widow fix up fails. For instance, the line could be transferred regardless of contiguity or the previous page could be set long. This has not been implemented yet. An obvious prerequesite is that the program must be able to check whether it is running as a batch job or if there is a real user at the terminal. If the former, there is no point in asking questions.

6.7. Page Numbers

As described in section 6.5, page numbers may be part of a heading or footing. However, CICERO also provides a simple mechanism for generating a page number line when the page number is not included in a heading or footing. This facility is controlled by the /PNO and /NOPNO switches. Then,

```
./PNO
```

turns the printing of page numbers on, whilst,

```
./NOPNO
```

turns it off. The /PNO switch may also be used to specify a spacing for the page number line, a typeface and for photosetting, a typesize. Then, if () are the size delimiters, (see section 12.2) and [] are the font name delimiters (see section 4.6), then,

```
./PNO,11(10)[HR]
```

specifies a line spacing of 11 points, a type size of 10 points and a typeface "HR" for the page number line. When no values are specified, the previous settings are used.

The logical page number may be set with the /NUMBER switch. The value supplied as argument is the new logical page number of the current page. It is not used for that page but is incremented by one and applied to the next page when that page is started. Then for instance,

```
./NUMBER:0
```

results in the next page being numbered one.

Numbering may be in Arabic or Roman numerals depending on the setting of the switches /ARABIC or /ROMAN. Arabic numbering is the default and

```
./ROMAN
```

selects Roman numbering up to 4000.

Normally, the page number is centred and enclosed within square brackets. The delimiters for the page number may be changed using the /PNODELIMIT switch. Thus,

```
./PNODEL:--
```

sets delimiters "-" about the page number. As many photosetting typefaces do not contain square brackets this is a common requirement when photosetting. Note the /PNODEL switch will accept spaces as arguments. When a more complicated numbering system is required the number should be enclosed within the page heading or a footing.

Finally, the page number line may occur at the top or the bottom of the page. This is determined by the /PNOTOP and /PNOBOTTOM switches.

6.8. Chapter Descriptors

Wherever page numbers occur, including references within indexes, CICERO allows them to be preceded by an optional chapter descriptor. The purpose of this facility is to permit chapters to be numbered independently. This is a useful capability when setting manuals that will be updated frequently and distributed in loose-leaf form. It allows chapters to be extended without the need to renumber all subsequent pages.

The chapter descriptor may be specified as any of the following forms:

1. A macro name generating a string of characters. For instance, "Appendix A-3". Since the numeric counter option always assumes Arabic numbering, this form may be used to generate Roman numerals if these are required for chapter numbers.

2. A numeric counter, eg "6-13" for page 13 of chapter 6.

3. An alphabetic counter, eg "A-23" for page 23 of part A.

Control of the chapter descriptor is accomplished by a single predefined macro NCHAP which takes as parameter the symbolic name of the macro or counter to be used for subsequent pages. Thus

```
<NCHAP(NAME)>
```

defines <NAME> as the symbolic object to be used to generate the chapter descriptor for subsequent pages. The full action of NCHAP is as follows:

1. Force out pages until there are no figures or footnotes that have not been set.

2. If NAME is a macro check that it has no parameter list. If it has print a warning. The name will be used but there will be no actual formal substitution whenever the chapter descriptor is referenced.

3. If NAME is a counter autoincrement it.

4. Set the page number to 1.

5. Write the chapter descriptor and page number to the new page.

NAME must not be a named tabstop, a predefined macro nor undefined. If it is an error will be given and NCHAP will behave as an FF predefined macro.

6.9. Control of Extra Blank Lines on the Page

The normal page format allows two extra blank lines on the page, one before the first line of the main text (and after the heading or page number) and one at the bottom after the last line of text and before the footings and page reference line. The spacing of these two blank lines can be defined using the /XSP switch. Thus,

```
./XSP:5
```

would set a spacing of 5 vertical units for the Diablo printer (for instance) for these two lines. If the /XSP switch is not used the spacing will be defaulted to that current when the first line is output. A value of zero is acceptable and will not result in a default setting.

No space will normally be reserved for a page reference line unless this is specified with the /PRD switch (see section 9.3).

6.10. Forcing the Start of a New Column or New Page

Very often it is necessary to position some text at the top of a new page. Chapter headings are an obvious example. Occasionally, it is sufficient for the construction to appear at the start of a new column. In the original design of CICERO these two requirements were controlled by the occurrence of a form feed (FF) and vertical tab (VT) respectively. The use of a form feed for this purpose was obviously justified and as there appeared to be little legitimate requirement for a vertical tab that character could reasonably be assigned for this purpose. Unfortunately, form feeds have significance for text editors. In particular, line oriented editors use form feeds as new page marks when they start a new sequence of line numbers. If the numbering sequence is sparse so as to allow for easy additions they need to start new pages fairly frequently in order to avoid running out of line numbers. Therefore, CICERO now has a /NOFF switch which causes form feeds to be ignored. Whilst they can be re-enabled by the /FF switch the predefined macro <FF> provides a means to cause a new page regardless of the setting of the /NOFF switch. This predefined macro, along with other macros to generate or simulate special characters is described later in section 10.5.1. People who prefer to use line oriented editors should specify the /NOFF switch and force new pages with this predefined macro.

CHAPTER 7

Figures

Some output devices, for instance the better ones of those that provide direct computer output on microfiche, are capable of drawing illustrations as well as setting text. The figure facilities in CICERO are not for this purpose. At present, CICERO provides no facilities for graphics output. It is envisaged that most illustrations will be prepared separately and added during a paste-up stage. Consequently, the figure facilities described in this chapter have three purposes:

1. The reservation of space in the page format for illustrations.

2. The setting of text, for instance a table, that has to be positioned close to the point of reference to it in the main text but which is not read in the main stream of the text.

3. The provision of captions for illustrations.

6.1. The /FIGURE switch

The /FIGURE switch reserves space only. Using this switch a caption would have to be provided on the figure itself when it is added at the paste-up stage. The /FIGURE switch may take two arguments. The first is the height of the figure, the second, the width in columns. This second argument may be omitted, in which case a value of 1 is assumed. The restriction of figure widths to multiples of column width not only simplifies the typesetting problem by avoiding short lines at the edge of the figure, it is also in line with modern practice. For typescript output,

```
./FIG:33
```

would reserve space for a figure 33 lines in height and 1 column in width. Whereas,

```
./FIG:27,2
```

would reserve space across 2 columns and with height 27 lines. When photosetting, the first argument is interpreted in picas and if it contains a dot, points. Thus,

```
./FIC:25.6,2
```

would reserve space for a figure of height 25 picas 6 points and crossing two printing columns.

Note that in the above figure definitions because of the definition of a /FIGDELIMIT switch (see the next section), "/FIC" is not a unique abbreviation for "/FIGURE". However, CICERO allows it, actually by defining another switch /FIG which has the same action.

Various options are possible for the location of figures. Graphic designers generally prefer to position figures at the top or bottom of the page so as not to divide the text. For a particular class of publication the compositor may be required to position figures at the top always, at the bottom always or at the bottom if they fit on the page and at the top of the next page if they do not. On the other hand when positioning tabular data it is generally preferable to set tables in line if they will fit and send them to the top or bottom of the next page only if they do not. These various options are controlled by two switches:

/FIGFIT:arg determines where to put the figure if there is space for it on the current page.

/FIGCARRY:arg determines where to put the figure if it is carried over to the next page.

The arguments for the two switches are the same but have slightly different effect:

TOP causes the figure to be positioned at the top of the page or if there are figures there already, below the last of them. In multiple column setting CICERO checks to see that the figure does not extend further right than any existing "top" figures. This is to ensure that short areas of discontiguous text are not created. If the test fails, the figure is not regarded as fitting.

FLOAT when given as the argument of the /FIGFIT switch, it specifies that the figure should be positioned where it is declared if it fits. Otherwise, it is moved to the first place where it will fit. This is the option appropriate to tables and corresponds to the former implementation before these switches were added. When given as an argument of the /FIGCARRY switch, its effect is similar to the TOP argument except that no check is made to see that the figure does not extend beyond the right edge of any existing top figures. FLOAT figures can cause discontiguous text even when they are transferred to the next column or page.

SINK causes the figures to sink to the bottom of the page, in so doing it elevates any existing "SINK" figures. If the figure is not at least as wide as those it would elevate it is regarded as not fitting. If a sink figure extends into the present column from a column to the left it cannot be moved but the new sink figure is allowed to be positioned above it. A check is made to see that the top of the highest figure elevated does not clash with any figures already on the page or even the bottom of any existing text.

The two switches set up values which are examined every time a new figure is created. That is, at the termination of a piece of figure text or after a /FIGURE or /FIG switch specification.

CICERO will try to set the figures in the order specified and if a figure will not fit in the required position, other figures will not be set until that figure has been set. However, by changing the arguments of the /FIGFIT or /FIGCARRY switches a user can arrange that a subsequent figure appears at the top of a page ahead of an earlier "sink" figure.

Care has to be taken in shiftable text when positioning "float" figures between paragraphs. The obvious approach,

```
"
  this is the end of the previous paragraph.
  ./FIG:20
        This is the start of the next paragraph.
"
```

will usually lead to the figure being inserted before the last line of the paragraph effectively producing a widow line. The reason for this is not difficult to see. When the /FIGURE switch is encountered in the above example the last line of the previous paragraph has not yet been output. It will not be output until the tab identifying the start of the new paragraph has been read. Hence the figure precedes the last line of the paragraph if

there is room. The figure may be positioned between paragraphs by the following procedure:

```
"
  this is the end of the previous paragraph.
        This is
  ./FIC:20
  the start of the next paragraph.
"
```

In this case the /FIGURE switch is encountered after the previous paragraph has been forced out. The short line at the start of the new paragraph does not matter in shiftable text as the line structure of the input source file is ignored in this processing mode. Note that using a short line here also ensures that the /FIGURE switch is read before the first line of the new paragraph is ready for output.

7.2. Figure Text

Figure text provides CICERO with a much more powerful facility than the /FIGURE switch for setting and positioning figures and tables. Text that is delimited by figure delimiters is known as figure text. When it is encountered it is set as a contiguous whole. It is then positioned in exactly the same way as the /FIGURE switch blank space. Figure text allows the positioning algorithms of the /FIGURE switch to operate on chunks of contiguous text. Figure text provides contiguity but unlike the contiguous constructions of the last chapter, this contiguity is not achieved at the expense of large areas of blank space at the foot of the previous page in the cases where the contiguous block just fails to fit. Effectively, it allows the main text to flow past the contiguous figure into that blank space. Obviously, that is not the right solution for all types of contiguous text. However, for figures it allows captions to be attached to blank space or boxes to be drawn around the illustration. It is an ideal way to enter tables that are not referenced in line but by name or number.

The figure text delimiter is defined by the /FIGDELIMIT switch. Thus,

```
./FIGDEL:%
```

would define "%" as the delimiter for figure text. Recommended standard default delimiter settings are given in Appendix A.

On entry to figure text, the default processing mode is free text, that is there is no justification and the division into lines of the output follows that of the input. Users should take care to ensure that the height of the figure is not greater than the space available on any page. If it is CICERO will stop and print an error message.

Figures created with the figure text delimiter may extend over more than one column. However, the figure text itself must be set in one column. In the case of a table this means it must not be created with tabulated columns using the <TON> and <TOFF> predefined macros. It can have normal tabstops specified by the <TAB> predefined macro. Usually, that is the best way to set a table so the restriction is not significant. The number of columns required to take the figure is evaluated when the terminating delimiter is encountered. If it is bigger than the number of columns current, CICERO will print an error message and stop. Multiple column figures will be positioned at the first place where there is space enough both vertically and horizontally.

CHAPTER 8

Footnotes and Sidenotes

Footnotes and sidenotes provide means for expanding upon information provided in the main text without interrupting the flow of the argument. For instance, references to relevant published work may be included as footnotes. For formatting purposes, the footnote has two parts, the note itself and the reference to it in the main text. The object is to get the two parts on the same page if possible. When the formatting program paginates automatically, the user does not know where the page division will come and consequently, cannot be responsible for positioning the actual note. Instead, the program must do it automatically. The simplest way to let the program know which note belongs to which reference is to declare the footnote at the place where it is referenced. This may be done for CICERO by enclosing the footnote within delimiters.

8.1. The /FND switch

This switch is used to declare the footnote delimiter. Thus,

```
./FND:"
```

declares " as the footnote delimiter. Then text between the "s will be formatted into a footnote. The processing mode when starting to format a footnote is always free text. The mode existing at the point where the footnote is declared will be resumed when the footnote has been processed. If there is sufficient space on the page after the line containing the footnote reference has been set it will be added to the foot of the page. Otherwise, it will appear at the foot of the next page or in extreme cases, the first page upon which there is any space not already occupied by figures or footnotes. The /FND switch can also be used to specify a different spacing for footnote lines, a different typeface or a different type size. For instance, if "(" and ")" are the type size delimiters (see chapter 12) and "[" and "]" are the font name delimiters (see chapter 4), then

```
./FND:",9(8)[HR]
```

will cause footnotes to be set in 8 point characters with 9 point line spacing and the typeface HR. These values will be set up automatically on entry to all subsequent footnotes but may be changed for individual footnotes by enclosing the appropriate commands within the footnote. The default action if no values are specified with the /FND switch is to use the values current in the main text at the point where the footnote is declared.

The /FND switch can also be used to define just the spacing, point size and typeface for footnotes. To do this, simply omit the colon and delimiter declaration thus,

```
./FND,9(8)[HR]
```

The purpose of this variation is to allow these parameters to be defined when footnote declarations are to be delimited by the <FND> predefined macro (see section 10.5.9).

The current action for footnotes is as follows:- The call is replaced by a key determined by the option specified by the /FNOPTION switch below. The key also appears at the start of the footnote. Footnotes are separated from the main text by a line of horizontal rules. The rule character depends on the device and the typeface current. For typescript, an underline character is used. When photosetting, an EM or EN rule will be set. For most

layouts on the VIP phototypesetter italic faces have an EN rule whilst the others have an EM rule. EN rules do not join up and thus produce a broken line.

8.2. The /FNOPTION Switch

The key used to link footnotes to the reference in the text may be generated automatically by one of five different options. These are selected by the /FNOPTION switch which takes arguments as follows:

/FNOPTION:KEY:char[font],number

sets a key character char
a typeface font
and a starting number.

If no number is specified 1 is assumed initially and the switch causes no change to the sequence if issued subsequently.

If no typeface is specified the one current at the time the footnote is set is assumed for the text. For the note, the face defined by the /FND switch will be used if that is different.

If used at all this switch option must specify a key character. However, if the switch is omitted altogether, this option is defaulted and the key character will be ASCII #. The phototypesetter will probably set that as some other character depending on the mapping of characters on to the layout used.

/FNOPTION:SUPERSHIFT[font],number

Sets no key character but instead sets the number in supershift in the typeface specified and starting from the number specified. Depending on the typeface that will usually result in the number being set smaller and either superior or inferior to the line. If no typeface is specified the current face is used. If no starting number is specified the numbering sequence is left undisturbed. 1 is the default start for numbering.

/FNOPTION:MARK:char1[font1],char2[font2], ... char6[font6]

This option prints no number but a printers mark for the range specified. Specifying this option also sets the /FNZK switch automatically as the option is not good for printing large numbers. The normal limit to the number of printers marks that may be specified is 6 but this could be changed by modifying MAXFNCAR in the CONST section of the main program. If more notes are required the key character is duplicated or triplicated. For instance, in an extreme case suppose that only one character '*' is specified. Then the first note would be heralded by '*', the second by '**', the third by '***' and so on. To avoid filling the page with printers marks if the user specifies a silly starting number or the number gets corrupted a limit on the number of characters per herald is imposed. For simplicity, this limit is set at MAXFNCAR ie 6 too. Therefore, you cannot set more than 36 footnotes with this option. For large numbers of notes a numerical system is much better.

/FNOPTION:UP:offset[font](pointsize),number
/FNOPTION:DOWN:offset[font](pointsize),number

These options print no key character but offset the number as a

superscript or subscript. The switch will assign a scriptop operator to do the job for it and will fail if there are none left.

8.3. Numbering Footnotes

To distinguish different footnotes on the same page some footnote options use numbers. The default action is to number all footnotes in sequence. However, the switch /FNZK causes the count to be cleared every time there is no carry over of footnotes to the next page. The converse switch, causing the default action is /FNK. The count can be initialised to any value with the appropriate form of the /FNOPTION switch.

8.4. Footnotes in Multiple Columns

When printing in multiple columns footnotes will be appended to the right most column and may spread back into previous columns if there is insufficient space. At present there is no facility to force a footnote to the bottom of the column in which it is referenced or even to the bottom of the leftmost column. When forcing out a page it is usual to balance the blank space in each column. This can only be done if footnotes are appended to the end of the text before column evening rather than being attached to the bottom of the page. When a full page is output there is no difference between these cases. For single column output, it may be preferable to send the footnotes to the bottom of the page and as column evening never happens in this case there is no problem. These two alternatives are controlled by the complementary switches

./FNT - to append footnotes to the text

./FNB - to put the footnotes at the bottom

The former is the default action.

8.5. Sidenotes

The sidenote facility in CICERO provides unkeyed sidenotes for one or two column formats. For single column setting, the sidenotes appear in the right margin on all pages (since people read from left to right). For two column formats, the sidenotes relating to text in the left column appear in the left margin. There are no sidenotes in the central gutter. Sidenotes are not provided for formats with more than two columns because it would be necessary to position them in the gutters and that would seem a messy arrangement.

Because sidenotes are positioned close to the text to which they relate, a key system of the kind used to identify footnotes is not usually necessary. There is nothing to stop a user including a superscripted number or printers mark in the sidenote and at the point of reference. However, CICERO does not do it automatically.

An obvious disadvantage of formats with sidenotes is that space must be reserved for sidenote columns. To minimise the lost space, the width of this will usually be kept small. Consequently, the default action on starting a sidenote will be to turn off justification. That is automatic line division will be done in shiftable text but lines will not be justified. When setting sidenotes, CICERO assumes that the notes are distributed sparsly in the sidenote column. Sidenotes will only be transferred to the next page or column if the line of main text in which the note is declared is also transferred. If the sidenote will not fit within the space left in the sidenote column, it will be moved up in the column. If in doing this it runs into the bottom of an existing note that too will be raised. Provision is made for the specification of a minimum separation for sidenotes. If the addition of a sidenote would cause the column to overflow, the note will not be added and a diagnostic will be printed.

Sidenotes are allowed from normal text and from figure text. In the latter case, the sidenote will not be set until the figure is positioned. This allows captions for figures to be set in the sidenote column.

Finally when adjusting the text columns on the last page of a two column format so as to even up the columns, sidenotes will be transferred between the two sidenote columns if the calling line is transferred. In any event, the sidenote columns will be re-adjusted so as to align the sidenotes with the calling lines as closely as possible. This action also occurs for the sidenotes of the shortest column when columns are adjusted so as to align the bottom lines.

8.6. User Control of Sidenotes

Sidenotes are controlled by three switches and two predefined macros. The /SND switch defines the sidenote delimiter, line spacing, character size and typeface as follows:

```
./SND:char,spacing,(size)[font]
```

where

char is the character selected to delimit sidenotes.

spacing is the line spacing in the sidenote column.

size is the point size for sidenotes

font is the typeface for sidenotes

and (,), [and] would have been defined elsewhere as the point size and typeface delimiters.

The default action if the spacing, size or typeface specifications are omitted is the same as for the /FND switch described earlier in the chapter. Like the /FND switch, the /SND switch need not define a sidenote delimiter and ":char" can be omitted. Sidenotes should then be delimited by the <SND> predefined macro (see section 10.5.9).

The /SIDENOTES switch defines the gutter and line length for the rightmost sidenote column and optionally, the gutter and line length for the leftmost sidenote column if those values are to be different. Thus,

./SIDENOTES:g_r,l_r,g_l,l_l

Obviously, for a single column format one need specify the first two arguments only. If more are specified, the program will remember them and assume that the user may intend to change to a multiple column format later.

The /SNSPACE switch specifies the minimum spacing between sidenotes. Sidenotes will be positioned opposite the calling line if possible. Conflicts will be resolved by moving the new note down and the existing note with which it clashes up. During this process of adjustment, sidenotes will not be allowed to encroach closer than the minimum spacing set by the /SNSPACE switch.

For two column formats, one obviously has to reserve space for the left sidenote column in advance. This might not be strictly necessary for single column formats. However, efficiency is improved if the sidenote column is suppressed when not required. Consequently, space for sidenote columns will not be reserved until the <SNON> predefined macro has been encountered and a new page text block started. That happens on turning the page or on increasing the number of columns from one to more. The predefined macro <SNOFF> turns the reservation of space for sidenotes off.

CHAPTER 9

Indexes

An indexing facility is a means that allows entries declared at some point in the source text to be collected together for presentation at some other location in the output text. With this definition CICERO allows four types of index:

1. An alphabetically sorted index.
2. A table of contents.
3. A page reference line.
4. Tables of references.

It will be the function of this chapter to describe these four fully. In all cases the entry is declared by enclosing it within the appropriate kind of delimiters. For the first three the entry will normally also appear in the output text generated at the point of declaration. This action may be suppressed by making the entry from null mode text.

9.1. Alphabetic Indexes

This is a conventional index. Before printing it, entries are sorted into alphabetical order and identical entries merged. An example of a conventional index prepared entirely automatically will be found at the end of this manual. Entries for the index are declared by enclosing them within index delimiters defined by the /IND switch. Thus,

```
./IND:|
```

defines "|" as the index delimiter. Then for instance,

```
"In this piece of the source text the |text within these delimiters|
 will be entered into the index along with the number of the page on
 which the current line will be output."
```

The delimited text will also be printed in the output. To suppress that action the entry and its delimiters should be enclosed within null mode delimiters. These are defined by the /NMD switch. Thus,

```
./NMD:'
```

defines "'" as the null mode delimiter. If the processing mode outside the region delimited as null text is anything other than shiftable text it is important not to introduce stray carriage returns after the terminating null delimiter. Thus it is usually better to put the terminating delimiter on the next line. However, the null mode delimiter clears the start of line flag. This is an important flag in shiftable text when a tab at the start of line is used to signify a new paragraph or when a dot is used to signal a command line. In these circumstances, the null mode terminating delimiter must be on the previous line.

A common requirement is for sublists within indexes. CICERO accomplishes this by introducing an index level separator character defined through the /ISC switch. Thus,

```
./ISC:+
```

defines "+" as the index level separator. This character is used to separate different levels in the index entry. When printing the index, printing of an entry starts at the first level which is different to that for the previous entry. Each subsequent level is printed on a separate

line. In this way all levels are considered when ordering the index, but only those that differ from the previous entry are actually printed. For instance if two entries were:

```
Switches+/IND

Switches+/ISC
```

These would be printed in the index as,

```
Switches
    /IND
    /ISC
```

where "Switches" is the first level of both entries and "/IND" and "/ISC", the second level.

The indentation of sub-levels in the index is controlled by the /INDSUB switch. Thus,

```
./INDSUB:4
```

sets an indentation of 4 spaces (the default value) for typescript output. When photosetting the argument is interpreted as picas and points, for instance,

```
./INDSUB:2.3
```

would set up an indentation of 2 picas 3 points for sub-levels.

It would be unusual for an index entry containing sublevels not to be generated from null mode text. However, if the entry is also set on the page referenced, the index separator will appear as a substitute space.

Long entries are unusual in indexes but should an entry require more than one line it is usual to indent the continuation line. This indentation is set with the /INDINDENT switch. Thus,

```
./INDIND:4
```

sets an indentation for continuation lines of 4 spaces (the default value) for typescript output. Again, this switch can take an argument expressed in picas and points when photosetting. Also index entries that are long enough to occupy more than one line are not justified since the line length for the index is usually quite short. Justification of short lines is liable to produce unsightly white space and is best avoided.

Other parameters of the index pages may be set with the /INDEX switch which takes arguments as follows:

$$./INDEX:a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8[a_9](a_{10})[a_{11}](a_{12})<a_{13}>$$

where

a_1 is the line length for the output device.

a_2 is the line length for the listing device. When creating output for listing device alone, this parameter is omitted.

a_3 is the number of columns for the index. This may well be more than the main text.

a_4 is the left margin for odd pages.

a_5 is the left margin for even pages.

a_6 is the intercolumn gutter.

a_7 is the page length.

a_8 is the spacing between lines.

a_9 is the font name for the index text.

a_{10} is the point size for the index text.

a_{11} is the font name for the heading 'INDEX'.

a_{12} is the point size for the heading 'INDEX'.

a_{13} is the name of the macro to generate text to append after the index. Actually, it may contain a parameter list and repeats and is executed just as though it were a special character macro (see section 10.6).

also it is assumed that [], () and <> are respectively the font delimiters, the pointsize delimiters and the macro name delimiters.

It is not necessary to specify all the above parameters. Those not specified will be defaulted to the values ruling at the time the index is set. If no purely numeric values (a_1 to a_8) are being set, the ":" may be omitted. If some are to be specified all preceding values must be given. (Otherwise, the program has no way to identify which is meant). A value of -1 will default to the ruling value.

There are two ways to append text to the end of the index. One is to define the macro <INDFOOT> to be the text to be appended. If the user does not define <INDFOOT>, a predefined macro with that name is found. This causes the macro defined in the /INDEX switch string to be called. If no such macro has been specified the macro <INDFOOT> simply does nothing.

Before setting the index the predefined macro <INDHEAD> is called automatically to generate the centred heading 'INDEX' followed by a blank line. The user can redefine this macro to generate a different heading if required. Redefinition of any predefined macro always produces a warning diagnostic from the A.R.L. programs. Even though the redefinition in this case is likely to be intentional, there is no way the program can be sure. The safe thing to do is to print a warning always.

Under normal circumstances the index will be generated after formatting all pages of the main text and before generating a contents if there is one. Occasionally, it is necessary to locate the index somewhere else than right at the end of the publication. This can be done using the predefined macro <INDEX>. Note that generation of the index does not occur until all the input has been read but the intervening pages are dumped on to disk and reread after the index has been set. Page numbers both in the index and on the pages are adjusted to allow for the length of the index if it precedes the numbered page.

9.2. Contents

A table of contents differs from an index in several respects. The ordering is that of the entries in the output text. Usually that is the same as the order in the input text and there is consequently no need to sort the entries before printing. Because the user can control the order of entries there is no need for a sub-level system as there was for alphabetic indexes. There should be no need to merge identical entries and there is, therefore, only one page number for each entry. The contents is also

usually required at the beginning which creates additional problems since the information is not usually complete until all the input text has been divided into pages.

Entries for the table of contents are made by enclosing them within contents delimiters defined by the /TCD switch. Thus,

```
./TCD:`
```

defines "`" as the table of contents delimiter. Such entries normally also appear in the normal output text for the page currently being set. To make the format of an entry in the contents different to that in the main text (eg. to indent the entry) null mode may be used to advantage. However, note the comment about stray carriage returns that was made for index entries. As headings are often set in free or right text and contents entries are often headings, that comment is even more relevant here.

The index level separator character may be used in contents entries to force the start of a new line or to add vertical spacing before the entry. The first index separator character at the start of an entry causes the line spacing to be increased by the setting of the /BLANKSP switch. Further index separator characters add spacing equal to the current line spacing. Thus the action is equivalent to that of extra CRLFs in shiftable text. If the index separator occurs in the middle of the entry, the entry so far is printed and a new line started . There is no change in left margin indentation as in this case the margin indentation can be controlled by including substitute spaces after the index separator. The purpose of this facility is to allow section headings to be included in the contents.

When printing the contents the processing mode at the start of each entry is shiftable text. The last line of each entry is justified with a leader (a one character repeated until the space is filled) and the page number appended. The character used as leader may be specified using the /TOCLEADER switch. Thus,

```
./TOCLEADER:.(6)[HI]
```

would specify a leader character "." to be set in 6 point and with the typeface HI, where (and) are the point size delimiters defined by the /SIZE switch (see chapter 12) and [and] are the typeface delimiters. It would be more usual to use an EN leader character than a dot. An EN leader is a dot in the middle of an EN space. The normal width of a dot is a thin space so this will result in a line of dots spaced rather more generously. An EN leader is in fact the default leader character and will be selected if the ":" is omitted. The switch then just specifies the typeface and point size. These may be specified in the opposite order and if either is missing the default action is to leave that setting unchanged from what it was at the end of the contents entry. A space may be specified as the leader character. (See the contents at the front of this manual).

The indentation of continuation lines is set by the /TOCINDENT switch. Thus,

```
./TOCIND:10
```

sets an indentation of 10 spaces (the default value) for typescript output. As with the equivalent switch for index entries, this switch takes an argument in picas and points when photosetting.

Space may also be reserved for page numbers with the /TOCNOS switch. Thus,

```
./TOCNOS:6
```

sets a column of width 6 spaces (the default value) for page numbers for typescript output. When photosetting this argument is interpreted as picas and points.

Other parameters of the contents page may be set with the /CONTENTS switch which takes the same arguments as the /INDEX switch. Again, there are two ways to append text to the table of contents. Either one may define the macro <TOCFOOT> directly, or one may include the macro name in the /CONTENTS switch specification and allow the predefined macro <TOCFOOT> to call it up. The macro <TOCFOOT> is called at the end of the table of contents always.

Before setting the contents the predefined macro <TOCHEAD> is called automatically to generate the centred heading 'CONTENTS' followed by a blank line to separate it from the entries that follow. The user may redefine this macro to generate an alternative heading. For instance, for this manual the macro was redefined as,

```
<TOCHEAD=@_CONTENTS_
@
$Page
$
>
```

Redefinition of any predefined macro always produces a warning diagnostic for the A.R.L. programs. Even though the redefinition in this case is likely to be intentional, there is no way the program can be sure. The safe thing to do is to print a warning diagnostic always.

Like the index, the table of contents cannot be generated until all the input file has been read. When it must be positioned elsewhere, this can be specified by placing it with the <CONTENTS> predefined macro. The table of contents is always printed on a separate page or pages. The action upon encountering the <CONTENTS> predefined macro is to output the current page and then to dump subsequent pages in binary on to temporary disk storage. When the contents has been set these pages are re-read and set. If there is an index as well, its location will be entered automatically in the contents. Page numbers will be adjusted both in the contents and in the referenced pages to compensate for the lengths of the contents and/or index if they follow them. They will also be printed in either index in Roman or Arabic numbers depending upon which system was used on the page referenced.

9.3. The Page Reference Line

A page reference line is a kind of running index often used in manuals. Entries are accumulated and printed at the bottom of the page to which they relate. On turning the page the last entry is carried forward to the next. In this way the line presents a summary of what is on the page. Entries for the page reference line are made by enclosing them within page reference delimiters defined by the /PRD switch. Thus,

```
./PRD:!
```

defines "!" as the page reference delimiter. Like the /FND switch for footnotes this switch can take further arguments to define the spacing for the page reference line, the character size and the typeface. For instance,

```
./PRD:!,11(10)[HB]
```

would set up a spacing of 11 points, a character size of 10 points and select the typeface defined as HB by the user. Note that before this switch is used, the character size delimiters, here "(" and ")", the font name delimiters, here "[" and "]", and the particular typeface HB (see the

/ASSIGN switch in chapter 12) must all have been defined. Note that the space to be occupied by the page reference line must be established before text is entered on to the page. Therefore, the page reference line is limited to one line and the spacing is not effective until the next page is started. Again like the /FND, /SND and /REFN switches, the /PRD switch need not specify a delimiter. If the colon and delimiter are omitted,

```
./PRD,11(10)[HB]
```

just sets up the spacing, point size and typeface for the page reference line. Entries must then be delimited by using the <PRD> predefined macro (see section 10.5.9).

The start of the page reference line may be indented by using the /PRINDENT switch. Thus,

```
./PRIND:10
```

sets an indentation of 10 spaces for typescript. The default value is zero. When photosetting, the argument is interpreted as picas and points.

At the present time the length of the page reference line is limited only by storage considerations and the physical capabilities of the output device. An error message will be printed if the storage capacity of the line is exceeded. This is not entirely a satisfactory system as a user might wish to have some prior warning that the line has become unduly long before setting it. Therefore, we plan to introduce a /PRSIZE switch to limit the line length for this line and print a diagnostic message whenever it is exceeded.

9.4. References or End Notes

The purpose of the reference facility is to allow references to be declared within the text at the point where they are referenced but to be assembled together into a table at the end. Such a scheme is primarily a handy convenience for authors, allowing ready addition of extra references should that become necessary as the work progresses. A few journals require references to be sorted into alphabetical order of first author prior to numbering, a task that it is obviously sensible to do automatically. Unfortunately, implementing this requires a preliminary scan to pick up references which are then sorted and numbered. A less demanding requirement is to number the references in order of their occurrence in the text. This can be done in one pass. Which of these two options is performed depends on the location of the /REFN switch that defines the reference delimiter for numbered references. To do a preliminary scan, the switch must be specified in the terminal command string. If it is declared within the source file references have to be numbered in order of occurrence. The basic format of the /REFN switch is

```
./REFN:!
```

which defines "!" as the delimiter for numbered references.

When printing the reference table the reference number may be preceded by an arbitrary string which can also be defined through the /REFN switch. Thus,

```
./REFN:!,"Ref*"
```

defines a string Ref* which will precede every reference in the table of numbered references. Note that any character can be used in place of the "s to delimit the string. This string is processed as though it were input text and all control characters in it have the effect appropriate to them

at the time the reference table is printed. Thus in the example "*" might be a substitute space. The string could be used to set the character size and typeface or to control the indentation of subsequent lines. The colon and delimiter declaration can be omitted from the /REFN switch, in which case it defines just the string to precede each entry. This allows the string to be defined when references are being delimited by the <RFND> predefined macro (see section 10.5.9). Note that a preliminary scan to order the references will not be done in this case so references will be numbered in their order of occurrence.

A second kind of reference is permitted by CICERO. These are alphabetic references and are not numbered. For instance, in the main text one might require,

"(Jensen and Wirth, 1975)"

whilst in the reference table the corresponding entry might be

"JENSEN, K and WIRTH, N. (1975), "PASCAL - user Manual and Report"
Springer-Verlag, 2nd Edition., New York 1975"

In this case the user is responsible for specifying the entry in the text, i.e. "(Jensen and Wirth, 1975)" whilst the entry for the table follows it enclosed within reference delimiters defined by the /REFA switch. Thus,

./REFA:~

defines "~" as the delimiter for alphabetic references.

When setting the reference table for either kind of reference the processing mode is set to shiftable text before starting to set each reference. Indentation of the subsequent lines can be accomplished by including predefined macros in the entry declaration.

The reference tables will normally be output at the end with the table of numbered references preceding the alphabetic table if both exist. Alternatively, the predefined macros, <REFA>, <REFN> or <REFERENCE> may be used to force out either or both tables. The predefined macro <REFERENCE> puts the alphabetic reference table before the table of numbered references if both are present. Printing the reference table clears the list of references, consequently, one may have multiple reference tables at the end of each chapter so long as their generation does not require a preliminary pass. A preliminary pass is necessary if the references have to be sorted into alphabetical order before numbering or if the table has to be printed before all the entries have been added to it.

End notes are notes that are set at the end of a chapter rather than the foot or side of a page like footnotes or sidenotes. The /REFN facility was originally designed to set numeric references in papers. However, the content of the reference is not important for the setting process. The facility is just as good for end notes, the predefined macro <REFN> being used to output and re-initialise the table at the end of each chapter. Bibliographies are probably better coded using the /REFA type of alphabetic reference, printing the table once at the end of the publication. The tabulated columns facility (see sections 6.4 and 10.5.5) provides a powerful means to set bibliographies that are not generated automatically.

CHAPTER 10

Symbols and Macros

The purpose of this chapter is to describe the facilities which CICERO provides for storing and processing objects by symbolic name. Giving an object a symbolic name makes it identifiable by CICERO and allows the program to be instructed to do things with the named object. This saves the user time and effort. An extension of the naming system to allow predefined names provides a means to allow arbitrary extension of the control language without requiring the definition of more special characters. Conversly, a facility is also available to associate macro names with particular specially defined local control characters.

CICERO allows four classes of named objects:

1. Macros.
2. Tab stops.
3. Counters.
4. Predefined macros.

The A.R.L. implementation of CICERO allows names of up to 10 alphanumeric characters.

10.1. Name Delimiters

The variable nature of natural language text makes the automatic identification of names within such text impracticable. Consequently, all operations on named objects must be enclosed within delimiters. These will be called "macro delimiters" although they are used to delimit all four classes of named objects.

Macro delimiters are defined by the /MACRODELIM switch. Thus,

```
/MACROD:<>
```

defines "<" and ">" as the macro delimiters. Notice that, in order to allow macro calls within macros it is necessary to specify different opening and closing delimiters.

10.2. Macros

Macros are named strings of text. Whenever the macro call occurs it is replaced by the macro string. The string may contain further macro calls. The simplest form of macro definition allowed by CICERO is

```
<NAME=string>
```

whilst the simplest call is

```
<NAME>
```

where "<" and ">" are the macro delimiters and "NAME" is the symbolic name of the macro. The "=" after the macro name identifies the first form as a macro definition. All such a simple definition does is to provide a handy abbreviation for a string of text that occurs frequently in the text. This is not a negligible accomplishment if the stored string can contain control information since it means that control can be grouped. By changing the macro definition, the change is automatically implemented for all the calls. For instance, suppose the macros <SH> and <XH> are used to start and

terminate headings respectively, and that these macros accomplish the necessary changes in point size and typeface. Then firstly, all headings will be set with the same point size and typeface. Secondly, to change these settings, it is necessary only to modify the definitions of <SH> and <XH>.

A small increase in complexity is to allow repeated macro calls. This is coded for CICERO by following the macro name with an asterisk and the decimal number of repeats. Thus,

```
<NAME*10>
```

is a macro call that will result in the macro string defined by "NAME" being repeated 10 times.

Finally, one may add a formal parameter list to the definition and an actual parameter list to the call. Thus,

$$\text{<NAME(A,B)=string}_1\text{<A>string}_2\text{<B>string}_3\text{>}$$

defines the macro NAME with formal parameters A and B whilst,

```
<NAME(X,Y)>
```

calls it with actual parameters X and Y. The identification of formal parameters within the string is subject to the same difficulties as the identification of named objects within the main text. It is sensible to adopt the same solution and enclose formal parameters within the defined string between macro delimiters. Thus the formal parameters provide a linking mechanism defining named objects within the string at call time. For the present, CICERO requires that the actual parameters (X and Y in the example) should also be names. The objects named need not be macros but can be any of the four types of named object allowed by CICERO.

When a macro string itself contains definitions of named objects those definitions are valid only during the call unless the name is passed as a formal parameter in which case the scope of the symbol is determined by the actual parameter. This permits a macro to contain, for instance, locally defined tabstops which are unique to the particular level of call. Even if the macro is called recursively each level of call retains the value current at the time of definition during the call.

10.3. Named Tabstops

Named tabstops allow a particular horizontal position to be referenced by name. Thus,

```
<NAME.>
```

defines NAME as the current horizontal position. The dot following NAME signifies to CICERO that this is a tabstop definition. Then,

```
<NAME>
```

causes a tab forwards or backwards to that named horizontal position. Notice that the call of a named tabstop is formally the same as that of a simple macro. CICERO remembers the kind of object it has to store obtaining the information from the form of the definition. At present CICERO also allows one to redefine names changing the class of object to which they relate. A warning message will be produced if the format of the call is obviously inconsistent with the definition (for instance, a parameter list or repeats requested for a tabstop).

Named tabstops are invaluable when setting mathematics. However, they have many other uses. For instance, the common indentation of all the examples in this manual is achieved by using a named tabstop.

10.4. Counters

Counters provide a simple means for generating and incrementing automatically, numbers or letters in the text. Usefully, they may be used to number sections or equations. Their principle justification is that editing is made much easier as entire sections can be added or deleted and the numbering scheme will adjust automatically. Numeric counters are defined as follows:

```
<QC:=1>
```

defines QC as a counter and assigns it the value 1. Then,

```
<QC>
```

transmits the value 1 to the CICERO whilst,

```
<QC+>
```

increments the stored value by 1 and transmits the incremented value. Auto-decrementing is also possible. Thus,

```
<QC->
```

decrements the stored value by 1 and transmits the decremented value.

Counters may also be assigned alphabetic values. Thus,

```
<QA:='a'>
```

defines QA as a counter and assigns it the value 'a'. Then

```
<QA>
```

transmits the letter 'a' to CICERO. Both auto-incrementing and decrementing may be applied to alphabetic counters. Note that repeats are also valid. Then the sequence,

```
<QA:=' '>
<QA+*94>
```

would generate all the printable ASCII characters. Alphabetic counters that overflow due to auto-incrementing start again with a space whilst those that run out due to auto-decrementing restart with an "~", ASCII 126.

Numeric counters may be used to specify the number of repeats of a macro call. Thus,

```
<NAME*<QC+>>
```

is perfectly valid.

10.5. Predefined Macros

Predefined macros have essentially the same function as local control characters. However, instead of the function being initiated by the occurrence of a specially assigned character in the input file, it is generated by a macro call. This enables the local control concept to be extended arbitrarily without risk of exhausting the set of suitable characters. The cost of this extension is that the call is rather more verbose. Consequently, predefined macros are used mostly for those functions that are seldom

required. The following subsections list all the predefined macros currently available. Note that, unlike switch names, the names of predefined macros cannot be abbreviated but must be typed in full.

10.5.1. Substitute Characters

A range of fixed spaces of different widths is required for typesetting. Unlike the spaceband character, these spaces retain their widths during justification. Since all characters have the same spacing these spaces are not found on an ordinary typewriter and consequently are not part of the normal ASCII character set. The following predefined macros are available for entering the fixed spaces required for typesetting.

<EMSP> transmits an EM space.
<ENSP> transmits an EN space.
<TSP> transmits a thin space.
<VTSP> transmits a unit space.
<VTBS> transmits a unit backspace.
<3USP> transmits a 3 unit space.

The significance and use of these spaces will be described in the chapter on typesetting (section 13.3). In addition to fixed spaces,

<NLDR> transmits an EN leader.

An EN leader has the width of an EN space and contains the "EN leader" character for a particular typeface. Usually, that character is a dot. The EN leader character is often repeated to make a line of dots.

Finally, the predefined macro <FF> has the same effect as a form feed except that the latters action can be suppressed with the /NOFF switch, whereas that of the predefined macro cannot. Thus it is useful for forcing new pages when the input has to contain page marks for editing convenience.

10.5.2. Local Control of the Left Margin

The convention described in chapter 2 which allows the indentation of the left margin to be set by entering leading spaces on a line, is unfortunately inadequate for photosetting since indentations that are not multiples of EN spaces are often required. Commonly, it is required that the left margin should align with the start of some character in the previous line. Alternatively, CICERO allows the left margin to be incremented by combinations of fixed spaces. Then,

<LM> sets the left margin for future lines of shiftable text at the current position while,

<LEMSP> increments the left margin by an EM space.
<LENSP> increments the left margin by an EN space.
<LTSP> increments the left margin by a thin space.
<LVTSP> increments the left margin by a unit space.
<LVTBS> decrement the left margin by a unit space.
<L3USP> increment the left margin by a 3 unit space.
<ZM> clears the left margin indentation.

Of these <LM> is the most commonly used form. Note that the indentation is cancelled on leaving shiftable text or by a double CRLF. Repeats are allowed for predefined macros so that,

<LVTSP*7>

would set an increment to the left margin of 7 unit spaces.

```
********<TFIX>x**********<TFIX>y*************<TFIX>z
A<XPND>1<ALIGN>.2<XPND>1000<ALIGN>.0<XPND>0<ALIGN>.7468
B<XPND>217<ALIGN>.0<XPND>0<ALIGN>.457<XPND>72<ALIGN>.13

sets as:

       x          y           z
A    1.2      1000.0       0.7468
B  217.0         0.457     72.13

That coding may seem somewhat verbose to achieve that ef-
fect. However if we use the special macro facility
described in section 10.6 the macro calls can be
generated from one character. Thus

./SPMACRO:|:<XPND><ALIGN>

********<TFIX>x**********<TFIX>y*************<TFIX>z
A|1|.2|1000|.0|0|.7468
B|217|.0|0|.457|72|.13

sets as

       x          y           z
A    1.2      1000.0       0.7468
B  217.0         0.457     72.13
```

Fig. 10.1. Example of aligning tabs.

10.5.3. Simple Tabulation

In free text tab characters in the input will cause tabulation to the next tab column, the column spacing being uniform and set by the /TABSIZE switch. There is often a need to set tab stops at non-uniform intervals. The following predefined macros allow this:

<STAB> sets a tabstop at the current horizontal position.
<ZTAB> clears all tab stops so defined.
<TAB> spaces across to the next defined tabstop.

Note that the tabstops defined in this way are totally independent both of the uniform tabstops defined by the /TABSIZE switch and the named tabstops defined earlier in this chapter. All three may be used at once.

10.5.4. Aligning Tabs

Sometimes it is necessary to align not the end of a space but some point in the following word for several lines. A common example is the need to align decimal points in tables of figures. However, there are many other cases where the same technique can be beneficial. For instance, when setting mathematical equations, it is often better to align the equals signs. The following predefined macros,

```
./TABULATE:17,1,34,2,16
./TSPAC:8
<TON> Gingrich, P.B.
 Child, R.D.
 Panageas, G.N.
<TNEXT>#Aerodynamic configuration development of the Highly
Maneuverable Aircraft Technology remotely piloted research vehicle
<TNEXT>NASA  CR-143841, June 1977.
<TSTART> McLaughlin, M.D.
<TNEXT>#Calculations, and comparison with an ideal minimum, of trimmed
drag for conventional and canard configurations having various levels
of static stability<TNEXT>NASA  TN-D-8391, May 1977.
<TSTART> Gloss, B.B.
<TNEXT>#Effect of canard location and size on canard-wing interference
and aerodynamic center shift related to maneuvering aircraft at
transonic speeds<TNEXT>NASA  TN-D-7505, June 1974.
<TSTART> Gloss, B.B.
 McKinney,L.W.
<TNEXT>#Canard-wing lift interference related to maneuvering
aircraft at subsonic speeds<TNEXT>NASA TM-X-2897, December,1973.
<TSTART> Krouse, J.R.
<TNEXT>#Effects of canard planform on the subsonic aerodynamic
characteristics of a 25 degree and a 50 degree swept-wing research
aircraft model<TNEXT>NSRDC EV-AL-91, May 1972.
<TOFF>
```

Fig. 10.2. Source text to produce the tabulated columns setting shown in fig. 10.3.

```
<TFIX> set an aligning point here.
<XPND> is replaced by a variable space to align,
<ALIGN> this point under the next spcified by <TFIX>.
```

Figure 10.1 shows some examples of the use of aligning tabs to align the decimal point in tables of numbers.

10.5.5. Tabulation in Columns

The /TABULATE switch to define tab columns for column tabulation has been described already in chapter 6. Four predefined macros are used to control tabulation in columns:

```
<TON> to switch tabulation on.
<TNEXT> causes filling of the next column to start.
<TSTART> outputs the previous tabulated record and starts at the
        left column again.
<TOFF> turns off tabulation in columns.
```

The difference between simple tabulation and tabulation in columns is that records are not restricted to one line in length. Text may even be justified within columns. At present the processing mode continues over columns within a record but is reset to free text at the start of each new record (<TON> and <TSTART>). It is important to emphasise that for a particular record all the data for each column in turn is entered before

<TNEXT> starts the filling of the next column. A simple example is given in figures 10.2 and 10.3. Figure 10.2 shows the source code and figure 10.3 the result.

Gingrich, P.B. Child, R.D. Panageas, C.N.	Aerodynamic configuration development of the Highly Maneuverable Aircraft Technology remotely piloted research vehicle	NASA CR-143841, June 1977.
McLaughlin, M.D.	Calculations, and comparison with an ideal minimum, of trimmed drag for conventional and canard configurations having various levels of static stability	NASA TN-D-8391, May 1977.
Closs, B.B.	Effect of canard location and size on canard-wing interference and aerodynamic center shift related to maneuvering aircraft at transonic speeds	NASA TN-D-7505, June 1974.
Closs, B.B. McKinney,L.W.	Canard-wing lift interference related to maneuvering aircraft at subsonic speeds	NASA TM-X-2897, December,1973.
Krouse, J.R.	Effects of canard planform on the subsonic aerodynamic characteristics of a 25 degree and a 50 degree swept-wing research aircraft model	NSRDC EV-AL-91, May 1972.

Fig. 10.3. Example of tabulated columns. The figure is the result of setting the data shown in fig. 10.2. Note this figure was not and could not at present be set as figure text.

10.5.6. Output Switches

The photosetter and list device may be controlled by the following predefined macros:

<FON> turns printing on.
<FOFF> replaces printing by equivalent spacing.
<RED> prints in red on the list device.
<BLACK> prints in black on the list device.

Also in this class are the predefined macros to control the setting of simulated bold and extra bold on fractional spacing daisy wheel printers Bold text is simulated by overprinting with the same character offset by the minimum fractional horizontal space (usually 1/120 of an inch). The result is to broaden the character horizontally. Typographically, it is somewhat less than splendid. Probably, the best results are achieved with a sans serif typeface like Letter Gothic. The option is controlled through the following predefined macros:

<TNRM> restores normal printing.
<TBLD> **causes one offset overprint.**
<TXBLD> **causes two offset overprints.**

10.5.7. Placing Indexes etc.

The location of contents, index and tables of references are now specified with predefined macros rather than switches (as was the case for earlier programs). Thus,

<CONTENTS> defines the position of the contents.
<INDEX> defines the position of the index.
<REFA> prints the alphabetic reference table.
<REFN> prints the numeric reference table.
<REFERENCE> prints either or both.

These predefined macros are described in the appropriate sections of chapter 9.

10.5.8. Macros to Facilitate Macro Coding

A number of predefined macros have been included primarily to make it easier to write general purpose macros. For instance, the predefined macro <ADDPLD> causes the line spacing to be incremented appropriately on the next change of point size. The purpose envisaged for this macro is to increase the line spacing for headings when the point size is changed. Normally, the code that evaluates the spacing for blank lines output via a double CRLF in shiftable text or for the first line of free text, looks ahead for a change of point size. However, this look ahead is defeated if the point size change is concealed within a special macro character call (see section 10.6). Thus, in the example given in 10.6., the macro <SH> should call <ADDPLD> if it produces a change of point size, eg.

```
<SH=<ADDPLD>(12)[HB]>

<XH=(10)[HR]>
```

where it is assumed that () are the point size delimiters and [] the typeface delimiters. [HB] is a bold typeface and [HR] is a medium face.

The predefined macros <NOHYPHEN> and <HYPHEN> allow local control of hyphenation when hyphenation has been specified with the /HYPHEN switch. Otherwise, they do nothing. The intended purpose of this facility is to suppress hyphenation for headings in photoset text. For typescript, this action is an automatic consequence of underlining. Probably one would not include these macros explicitly. but call them by including them in the definitions of <SH> and <XH> above.

The predefined macro <UPGNL1> sets a flag that causes the contiguity requirement for the line to be incremented by 2 when the line is set. It is effective only if called at the start of the output line. These conditions are the same as those described for underlining in typescript and the purpose is the same, to make the contiguity requirement for headings be bigger than that for a forced new line. It is useful when photosetting. More generally, the predefined macro <UPGNL> increments the contiguity requirement by 1 but may be repeated and need not be given at the start of a line.

The predefined macro <NOJUST> turns off justification until a line is forced out quadded. Its purpose is to allow headings to be set in shiftable text but without justification if they occupy more than one line. Whilst it could be called explicitly the intention is that it should be included within the start of heading macro string. The action of the <NOJUST> macro is also defaulted for index entries since these are usually set on very

short lines and the visual effect of a few sparsly filled justified lines is not pleasing.

The writing of general macros is also facilitated if the current typeface and point size before the call can be restored upon exit. The following predefined macros save and restore these parameters on stacks:

<SVF> saves the current typeface or font.
<RSF> restores the typeface from a stack.
<SVS> saves the current point size.
<RSS> restores the point size from the stack.

Separate stacks are used for fonts and sizes and the stack sizes allow 16 values to be stored. As an example, suppose a macro to generate a large sigma is required. On the Diablo, one would probably produce this by a combination of slashes and underlines. However, on a photosetter, the result would be better if it were borrowed from a greek typeface and set in a larger point size. One might do this with a macro,

<SIGMA=<SVF><SVS>(18)[TC]S<RSS><RSF>>

Whilst this definition appears long, it can very likely be included in a default file, and all the user needs to enter to generate a sigma set in 18 point is <SIGMA> without regard for the typeface current or the ruling point size, both of which will be restored after the call.

10.5.9. Macros to Conserve Special Characters

ASCII provides only a limited set of PI* characters suitable for local control. It seems possible that some applications might exhaust this limited range. Consequently, the following predefined macros have been implemented to reduce the number of special characters required. They are not usually the best way to implement functions. However, in conjunction with the special character macros defined in section 10.6., they might form the basis for a more powerful control system with many functions grouped on one control character.

<CTD> - Centre text delimiter.
<FIGD> - Figure text delimiter.
<FND> - Footnote delimiter.
<IND> - Index entry delimiter.
<ISC> - Index entry level separator.
<LIT> - Literal marker.
<NMD> - Null mode delimiter.
<PRD> - Page reference entry delimiter.
<RFAD> - Alphabetic reference delimiter.
<RFND> - Numeric reference delimiter.
<RTD> - Right text delimiter.
<SND> - Sidenote delimiter.
<STD> - Shiftable text delimiter.
<SUB> - Substitute space.
<TCD> - Table of contents entry delimiter.

10.5.10. Macros Used When Setting the Index or Contents

The predefined macros <TOCHEAD> and <INDHEAD> generate the centred headings followed by a blank line for the contents and index respectively. If no macro delimiters are defined the headings will be generated by other

---

* characters other than the upper and lower case alphabets, the numerals and the normal punctuation marks are commonly called PI characters by compositors.

means. However, this use of the predefined macros allows the user to substitute his own definitions and thus to specify alternative headings.

The predefined macros <TOCFOOT> and <INDFOOT> are defined primarily because macros of these names are called whenever the index or contents are set. If the user defines them his version will be used. Alternatively, they will call the macros defined in the /INDEX or /CONTENTS switches.

10.5.11. Macros to Control Text Processing Sub-Modes

Two sub-modes of free text are delimited by predefined macros rather than by single character delimiters. The reason for this implementation is that they are likely to be used relatively rarely (for instance, to set chapter headings). Consequently, it is not sensible to tie up a special control character as a mode delimiter. The predefined macros are used in exactly the same way as single character delimiters. That is, the first occurrence turns the mode on, the second turns it off.

The function of the two modes is to shift text away from or towards the binding. The normal binding convention is that odd numbered pages have the binding on the left. Consequently, the action in these modes depends on the page number when the page is output rather than when the line is input. The page number may be modified, for instance, by a preceding table of contents. The implementation works simply by swapping the left and right margin increments when necessary. This can be done when the line is being output.

The predefined macros are:

<QOUT> Quad out from the binding.

<QIN> Quad in towards the binding.

<QOUT> is particularly useful for chapter headings as it allows them to be shifted to the position on the page where they will be most prominent.

10.5.12. Macros to Control Sidenote Column Reservation

When setting sidenotes for double text columns it is obviously necessary to reserve space for the left sidenote column even though there may be no entries in it. Equally, even when setting in a single column format there is likely to be some loss of efficiency in establishing the sidenote column. Consequently, sidenotes are enabled and disabled by the predefined macros <SNON> and <SNOFF> respectively. These are effective at the time the next page text block is commenced. That is usually when the next page is started. However, it also happens when a transition is made from one to more columns.

10.5.13. Predefined Macros with Parameters

There is no reason why a predefined macro should not accept a parameter list as argument. Most do not need such a list to perform there function and the only one that does is the <NCHAP(NAME)> predefined macro that takes as parameter the name of the macro or counter to be used for the chapter descriptor of subsequent pages. Error diagnostics are printed at the time of call if the symbolic name supplied as a parameter is not defined as a macro or counter. For instance, it cannot be a predefined macro nor a named tabstop. A warning message is also given if the name supplied is that of a macro which itself has a parameter list.

In detail the action of the <NCHAP(NAME)> predefined macro is as follows:

1. Force out pages until there are no figures or footnotes that have not been set. That is terminate the previous chapter.

2. If name is a macro check that it has no parameter list. If it has warn the user. The name will be used but there can be no actual for formal substitution whenever the chapter descriptor is referenced. For macros, the string address is stored by the <NCHAP(NAME)> predefined macro. This allows the name to be redefined, for instance for the next chapter, without affecting the chapter descriptor on the following pages until the next <NCHAP(NAME)> predefined macro is encountered.

3. If the name is a counter, autoincrement it and store the incremented value as the chapter descriptor for subsequent pages. For instance, one could use a single counter to define the chapter number. This could then appear in the heading for each chapter and also in each section heading. It would be incremented automatically by the <NCHAP(NAME)> call at the start of each chapter. Additional chapters could be inserted and the numbering scheme would adjust automatically.

4. Set the page number to 1 for the next page.

5. Write the chapter descriptor and page number to the new page.

10.6. Special Character Macros

Taking the opposite approach to that of section 10.5.9., on predefined macros, it is also possible to cause a special character to call either a named object or one of two named objects alternating between the two. The switch, /SPMACRO is used to associate the special character with the object names. Thus,

```
./SPMACRO:%:<NAME1><NAME2>
```

associates the character "%" with the names NAME1 and NAME2. On the first occurrence NAME1 is called, on the second, NAME2 and so on. NAME1 and NAME2 will most probably be macros, though they can be any named object (i.e. macros, named tabstops, counters or predefined macros). Note that for macros, counters and predefined macros the text between the macro delimiters may contain a repeat. For counters, it may contain a plus sign or a minus to cause auto-incrementing or decrementing whilst for a macro there may also be an actual parameter list. None of the names so defined need be declared prior to the switch definition. However, all names must have been declared before they are actually called by the occurrence of the special character. The calling mechanism looks for the called object by name. Consequently, the assignments of objects to the names can be changed during processing without having to redefine the switch.

To associate a single name with a special character it is necessary to specify the name only once. Thus,

```
./SPMACRO:%:<NAME>
```

would cause <NAME> to be called for each occurrence of "%" not preceded by a literal character.

An important use of special character macros is to enable underlining in a version of the file intended for typing to be replaced by a change of point size and/or typeface when the file is photoset. For instance, suppose the macros <SH> and <XH> accomplish the required change in typeface and point size and, that in the version for typing, "_" is the underline delimiter. Then the special character macro definition,

```
./SPMACRO:_:<SH><XH>
```

would replace underlining by the required changes in typeface and point size when photosetting. Now suppose that the definition of the underline

character is made in the default file called by the /SDEFAULT switch when outputting to the list device, whilst the definitions of the special character macro and macros <SH> and <XH> are made in the default file called when photosetting. Then the same file can be used for photosetting or typing with **absolutely no change in the data.** That is a significant achievement as it means most of the errors can be eliminated without producing any photoset copy.

CHAPTER 11

Special Facilities for the Diablo 1610/1620 and Qume Sprint 5 Printers

The Diablo 1610 is a microprocessor controlled daisy wheel printer. It is capable of fractionally spacing both vertically (in increments of 1/48 of an inch) and horizontally (increments of 1/120 inches). When provided with a two colour fabric ribbon it can change ribbon colour under program control. Alternatively, using a carbon ribbon it is capable of fairly high quality typescript.

The microprocessor control allows the printer to perform various useful functions with a minimisation of both data transmission and mechanical motion. For instance, it can tab to absolute positions both vertically and horizontally. This is much faster than a succession of spaces or line feeds as the motion does not have to be arrested after each successive character. The microprocessor can also control character spacing by program control. Not only does this allow the pitch to be set by program, but justification can be done very evenly by varying the spacing of a space. It is also possible to increase the line spacing by a small amount for new paragraphs or before headings. Also sub and superscripts can be offset by less than a whole line feed. It is the purpose of this chapter to describe the additional facilities which CICERO provides for the control of these functions.

The data input to the printer is as 8-bit ASCII characters transmitted across a serial RS232 interface. This means it can be connected to a modem or used as a replacement for a terminal in most computer systems. This is not necessarily true for all printers designed for word processing applications. As many of the control functions are not provided directly in the ASCII code they are reached via escape sequences. An escape sequence may be any sequence of characters following an escape character (ASCII code 33 octal).

Although the printer is limited by mechanical constraints to a printing speed of about 45 cps it is advantageous to transmit data to it at the maximum transmission speed of 1200 baud (120 characters per second). This is because the escape sequences that vary the horizontal and vertical spacing are non-printing and, therefore, are processed very rapidly by the microprocessor. Text that is justified or printed in multiple columns is likely to contain many such control sequences. When running the printer in this way care has to be taken to ensure that the buffer in the microprocessor is not overwritten. A program to do this is described in Chapter 17.

The 1620 terminal is basically a 1610 printer with a keyboard for input. A.R.L. has a 1620 terminal.

The Qume Sprint 5 is a microprocessor controlled printer/terminal very similar to the Diablo 1610/1620. It will correctly interpret all the escape sequences coded for the Diablo but also has some of its own. It can print the 95th and 96th character positions on the daisy wheel which the Diablo can not. It also has a proportional spacing mode in which each character has to be followed by a 7-bit character giving width and hammer intensity information. Its printing speed is marginally faster than the Diablo (55 cps instead of 45) and the maximum transmission speed of 1200 baud is switch selectable. The buffer synchronisation protocol is the same as that for the Diablo and we have successfully driven a Qume Sprint terminal at 1200 baud with CICERO output coded for a Diablo. At present CICERO supports only those features of the Qume Sprint which it has in common with the Diablo.

11.1. Controlling the Pitch or Character Spacing

The /FSPERSPACE switch is used to control the horizontal spacing per character. The argument is the number of horizontal units (1/120 inches) per character. Thus,

```
./FSPERS:12
```

sets up a spacing of 12 horizontal units per character (10 pitch) whilst

```
./FSPERS:10
```

sets up a spacing of 10 horizontal units per character (12 pitch). These two pitches are the most common, although, at the time of writing, Qume make one 15 pitch wheel and a series of proportionally spacing wheels. Qume and Diablo wheels are interchangeable. Of the more common typefaces, Courier and Pica are 10 pitch, whilst Elite is 12 pitch.

The default setting of the /FSPERS switch is 1. If the switch has a value greater than one, output containing the appropriate escape sequences to drive a Diablo printer will be produced. Note that both the Diablo and Qume printers can print files created with the default setting of /FSPERS:1 but the output will contain no escape sequences. Therefore, the pitch will be determined by the switch setting on the printer or by the printing program. Also the interword spacing will be integer multiples of the character spacing. Specification of a larger value of the /FSPERS switch argument gives access to the extended capabilities of these printers and allows better justification.

Under normal circumstances, the following switches have their arguments expressed in character spaces and when converting to internal units, the new value is multiplied by the argument of the /FSPERS switch:

```
./GUTTER   to set the intercolumn margin.

./INDIND   to set the indentation for index continuation lines.

./INDSUB   to set the indentation for index sub-levels.

./LINE     to set the line length.

./MARGIN   to set the left margin.

./PRIND    to set the indentation of the page reference line.

./TABSIZ   to set the spacing of uniform tab stops.

./TOCIND   to set the indentation of continuation lines in the table
           of contents.

./TOCNUM   to set the space reserved for page numbers in the contents.
```

By normal circumstances, is meant that the previous argument supplied for the /FSPERS switch was non-zero. When a zero argument is supplied for the /FSPERS switch the arguments of any of the above switches that follow are interpreted as horizontal units (1/120 of an inch). Then, for instance, to set an intercolumn margin of 3-1/2 spaces for Elite typescript one might enter,

```
./FSPERS:0 /GUTTER:35 /FSPERS:10
```

Provided neither the new setting nor the previous setting was zero, specification of a new value for the /FSPERS switch scales the values set by all the above switches

11.2. Controlling the Vertical Spacing

The /LFLEAD switch does not itself control the vertical spacing. Its value is used in various otherways and it is important that it be specified. Primarily, it is a measure of character height. It is also used in the absence of more specific information to calculate the offset for underlines specified by the /UFLF and /ULF switch. Its value is used as a multiplying factor when processing arguments for those switches that specify vertical spacing in terms of numbers of lines. These are:

./PAGE that sets the logical page length.

./RPAGE that sets the physical page length.

./FIG that reserves space for a figure.

When assessing whether the line spacing is sufficient to clear subscripts from the previous line or superscripts on the present line, the argument of the /LFLEAD switch is used as a measure of character height. This interpretation is adopted also when spacing the first line on a page or after a figure. The /UFLF and /ULF switches establish a mode of underlining requiring paper advance before the underline is set. The amount of this advance may be specified through the switches or it may be defaulted. The default value is determined by the setting of the /LFLEAD switch. At present the program provides no means of telling whether the default value was used so it cannot modify the value upon a subsequent declaration of the /LFLEAD switch. Therefore, it is important to precede the declaration of these switches with the /LFLEAD switch.

The /SPACING switch is used to control the vertical spacing directly. Frequently, the /LFLEAD and /SPACING switches have the same argument but this is not necessarily so. For instance, one might choose to set with a somewhat larger spacing to clear super and subscripts or to facilitate correction. It would be quite reasonable to change the /SPACING switch several times during processing of a file but there is unlikely to be any case where one would need to change the /LFLEAD switch setting. The argument for both these switches is interpreted in vertical units (1/48 of an inch). Thus,

./LFLEAD:7 /SPACING:14

would set lines spaced by 14 vertical units and give a character height of 7 vertical units. A spacing of 7 vertical units is just tolerable for an Elite typeface but for the 10 pitch faces a spacing of 8 vertical units (6 lines per inch) is required. The default setting for the /LFLEAD switch is 1, the right value for non-fractionally spacing printers, and any greater value causes code containing escape sequences to be generated for the Diablo or Qume.

11.3. Controlling Blank Space

When printing it is rare to increase the spacing for a new paragraph by more than about 4 points and no increase in spacing is now common. Since most printed text is lower case and the overall density of ascenders and descenders is quite small, the impression of a blank line can be created whilst spacing by very much less. This is a technique which may usefully be carried over into typescript whenever the output device is capable of doing it. Tighter setting not only keeps the text looking neat, it also saves paper.

The interparagraph spacing increment can be set with the /PARASP switch, whilst the spacing for blank lines caused by a double CRLF or the first blank line of free text may be set with the /BLANKSP switch. Both switches

take an argument specified in vertical units (1/48 inch). For an Elite type wheel where tight setting is required, these values could be used,

```
./PARASP:3 /BLANKSP:3
```

to set an inter-paragraph spacing and a blank line spacing of 3 vertical units. Whilst for a 10 pitch wheel where pleasing appearance is probably more important than economy of space,

```
./PARASP:4 /BLANKSP:6
```

sets acceptable values of 4 and 6 vertical units respectively.

When tabulating records (see section 6.4) the space between tabulated records may be controlled with the /TSPACING switch. For instance,

```
./TSPACE:3
```

would set an inter-record spacing of 3 vertical units.

The normal page format also contains 2 blank lines, a page reference line and a page number line. When the output is to be printed on a Diablo printer the arguments of the three switches that determine the spacing for these lines should be specified in the appropriate vertical units (1/48 of an inch). For instance,

```
./PNO,8     for the page number line.

./PRD:!,8   for the page reference line.

./XSP:4     for the blank lines at the top and bottom of the page.
```

Note that in the first two examples the value is separated with a "," because it is optional. Actually, the format is intended to be compatible with that for the underline switch where a comma must be used, as another optional argument is preceded by a colon. In the last case the sole function of this switch is to supply this value. Therefore, the conventional separator, a colon, is used.

11.4. Controlling Justification

Just as it is quite easy to recognise a blank line as such even though its spacing is appreciably less than the mean line spacing, so it is possible to recognise an inter word space that is significantly less than a full character width. Since the process of justification expands blank spaces, an appreciable saving is possible if this process starts from the smallest acceptable interword space width. When a line is output unjustified, for instance, the last line of a paragraph, the spaces will be expanded to a mean value. Both these widths can be controlled through switches. Thus, for an Elite or 12 pitch type one might set,

```
./SPBMIN:7 /SPBMEAN:10
```

whilst acceptable values for a 10 pitch wheel would be set by,

```
./SPBMIN:8 /SPBMEAN:12
```

In both cases the mean value was set equal to the normal character spacing. The units of measurement are the normal ones for horizontal measurements with the Diablo or Qume (1/120 of an inch).

An interesting logical problem occurs when setting a quadded line (unjustified line) if the width is under the line length when the minimum space width is set, but over length when the spaces are expanded to have the mean value. In this case CICERO will expand the spacebands uniformly to the

greatest width that the line remains underset. The small remainder of surplus space is then added to the right end of the line. In contrast, when photosetting, the V-I-P in fact justifies such a line.

When setting shiftable text with a ragged right margin (/NOJUST switch set on) the mean value is used for the space width.

11.5. Changing the Ribbon Colour

Ribbon colour can be changed by using the predefined macros <RED> and <BLACK>. Naturally, these only work if a two colour ribbon is mounted in the Diablo printer. Apart from accounting applications, this technique can be put to use for interpreting font changes in drafts of text which are intended subsequently to be photoset. Suppose a special macro delimiter is defined through,

```
./SPMACRO:`:<RED><BLACK>
```

then text delimited by "`" will be printed in red on the listing. When subsequently the text is prepared for photosetting the special macro definition might be changed (perhaps in a default file rather than the actual text) to

```
./SPMACRO:`:<ITALIC><ROMAN>
```

where <ITALIC> and <ROMAN> are user defined macros that generate the appropriate code to change typeface.

11.6. Changing Typeface

It is not easy to change type wheel mid way through a report printed with the Diablo 1610/1620 printer. It would be easier if moving the wheel system back to gain access to the typewheel disabled the servo control or if the system provided some means to do this under program control. Unfortunately, it does not, so for the safety of both operator and servo system it is necessary to turn the power off. That clears the microprocessor memory in the printer. Once the wheel has been changed, the program can retransmit the position on the line and the vertical and horizontal motion indexes for character and line spacing. What it cannot do is reset the top of form.

Since there are no bold or italic typefaces available, it seems likely that the only reason one would wish to change typeface mid way through a report is to print greek letters or mathematical symbols. These are provided only on a General Scientific wheel. Consequently, it is unlikely that there will be a requirement for more than two typefaces. In view of the difficulty involved in changing wheel a simple solution to the problem of printing reports containing mathematical symbols is to code the change of typeface in the output file and process this with a multi-pass program, each pass being done with a particular typewheel. All printing text not in the right font is replaced by spaces. Actually, lines that are blank need not be printed and therefore the pass with the General Scientific wheel should be quite fast. A sensible way to code the selection of typeface is by using the ribbon colour codes. This allows the output of CICERO to be printed in one pass for checking purposes. Then when proved right, it can be set using the two pass program. To replace the typeface control with ribbon colour change information in the output the /COLOURF switch should be used. The multi-pass printing program is described in chapter 17.

11.7. Simulating Bold Printing

Bold text can be simulated on the Diablo by fractionally spacing and overprinting once or twice. Typographically, the results vary in efficacy. With a sans serif typeface, eg. Qume's Letter Gothic, triple overprinting

is quite legible. With a serif typeface, it is not so good especially for lower case characters. Small "m" is probably the severest test. Provision is made for control of this facility via three predefined macros causing the specified effect on all following characters:

<TNRM> - restore normal printing
<TBLD> - **one offset overprint**
<TXBLD> - **two offset overprints**

11.8. Adjusting Multiple Columns

When setting multiple columns, contiguity conditions will often cause a few blank lines at the bottom of each column. Multiple column printing looks nicer if the columns are adjusted so that the bottom lines align. The Diablo is capable of sufficiently fine vertical spacing to distribute this amongst the blank lines that occur between paragraphs and before headings. This action is automatic. However, occasionally, a very long contiguous construction just fails to fit so the columns are of markedly different lengths. Therefore, the user is allowed to specify a maximum limit to the increase in interparagraph spacing. When distributing the extra blank space, it is divided into a uniform increment and a remainder. The remainder is distributed as one vertical unit per blank line from the top until it is used up. The /MAXADDLEAD switch limits the value of the uniform increment. Thus an argument of zero will not stop column adjustment so long as the number of blank lines is sufficient to accomodate the excess space without increasing the spacing of each by more than one vertical unit.

11.9. Setting the Line Characteristics for the Diablo 1610/1620 or Qume Sprint 5 Printers

When transmitting data to terminals, the communications software of most computer systems may perform various special actions. For instance, fill characters, usually the ASCII code 177 (octal), may be added after carriage returns, line feeds etc. The system may count the spacing characters in a line and insert extra CRLFs if the output line exceeds the terminals line length. Finally, it may substitute multiple spaces for tabs and line feeds for form feeds. All of these features are often useful and sometimes essential. Unfortunately, they cause trouble for the Diablo. Most of the special functions the Diablo printer performs are coded as two or three character escape sequences (the first character is an escape or altmode character). The three character sequences uses any character in the ASCII range other than NUL and DEL (codes 0 and 177 octal). Furthermore, if fill characters occur in an escape sequence, the output seems to go wrong. In both these respects, the Diablo control code is probably at variance with the ASCII standard. What these irregularities mean is that it is very important to set up the communications software so as:

1. Not to add fill characters at all since any character that normally requires filling may be part of an escape sequence. For the DECsystem-10 the command TTY NO FILL will do this.

2. Not to translate tabs to multiple spaces since the tab may be an argument or a qualifier in an escape sequence. For the DECsystem-10 TTY TABS will stop translation of tabs. Because the hardware tabs on the Diablo are set up in the memory of its microprocessor and are therefore lost whenever the terminal is turned off, it is quite likely that the default action of the software will be to assume it has no hardware tabs.

3. Not to translate form feeds into multiple line feeds for the same reasons. For the DECsystem-10 this can be done with the TTY FORM command.

4. Not to append free CRLFs to the output when the line becomes overlength. For the DECsystem-10 this can be suppressed by setting TTY NO CRLF.

The reason for the last requirement is that the communications software has no means of knowing the length of the escape sequences and therefore cannot tell whether a character is a printing one or part of an escape sequence.

CICERO will set up the line and character spacing automatically for the Diablo or Qume printers if the /LFLEAD and /FSPERS switches are specified. If the user does not define these switches the actual spacing that results will depend on what the printer was doing last. On the 1620 or Qume Sprint terminals they can be set by entering the appropriate escape sequence from the keyboard with the terminal in local mode. With the 1610 printer one should set the spacing with the switch inside the cover and press the general clear button.

CHAPTER 12

Facilities for Phototypesetting

In comparison with the output devices considered so far, phototypesetters provide three additional capabilities:

1. Variable letter spacing. That is an "i" and an "m" do not have to occupy the same width.

2. A large range of character sizes, eg from 6 to 72 point.

3. A range of typefaces interchangable by program control.

It is the purpose of this chapter to describe the control of these facilities. Because many readers will be unfamiliar with typesetting terminology, the following two chapters (chapter 13 and 14) will describe this and how to achieve good typography using CICERO.

Besides the control of these new capabilities, CICERO has to perform various other functions when photosetting. In particular, the character set available for a particular typeface will not generally be the same as for ASCII. Therefore, it must provide a mapping scheme for conversion between the two. Part of the mapping scheme must be automatic ligature conversion. The design of some typefaces requires some combinations of letters to be printed together as a single symbol. The most common ligatures are fi, fl, ff, ffi and ffl.

It is quite probable that many users of CICERO will not own a phototypesetter themselves but instead will arrange to supply preformatted input on paper or magnetic tape to an external typesetter. For them it is very important to be able to list the photoset output on a conventional hard copy device. CICERO provides facilities for generating such a listing as the photosetter input data is being generated.

Finally, large galleys of photoset output are inconvenient to handle. A galley is the name given to a single piece of output on a phototypesetter. Originally, it was a large tray used to hold the lead type produced by the setting process. When input to the phototypesetter is by means of paper tape, it is also inconvenient to handle very large lengths of tape. CICERO provides a facility for controlling the lengths of galleys.

12.1. The /ASSIGNFONT switch

The /ASSIGNFONT switch provides a means for specifying the physical location on the phototypesetter of a particular typeface identified in the input file by a logical name. In addition, it also reads the width table for the typeface. The general format for the /ASSIGNFONT switch is

```
./ASSIGNF[logical name]physical location
width table.
```

where "[" and "]" are the font name delimiters defined through the /FONT switch. The format for the physical location specification and the width tables will depend on the particular phototypesetter used. For the V-I-P they are described in chapter 16.

It is fairly obvious that the autoligature substitution needs to be done on a font basis. For example, Times Roman is a typeface that does require ligatures particularly for the fi and fl letter combinations. However, the mathematical fonts designed to be compatible with this typeface on the V-I-P use the ligature positions for other signs. For instance, in one layout the f and l positions are occupied by the Greek letters phi and

lambda whilst the fl ligature has a ">" sign. One clearly needs to be able to operate automatic ligature substitution when printing simple text but to turn it off when setting with the mathematical fonts.

Automatic ligature substitution now works on a typeface basis through the /ASSIGNFONT switch. Thus

```
./ASSIGNF:AUTOLIG[logical name]physical location
width table
```

or

```
./ASSIGNF:NOAUTOLIG[logical name]physical location
width table
```

now define the setting of this flag for a particular typeface. The default action will be no automatic ligature substitution. In specifying the arguments for the /ASSIGNF switch, AUTOLIG and NOAUTOLIG may be abbreviated to "A" and "N" so long as no new options are added. In any case all that will be required is that the option name abbreviation should remain unique. In either implementation ligature substitution occurs after the line division and hyphenation decisions have been made.

12.2. Changing Character Size

Character size changes are implemented in basically the same way as changes of typeface. The /SIZEDELIMIT switch is used to declare a pair of delimiters for character size information. Thus

```
./SIZE:()
```

declares "(" and ")" as the size delimiters. Then in the input text any number enclosed within these delimiters is taken as a specification of character size. The normal unit of size is printers points. What is measured is the minimum spacing between lines that does not cause overprinting. This is also the size of the EM of the character set. CICERO has a table of the typesizes available on the photosetter and uses this when processing a size specification. A request for an unavailable size will be met with the size next lower or the minimum size. For instance,

```
(13 point)Heading: (11 point)following text
```

would be set with "Heading: " in 12 point for a V-I-P which does not set 13 point characters but the following text would be set in 11 point. Note that the comment after the number is ignored but may help people work out what is going on.

12.3. Supershift and Character Conversion

The output to drive the phototypesetter will normally be coded in some variant of the TTS code (see the next chapter). In converting from ASCII to TTS some transforms are obvious. Thus, one will map upper and lower case alphabetic characters and numbers. Unfortunately, the selection of the other characters (PI characters) differs. Of necessity most typefaces will have the punctuation marks .,-(necessary in order to hyphenate);&?:!() and even $. Also the underline can reasonably be mapped into an EM or EN rule character. However, the mapping of the other characters may vary. In particular many typefaces will have explicit fractions (eg. 1/2, 1/4 etc) as well as a selection of printers marks. Mostly, they do not have mathematical operators (not even + or =). The best thing to do therefore seems to be to provide a standard mapping that gives access to all the characters available. Such a mapping is given for the V-I-P in chapter 16.

Most TTS keyboards have a supershift key which gives access to an extended character set. For the V-I-P the supershift characters are an additional set on the numbers and the ligatures. Because the former are usually numbers too (eg. superscripts) there is some point to carrying the convention through into the coding for CICERO. Thus a /SUPERSHIFT switch defines a supershift operator.

```
./SUPERS:^
```

defines "^" as the supershift operator. When this operator precedes a number a supershift character is generated for the V-I-P. Versions of CICERO intended for other phototypesetters might possibly use supershift for other characters than the numerals. The ligatures, supershift characters on some TTS keyboards, are mapped through characters in the ASCII set that are not normally available in common typefaces. In this way, the listing of a photoset output (see section 12.5) containing ligatures set by automatic ligature substitution will have these expanded but where the ligature is intended as a special character it will be listed as the ASCII mapping character.

12.4. Indenting New Paragraphs

When photosetting, the original intention was to specify paragraph indentation with the /PARINDENT switch already described. However,an indentation of 1 EM space is commonly used. Normally, substitute spaces translate to EN spaces when photosetting. Unfortunately, the width of an EN space is liable to vary with typeface, since it is the width of the numerals and two EN spaces do not necessarily add up to an EM space. To accomodate these variations the specification of the /PARINDENT switch is changed for photosetting and now sets the indentation in units of 1/2 EM spaces. Further, three other switches have been introduced. The /EMPARAGRAPH switch, cancelled by the /NOEMPARAG switch, causes the indentation to adjust automatically to the line measure depending on some rule. The rule implemented as a default happens to be that recommended for Australian Covernment publications and is,

1 EM space for measures up to 26 picas,
1-1/2 EM spaces for measures between 27 and 35 picas
2 EM spaces for lines longer than that.

The break points are determined by the /EMLIMITS switch which takes two arguments. The first, is the maximum line length in picas for EM indentation of new paragraphs, the second is the maximum line length for 1-1/2 EM indentation. The default setting of this switch is /EMLIMITS:26,35. Whilst, this description may seem complicated, what it really amounts to is that specification of the /EMPARA switch by itself will result in an appropriate indentation for the first line of new paragraphs regardless of the line measure.

12.5. Listing Photosetter Output

The easiest way to produce a readable listing of the photoset output is by generating the listing at the same time as the output for the phototypesetter. Such a listing is useful to show:

1. The division into lines of the photoset output. When headings are set in shiftable text, this is particularly useful for showing those that are overlength and flow on to the next line. It can also be used to check hyphenation.

2. If the listing output is produced on a Diablo 1610/1620 printer or similar fractionally spacing printer, the listing can reproduce the line spacing by a suitable conversion of leading points to vertical

fractional spacing on the listing device.

3. It can show changes of typeface by appending messages in the listing output or if the listing device is capable of it by stopping and allowing the user to change golf ball or typewheel. If there are only two typefaces used changes can be shown through ribbon colour.

These facilities are controlled by the /LISTLEADIN, /GBMSG and /COLOURFONT switches where the action is defined as follows:

./LISTLEAD:n

sets a vertical spacing of 1 vertical unit for the listing device to every n points on the phototypesetter. The /LISTLEAD switch is one of those (see section 12.6) for which the argument is in points and decimal points. Thus if the listing device spaces vertically in units of 1/48 inch

./LISTLEAD:1.5

would set an average spacing on the listing device equal to that on the phototypesetter at the expense of some slight loss of accuracy. Equally, when the listing device is a line printer the argument for the /LISTLEAD switch might be the mean spacing specified for most of the text. Then the listing would no longer reflect the line spacing of the output but it would still be usable to show the line division.

The /GBMSG switch may be used to direct the font change messages by varying its argument as follows:

./GBMSG:TTY sends messages to the terminal from which the job is being run. The user is required to type a carriage return when the appropriate golf ball or typewheel is mounted on the list device.

./GBMSG:LST sends the font descriptor part of the message to the listing output. Unfortunately, this distorts the format of the list output but at least the information is all there. The program does not stop to wait for the user to change golf ball or typewheel.

./GBMSG:NUL suppresses all font change messages.

As suggested by their name GBMSGs are really intended for use when the user can change the listing device typeface easily, for instance, when it is a Selectric typewriter.

The /COLOURFONT switch was described in the last chapter. It causes the first typeface defined to print in black and the second in red for the list device. If more typefaces are defined they will be processed for the list device depending upon the setting of the /GBMSG switch.

12.6. Arguments for Switches Specifying Vertical and Horizontal Measures

In previous chapters, the appropriate forms of switch specification have been given together for conventional hardcopy output devices and for phototypesetting. However, it is useful to summarise the information again here. The division of the switches into classes is then immediately apparent and easily remembered. This division is reflected in the implementation of CICERO by the procedure used to decode the argument.

The following switches set horizontal spacings and for photosetting the argument is specified in picas and points, a dot being used to separate the two. If the value is an exact number of picas the dot can be omitted:

./GUTTER to set the intercolumn margin width.

./INDINDENT to set the indentation for continuation lines in the index.

./INDSUB to set the indentation per sub-level in the index.

./LINE to set the line length.

./MARGINSIZE to set the left margin.

./PRINDENT to set the indentation of a page reference line.

./TABSIZE to set the spacing of uniformly spaced tabulation columns.

./TABULATE to set the width of tabulation columns and gutters for tabulated records.

./TOCINDENT to set the indentation for continuation lines in the contents.

./TOCNOS to set the space reserved for page numbers in the table of contents.

Of these switches, three are special. The /LINE switch may take a second argument which is the line length in characters for the listing device. Thus,

```
./LINE:34,87
```

sets a line length of exactly 34 picas for the phototypesetter and a length of 87 characters for the listing device. The /MARGINSIZE switch can take a second argument also in picas and points which is the margin size for even numbered pages if different to that specified by the first argument which is for odd numbered pages. Thus,

```
./MARGIN:1.4,0
```

sets a margin of 1 pica 4 points for odd numbered pages only. Finally, the /TABULATE switch can take a large number of arguments to specify the line and gutter widths for records output in tabulated columns (see section 6.4). Thus,

```
./TABULATE:4.7,1.2,11,1.2,11
```

would set up 3 tabulated columns of widths 4 picas 7 points, 11 picas and 11 picas, separated by gutters of width 1 pica 2 points.

For the control of vertical spacing the switches that set large values are specified in picas and points whilst those that set normally small values are specified in points and decimal points. Whilst, this may sound potentially confusing, in practice, it is not. There are only three switches that can set large vertical spacings and they are:

./PAGE to set the logical page length.

./RPAGE to set the physical page length. It is most unlikely that this switch would be used when photosetting.

./FIG or /FIGURE to set figure height.

All these take as argument a height expressed in picas and points, the points being optional and if present separated from the pica field by a dot.

The following switches all specify vertical spacing in points and a decimal point may be used to express tenths of a point. For many phototypesetters the minimum spacing increment is half a point so the figure after the decimal point should be five or zero:

./BLANKSP	to set the spacing of a blank line output by a double CRLF in shiftable text or at the start of free text.
./LISTLEAD	to specify the photosetter film advance for unit vertical spacing on the list device.
./MAXADDLEAD	to specify the maximum allowable addition of space to blank lines when adjusting the lengths of multiple columns on a page prior to setting.
./PARASP	to specify the interparagraph spacing.
./SPACING	to specify the film advance for a new line.
./TSPACING	to specify the inter-record spacing for tabulated records.
./XSP	to specify the spacing for the two blank lines, one at the top and one at the bottom of the page.

Thus,

```
./SPACING:9.5
```

sets up a line spacing of 9-1/2 points whereas,

```
./PAGE:49.5
```

sets up a page length of 49 picas 5 points.

The following three switches take an optional argument specifying the film advance in points. In all cases the argument is separated by a "," instead of a ":",

./FND	to set the line spacing for footnotes.
./PNO	to set the line spacing for the page number line (also turns page numbering on).
./PRD	to set the film advance for the page reference line.

For example,

```
./FND:",9.5(9)
```

sets a film advance of 9.5 points for footnote lines, a character size of 9 points and defines " as the footnote delimiter. Note that all these switches do other things besides specifying a line spacing and for this reason the argument is optional.

12.7. Setting the Spaceband Parameters

Essentially, these are the same controls as were required for the Diablo printers except that the arguments are now in different units. For photosetting all character widths are relative to the EM of the character set. Many typesetters employ unit width of 1/18 of an EM. For these one

might choose a minimum spaceband width of 4 units and a mean spaceband width of 8 units. These values could be set by

```
./SPBMIN:4 /SPBMEAN:8
```

When setting the widths of all characters, including spaces are scaled depending on the current point size.

The V-I-P will also perform letter spacing optionally if the spaceband becomes too big. However, this feature is not implemented through CICERO. Typographically, it is not universally regarded as good practice and is only really necessary for very short lines such as those that can occur at the sides of figures.

12.8. The /GALLEY Switch

Processing very long runs on the phototypesetter is not sensible since a variety of simple human errors can cause expensive wastage. Further, if the input to the setter is via paper tape, long lengths are extremely cumbersome to handle. Therefore, most phototypesetting is done with relatively short galleys of a few pages each. The length of the galleys may best be specified as a number of pages. The DECsystem-10 implementation generates file names with numeric extensions for these galleys. Each new galley file name is typed on the terminal controlling the job as that file is created and the galley number is entered on the listing output as well. Using this facility, one could for instance, do the logical processing for a whole book in an overnight batch run on the computer. The next day one could set the output galley by galley. To set the galley size, the /GALLEY switch is used. Thus,

```
./GALLEY:4
```

sets a 4 page galley size. The last galley will obviously be less than this in most cases.

12.9. Kerning

Kerning has not yet been implemented through CICERO. Kerning is a small and temporary reduction of letter spacing used to make headings that do not quite fit on a line do so or to remove widow lines. Whilst it would not be difficult to arrange to specify kerning by switch, the general philosophy of CICERO is to avoid this type of manual intervention and to aim for an automatic system.

CHAPTER 13

Typesetting Terminology

The purpose of this chapter is to give a very brief introduction to typesetting terminology and measurement systems.

13.1. Historical Introduction

Modern typesetting has developed from some five centuries of thought and experience. Some of the old ways still persist and certainly much of the terminology has its roots lost in the past. Consequently, an historical introduction is quite appropriate.

The first typesetting was done by assembling individual pieces of movable type by hand. The type was made from lead not only because it was easy to melt but also because it flowed well and was capable of reproducing the intricate shapes required. Additional lead was used to increase the line spacing, hence the term "leading" still used for film advance in phototypesetters. The origin of the terms "upper" and "lower case" is in this period and refers to the cases in which printers kept the type. The letter shapes are, of course, much earlier in origin. Upper case letters are designed with straight lines because they come from a representation of the Roman alphabet intended to be carved in stone. Lower case characters have flowing curves because their shapes are derived from handwriting. The variations in width characteristic of gothic script are those that result when writing with a broad nibbed quill pen.

In the latter part of the nineteenth century hot metal typesetting was introduced. With considerable mechanical ingenuity these machines actually cast the type as it was required using brass moulds. There were two basic forms of machine called Linotype and Monotype. The former were used mostly for newspapers, the latter for book work. The significant feature of both machines was that character selection was done by an operator pressing keys on a keyboard. Obviously, this is much faster than searching through trays of type. Both kinds of machine are still extensively used today but as far as the authors are aware no new machines are being built.

In 1932 a system called teletypesetting was introduced for the remote operation of linotype machines by paper tape. This system is still used for controlling phototypesetters so it is of some significance. The six level code, called TTS (TeleTypeSetting), is based on the five level Murray code used for telex but with extensions to provide the control codes for the linotype setter. The Murray code was designed to minimise punch wear by punching fewest holes for the most commonly used characters and this feature is carried through into TTS. Thus the code for "E" is 02. If a code is only used for data transmission minimising punch wear is a reasonable objective. However, such a code does not order the characters of the alphabet in a rational sequence, nor does it provide a simple sequence for numerals. These are useful properties and necessary for a code like ASCII that is used not only for data transmission but also for internal representation of characters within a computer. Another interesting property of six level TTS tape is that it is symmetric with the feed holes in the middle, which means that potentially there are four ways of getting it into the reader only one of which is right. To improve the odds, TTS tape has the feed holes advanced relative to the data frame. That means it is a non-standard tape requiring special punches and readers not normally found on computer systems. Some phototypesetters do accept eight level TTS encoded tape (the V-I-P is one). For such tapes the first six channels carry the

TTS code. Channel 7 is used for parity and channel 8 is punched only for rub out.

Implied in the definition of TTS is an operating strategy for the linotype machine. Essentially, this is to transmit and hold the characters for a line, then what to do with the line (justify or "quad" it) and finally, the instruction to set the line ("elevate"). A quadded line is set left, right or centred without justification depending on which function was specified. In the absence of a quad code the line is justified. This operating strategy evidently implies storage in the setter. On the linotype machine, that storage was a mechanical register into which the brass matrices or character moulds were switched. On a modern phototypesetter it is a buffer in a mini or microcomputer. A small problem for CICERO is the need to perform efficiently, operations like justification and quadding on devices that work like a linotype machine as well as conventional electric typewriters which require no internal storage.

Conventional linotype machines provide two complete typefaces in one point size. Usually, one of the typefaces would be a medium Roman style whilst the other would be a bold or italic face. The first phototypesetters were an improvement as they provided a range of typesizes on the one machine. However, it soon became obvious that it would be useful and not very expensive to provide a wider range of typestyles too. Thus modern machines can set almost all work on one machine. Doing that puts a greater premium on speed. Whilst the early phototypesetters used opto-mechanical means to select, magnify and position characters newer machines perform these functions electronically outputting the final character by means of a raster scan on a high definition CRT.

Although not strictly relevant to typesetting, it is probably worth stating that most modern printing does not depend on inking a raised surface but on making the printing plate sticky to ink by chemical means. This is a process that may readily be done by photochemical treatment. Hence, it does not matter that the end product of a phototypesetter is not raised type.

13.2. Measurement Systems

In Britain and America, the normal units of measurement used by printers are picas and points. A point is approximately 1/72 inch and a pica is 12 points or approximately 1/6 inch. Surprisingly, there are slight discrepancies between different manufacturers. However, to reasonable accuracy

1 pica = 0.1660 inches = 4.21638 mm.

1 point = 0.01383 inches = 0.351365 mm.

Continental Europe uses a different system based on the Didot Point (0.376 milimetres) and the Cicero (12 points didot make a cicero (4.512 mm.). There is nothing particularly metric about this unit as it is in fact derived from the pre-metric French foot.

Sometimes line widths are expressed in EMs. An EM space has width equal to the set size or height of the typeface. Therefore, a line length expressed in EMs is relative to the point size used.

Character widths are also related to the EM of the character set. It is possible to design a reasonable typeface on the basis of an 18 unit EM though some phototypesetters provide finer width specification. The 18 unit EM is very common, particularly for newspaper work.

13.3. Fixed Spaces

Apart from the space used for justification there is a need for spaces of predictable width. The normal practice is to provide three such spaces as follows:

1. An EM space having width equal to the set height. Such a space is often used to indent new paragraphs (see the /EMPARAGRAP switch of chapter 12).

2. An EN space has width about half that of an EM space though that varies with the typeface design. It is the width of the numerals and can be used to justify them.

3. A thin space has the same width as a dot. This is about half the width of an EN space though that varies with the typeface design. CICERO appends a thin space to dots used as full stops unless the dot is the last character of a line or was preceded by the literal character in the source. One might wish to suppress the increased spacing for a dot that signifies an abbreviation. This use of a thin space is a compromise between the old tradition of french spacing used in typing (double spacing after a full stop) and the modern practice of no extra spacing.

In addition many phototypesetters provide a unit space and a unit backspace, the unit being that used to express character widths.

When printing large numbers it is no longer the practice to separate thousands with a comma since some countries use a comma as a decimal point. The modern practice is to use a space and for an 18 unit EM a three unit space is about right.

Fixed spaces can be specified to CICERO using the predefined macro facility. Thus if "<" and ">" are the macro delimiters,

<EMSP> generates an EM space.

<ENSP> generates an EN space.

<TSP> generates a thin space.

<VTSP> generates a unit space.

<VTBS> generates a unit backspace.

<3USP> generates a three unit space.

The substitute space is also translated into an EN space. These expressions could reasonably be called verbose. However, CICERO rarely requires them as it will generate fixed spaces automatically in most of the situations where they are required, for instance when tabbing across to tabstops or when indenting the left margin to align it with some previously defined point set with the <LM> macro.

Occasionally, it is necessary to set a left margin indentation of a predefined value rather than to align it with a particular position in the previous line. For instance this might be the case if Roman numerals were used to number paragraphs since these have differing widths. For these cases the predefined macros:

<LEMSP> indent by an EM space.

<LENSP> indent by an EN space.

<LTSP> indent by a thin space.

<LVTSP> indent by a unit space.

<LVTBS> decrease the indentation by a unit space.

<L3USP> indent by a three unit space.

may be used either singly or in combination or multiply. For instance,

<LEMSP*2><LVTBS>

would set an indentation of 35 units (based on an 18 unit EM).

CHAPTER 14

Good Typography

Typography is a graphic art and as such some things are a matter of taste even fashion rather than rigorous logic. Nevertheless, the objectives remain substantially the same. One is trying to present the printed information in a way in which it will be read. To ensure that, it must be visually attractive, easy to read and not look too long.

Most organisations that produce printed matter have a style guide or manual with which publications are expected to conform. The Australian Government Publishing Service publishes a very wide range of typeset material. Consequently, its style manual is worth reading. It covers topics from spelling and punctuation to printing and binding. Because many copies of this manual are sold it is available in an inexpensive paper back binding. ("Style manual for authors, editors and printers of Australian Government publications", AGPS, Canberra, 1978 (ISBN 0 642 03345 5)).

Government publications should clearly comply with the style rules presented in that style guide. The following brief notes were written prior to publication of the Style Guide. Whilst they do not conflict with what is contained in the Style Guide the latter is a much fuller explanation of modern practice and should really be read by anyone contemplating using CICERO for photosetting.

14.1. Do not Waste Space

Space is a very valuable resource. If the final output looks long the potential reader may not even bother to start. Therefore, make the text look short. In particular,

1. Avoid excessive space before headings and between paragraphs. The impression of a blank line can be created by spacing by much less. For instance, when setting in 10 point, 4 points additional spacing is sufficient to indicate a new paragraph, 6 points extra spacing for a blank line prior to a heading and to separate a new paragraph from a heading a spacing of 2 points is sufficient. These values may be used as the arguments of the /PARASP and /BLANKSP switches and CICERO will automatically scale them for different point sizes. The reduction to 2 point (or half the normal new paragraph spacing) is automatic for a new paragraph that follows free text or a line started with a double CRLF.

2. Avoid excessive indentation. The points presented in the numbered paragraphs on this page are just as clearly made with the numbers starting in the left margin so why indent them any more? A trend in modern typography is to avoid indentation altogether, relying instead upon changes of typeface to draw attention without wasting space.

3. Do not use excessive line spacing. So long as the lines are reasonably short a line spacing of one point more than the set size is usually quite sufficient. i.e. set 10 point on no more than 11 point spacing. There are reasons that justify greater spacing. For instance, some older styles of typeface are extremely difficult to read when tightly set. If the line length is very long, wider spacing is necessary to facilitate finding the start of the next line. Finally, if the text contains super or subscripts within the main body of text it will be necessary to increase the spacing to accomodate them. CICERO will do that on a local basis if insufficient space has been allowed but it does not look very pretty.

4. Do not use excessively wide gutters between columns. If text is indented a wider gutter is necessary. Otherwise a gutter width of 1 pica is usually sufficient.

Not only do these actions result in shorter looking copy, they also save paper. In turn that saves on weight and therefore postage. All round they save money as well as being ecologically less hard on the trees.

14.2. Make it Easy to Read

Long lines are wearying so in general avoid them. As a rough rule of thumb, if the line has more than about 72 characters (including spaces) it is getting long. Notice how newspapers use very short lines whilst the less palatable clauses of legal contracts are set on very long lines of very small print. If a long line must be used, the reading effort can be reduced by increasing the line spacing judiciously. There is little point in justifying such lines. The object of justification is to make it easier to find the start of the next line. However, it fairly obviously does not work if one has to do a cinemascopic scan to find the start of the next line anyway.

Very long, widely spaced and unjustified lines are currently in vogue for some types of publication. Obviously, one has to balance a few major interruptions caused by line endings against very many minor ones when the text is set to a shorter measure. It is also quite common to set text with a ragged right margin when the objective is to attract attention, as for instance, in advertising copy. CICERO is not really designed for such applications, though the /NOJUST switch allows text to be set with a ragged right margin when required.

Large expanses of bold text tend to assault the reader visually, so use bold text sparingly. The proper role of bold typefaces is to add emphasis. Their over use is the visual equivalent of shouting.

14.3. Add Emphasis Properly

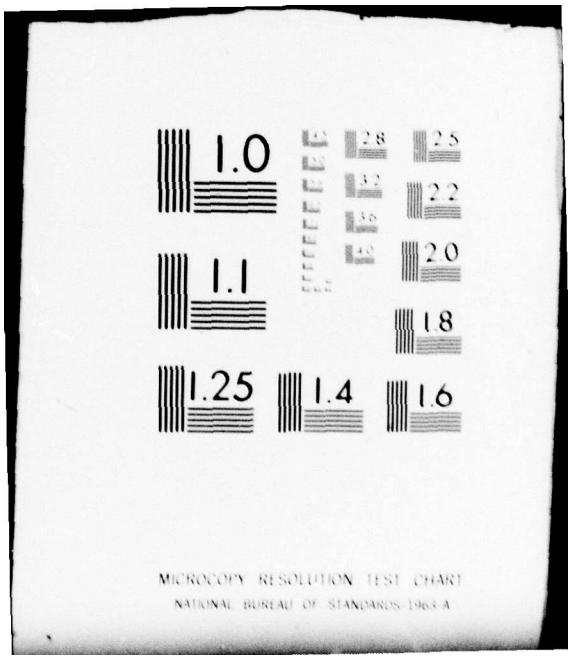
When photosetting, underlining is seldom the way to emphasise. So many other methods are available. Within text italic script is a very gentle way to emphasise, bold script rather less so. For headings bold script may be used by itself or for a major heading with an increase in point size.

When printing headings in a larger point size than the following text, or indeed anywhere where text of different sizes is mixed, it is important to allow space for descenders. The common characters with descenders are g, j, p, q and y. However, for a number of typefaces (for instance Baskerville), capitals J and Q have descenders too. At the present time CICERO does not test explicitly for such clashes. However, generally sensible selection of point sizes and settings for the paragraph and blank line spacing will eliminate such clashes. As a rule of thumb, the line spacing should be greater than the sum of 1/4 the point size of the upper line and 3/4 the point size of the lower line.

AD-A074 399 AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA) F/G 5/2
CICERO TYPESETTING MANUAL,(U)
JUL 79 R C ADAMS, G A CLEAVE
UNCLASSIFIED ARL/STRUC NOTE-453 NL

2 OF 2
AD A074399

END
DATE FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

CHAPTER 15

Setting Mathematical Expressions

Setting mathematical expressions is an expensive process by any conventional means. Not only does mathematics require an extended range of symbols, text must also be set on a multiplicity of base lines. When the photosetter has no reverse leading capability it is necessary to order the base lines of the characters in the correct sequence whilst also aligning the characters horizontally. This error prone task is greatly facilitated by the features described in this chapter.

15.1. Super and Subscript Operators

CICERO allows up to 16 different pairs of up and down operators to be defined, each causing a different motion of the base line. These operators may be used multiply or in conjunction and there is no limit to the level of super or subscripting allowed. Besides subscripts, these operators also permit complex mathematical expressions involving many fractions to be set as one line. The up and down operators are defined through the /SCRIPTOP switch. Thus,

```
./SCRIPTOP:/:\:4
```

defines for the Diablo printer an up operator "/", a down operator "\" and an offset of 4 vertical units (4/48 inches) which gives an offset of approximately half line spacing. When photosetting the same effect would be achieved by

```
./SCRIPTOP:/:\:5
```

which defines the same operators and gives them a base line movement of 5/10 the current point size. Then for instance,

```
y = a\0/ + a\1/x + a\2/x/2\ + ... + a\n/x/n\
```

will set as

$$y = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$$

It is important to return to the original base line at the end of the expression or the resulting offset will persist. Note that when photosetting this implies that when lowering the base line to return from a superscript, the point size current must be the same as when the base line was raised initially.

15.2. Line Spacing to Clear Superscripts

In the presence of sub and superscripts or underlining, the line spacing is defined as the separation of the un-offset base lines. CICERO will increase the line spacing to avoid overprinting in some instances. This happens if the line spacing is bigger than the offset of the top of the highest superscript but less than the sum of that and the lowest base line offset of the previous line. This condition covers most accidental cases but allows mathematics to be set with different symbols on very closely spaced base lines. Some people find this degree of protection against overprinting unnecessarily conservative. They are happy, for instance, to allow superscripts to protrude into the previous line especially, if it was blank. The algorithm to print the page has to process lines in sequence. However, so long as the base lines from different lines do not intermingle this condition can be met whilst still allowing some merging of

superscripts. This lower level of protection is selected by specifying the /CLASHES switch which causes the character height of superscripts to be ignored and /NOCLASHES which restores the normal protective mode.

15.3. Use of Named Tab Stops

Named tab stops were introduced in chapter 10. If "<" and ">" are the macro name delimiters, then

<QT.> sets a tab stop named "QT" at the current horizontal position, whilst

<QT> tabs over to that position either forwards or backwards.

This is useful in conjunction with the super and subscript operators to set fractions. Then, for instance, to set

$$\frac{dv}{dt}$$

one would write,

```
<ST.>/_/dv\_\<ST>\dt/
```

(Where "_" is to be read as the underline delimiter). The call of <ST> generates backspaces to space back to the start. When outputting to the listing device these backspaces are not normally printed because space is accumulated and converted to a single motion for each base line. For photosetting the backspaces print but only for the base line in which they occur.

15.4. Use of Macros for Fractions

The code to generate dv/dt was not difficult to follow but neither was it very readable. Macros provide a simpler way to do the same thing. Suppose a macro <FRAC> is defined by

```
<FRAC(A,B)=<FS.>/_<A>_\<FS><B>>
```

One can easily verify that this macro generates a fraction with the contents of the macro <A> in the numerator and the contents of <B> in the denominator. Note that the position on the line upon completion of the macro is at the end of the macro <B> so care needs to be taken if <B> is shorter than <A>. The macro <FRAC> can be used to generate dv/dt as follows:

```
<DV=/dv\><DT=\dt/>

<FRAC(DV,DT)>
```

which will be found to give

$$\frac{dv}{dt}$$

Further, macros can be strung together to generate more complex expressions. For instance, with

```
<DY=/dy\>
<DX=\dx/>
<WX3=/wx/3\\>
<WL3=/wl/3\\>
<EI=\6EI/>

<FRAC(DY,DX)> = <FRAC(WX3,EI)> - <FRAC(WL3,EI)>
```

gives

$$\frac{dy}{dx} = \frac{wx^3}{6EI} - \frac{wl^3}{6EI}$$

Evidently, by this technique one can go on to build very elaborate expressions.

15.5. Use of Macros for Special Symbols

Another interesting use of macros is to generate special symbols. One might, for instance, generate a sigma sign from slashes and underlines for the Diablo printer. However, this would hardly be the way to do it when photosetting. Then one would probably take the sign from a Greek font and increase the point size for the character. By defining some of these macros in the default file, one can arrange for different definitions to be read depending upon whether one is photosetting or typing the output.

CHAPTER 16

The Mergenthaler V-I-P Phototypesetter

The first implementation of CICERO for phototypesetting was written for an eighteen face base aligning Mergenthaler V-I-P phototypesetter. The purpose of this chapter is to present the features of this machine as they affect a user of CICERO.

16.1. V-I-P Specification

The V-I-P has been a very successful phototypesetter and has been sold in a great many different variations and options. Consequently, it would be prudent to verify the characteristics of the particular model that is available. Currently, three models are being made, all of them base aligning and differing only in the number of typefaces they provide on line (6, 12 or 18). The following specification refers to the 18 face machine that A.R.L. currently uses:

1. Font layout. There are 96 printing characters on a standard typeface arranged in 4 rows of 24 each. The rows are given letters (A, B, C, D) whilst the columns are numbered 1 to 24. A standard mapping of ASCII characters on to this arrangement is given in fig 16.1.

2. Character widths are in units of 1/18 EM. The format for inputting the width data is given in section 16.3. Note that the minimum horizontal spacing increment is actually 1/9 point.

3. Point sizes range from 6 to 72 point but exactly what is provided seems to vary with model. As a zoom lens is used for character magnification, addition of extra sizes at the factory does not present a major manufacturing problem. The range of sizes on the machine A.R.L. uses is

   6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, 33, 36, 42, 48, 54, 60, 72.

4. The maximum line length is 45 picas regardless of point size.

5. The minimum leading increment is 1/2 point. That is the spacing can be set to 9.5 points, for instance but not 9.3.

The V-I-P accepts 6 or 8 level TTS encoded paper tape. The 8 level tape has parity in channel 7 and channel 8 is used only for rub out. CICERO produces output satisfying these requirements.

Q	1	5	9	W	C	J	,	O	E	c	j	e	o	w	q	`	*	<	&	%	^1	^5	^9
K	2	6	0	Y	M	D	.	N	T	m	d	t	n	y	k	'	#	=	?	$	^2	^6	^0
X	3	7	B	G	R	L	S	A	b	r	l	a	s	g	x	-	+	>	:	(	^3	^7	[
Z	4	8	V	P	U	F	H	I	v	u	f	i	h	p	z	;	/	_	!	)	^4	^8	]

Fig 16.1. Standard character mapping on to a V-I-P layout. Note the characters preceded by a "^" are set in supershift and must be preceded by the supershift operator in the input. The ligatures "fl" and "fi" map to the same positions as "[" and "]".

16.2. Standard Character Mapping

The most commonly used typeface layouts for the V-I-P do not have all the ASCII symbols and contain some non-ASCII characters (eg. printers marks and fractions). CICERO incorporates a standard mapping to give access to these characters. In most instances this mapping is what would be expected as it has been designed to minimise the work the user has to do for the standard typefaces. For mathematical symbols, this cannot be the case and some care is necessary to select the appropriate ASCII character that maps to the position of the character in the required font layout.

Mergenthaler divide their font layouts for the V-I-P into three plaques. Plaques 1 and 2 cover columns 1 to 16 and normally contain all the alphabetic characters (upper and lower case), a set of numerals, a comma and a dot. The mapping of these characters from ASCII is, therefore, predetermined. The third plaque contains a selection of symbols (pi characters), the selection varying with typeface. However, the variation is less marked for some positions than others. Columns 17 and 20 contain punctuation marks and the ampersand (&) and their selection is nearly invariant and again determines a standard mapping for ASCII. (position 17C contains the character used for hyphenation so must be fixed). The rightmost three columns, 22, 23 and 24 are most frequently used for a second set of numerals usually either super or subscripted. ASCII has no provision for more than one set of numerals. However, on the standard Mergenthaler keyboard these codes are generated by typing the numbers in supershift. Analagously, a supershift marker (see Section 12.3) can be used to generate the internal representation of these characters by preceeding the ASCII numeral with the operator. Positions 24C and 24D are used for the fl and fi ligatures if the typeface has these. Otherwise, they may be used for any pi character. The other columns in plaque 3 vary, commonly being used for fractions and printers marks. When designing a standard mapping for these positions the range of ASCII mapping characters was intentionally limited to those available in the single case subset used for the Teletype model 33, with the one exception that both " and ` map to position 17A. The mapping chosen for columns 18 and 19 is rather arbitrary but column 21 maps sensibly for the P145 layouts used for Helvetica.

16.3. V-I-P Dependent Part of the /ASSIGNFONT Switch Argument

The physical location of the typeface and the width table parts of the /ASSIGNFONT switch argument are photosetter dependent. For the three drum V-I-P, the physical location is specified by a letter (A, B or C) defining the drum and a number in the range 1 to 6 defining the position on the drum. For a single drum machine, the CICERO program currently requires the specification of drum A. However, no doubt that will be modified in time. Clearly, it is very important to ensure that the physical locations specified by the /ASSIGNFONT switch do correspond with the actual machine set up. There is no way the program can check that.

Width tables are specified in the standard order used by Mergenthaler starting with the width of the character in position 24D and proceding backwards column by column to the width for the character in position 1A and then giving the widths of the thin space, EN space, EM space and EN leader. The true width rather than the 4 bit tape width is required by CICERO. Note that the width table read by CICERO is used by the program for its own calculations and is not transmitted to the V-I-P. When setting the machine up it is necessary to allow it to read the width tapes supplied with each font. This function is not performed by CICERO. Examples of the /ASSIGNFONT switch with arguments appropriate for a three drum V-I-P are shown in Appendix A.

CHAPTER 17

DECsystem-10 Implementation

The purpose of this chapter is to provide information relating solely to the DECsystem-10 implementation of CICERO written at A.R.L. At the time of writing this information applies equally to the version for the DECsystem-20.

17.1. Program Implementation

For the present there are three implementations of the formatting scheme CICERO. These are:

1. CICERO
2. STENO
3. DIDOT

The program CICERO implements all the features described in this manual. It provides these alternative functions:

1. Formatted typescript.
2. Formatted photoset output.
3. Formatted photoset output plus a listing of that output.

The selection of these options will be described later in this section. The program STENO is a cutdown version of CICERO without the photosetter code. This saves on program size and also on runtime because the program never has to test to see which option it is performing. It only performs the first function of formatting typescript. Typescript produced by STENO and CICERO in this mode of operation should be identical. Either program will produce typescript output to drive a Diablo 1610/1620 or Qume Sprint Micro 5 printer as described in chapter 11 or any conventional ASCII coded printer. DIDOT is an even more drastically reduced version of STENO. The features omitted are listed in Appendix C. The main reason for this surgery was to obtain a program capable of running without virtual memory priviliges at A.R.L. The maximum allowable size of physical memory for a program will obviously vary among -10 sites. Consequently, this criterion has no general significance. In the future DIDOT may be developed to provide special facilities of its own. For instance, bidirectional printing is more complicated to program if one has to allow for changes of typeface. The latter facility has been cut out of DIDOT so we might develop the program to print in both directions.

A segmented version of STENO has been generated and is in use at A.R.L. It runs as four segments:

```
GETCOM
MAINDS
STENO (from the source SST.PAS)
DOPAGE
```

The motives for writing this version were:

1. To avoid the need for virtual memory at A.R.L.

2. To make a program that would use minimal core on a KA processor.

This version will do most things in 36K of memory.

The photoset output produced by CICERO is best sent to disk. It is 7 bit TTS with parity on the most significant bit. The operating system handles it as though it were ASCII.

The photoset listing is generally different to that which would be produced for typescript only output. It is not intended to be beautiful but to show the errors in the photoset output before setting it. In particular the line division is determined by the character widths for the phototypesetter. If the listing is coded for the Diablo printer the vertical spacing can reflect that of the phototypesetter which is useful for checking some faults. Also changes of typeface can be shown by changes in ribbon colour. However, the most important errors show up in the line division which can be seen even on a line printer listing.

The program CICERO is called by typing

```
.R CICERO
```

and when loaded it replies by typing an asterisk. The general format for the command string is,

```
*photosetter file spec,listing file spec=input file specs/GO
```

If the photosetter file specification (but not the comma) is omitted, it produces typescript like STENO. If the listing file specification and comma are omitted it produces photoset output only. If both are specified it produces both photoset output and a listing of that output. Note that by specifying the device NUL: for the photoset output one can produce the listing of that output by itself. The photoset output should be the same whether or not a listing file is produced.

File specifications for the DECsystem-10 have the form,

```
DEV:FILE.EXT<PROTECTION>[AREA]
```

where

DEV is the output device or structure name.

FILE is the file name and is up to six characters in length.

EXT is the extension and is up to three characters in length.

PROTECTION is a 3 digit octal number specifying access rights to the file. Protection is only relevant for output files and can normally be omitted when the system default is acceptable.

AREA is a project programmer pair defining the directory in which the file is located. If it is omitted the default action is to use the directory for the current job.

There is only one command string for STENO and DIDOT,

```
*Typescript file specification=input file specifications/GO
```

Note that CICERO requires a leading comma in its command string to perform the same function. The /GO switch enables one to enter commands over several lines. It is also necessary if one is to be able to examine the status before commencing setting (the /LIST switch).

Because some people prefer to use line number oriented editors like SOS and EDIT, CICERO and STENO now strip line numbers automatically. On the 36-bit DEC machines line numbers are indicated by setting bit 35 of the word. The action of the programs is to discard as data any input word with bit 35 set and also skip the next character (a tab). The programs do remember the line number word and those error messages that depend on location in the input file print it out if it is available. The programs also count pages. A new page is indicated by a line number of spaces followed by a form feed. This page number is appended to the line number separated from it by a "/" in error messages.

Obviously, these operations are highly machine dependent and will probably not carry over into other implementations. Since SOS on the DEC-10 and EDIT on the -20 can strip line numbers this function is not a necessity. DIDOT, therefore, does not do it.

17.2. Additional Support Programs

This section will describe the following programs:

PUNCH

TTSSPY

DIABLO

MULTI/DOUBLO

17.2.1. The Program PUNCH

Were it not for the fact that the paper tape service provided in the TOPS-10 monitor appends 2 inches of blank tape after every ASCII form feed (TTS "i"), one could output directly to the paper tape punch or even use PIP to punch files. As it is, the paper tape punch has to be operated in image mode and it is necessary to write a short program to perform this task. PUNCH is that program and its command string is:

```
*PTP:=file name to punch
```

The program counts the characters punched and prints the total length punched at the end. If the phototypesetter were on line, one could probably use PUNCH to copy the file to it though one might also do it through a spooling system. PUNCH is written in MACRO.

17.2.2. The Program TTSSPY

Hopefully, the photosetter output produced by CICERO is now coded correctly. However, when there is suspicion that the coding might be wrong it is helpful to be able to list the output tape or its disk image. TTSSPY does this. The command string is,

```
*Listing file spec=TTS file spec
```

The listing is printed in three columns with the sequence being down each column in turn. As there is an entry for each character on the tape, this listing is quite voluminous.

17.2.3. The Program DIABLO

The Diablo 1610/1620 daisy wheel printer can print at a maximum speed of about 45 cps. It also has a 158 character buffer within its microprocessor memory. Consequently, when transmitting to it at 300 baud there is little risk of buffer over-run and no need for fill characters.

Output to the Diablo that is justified contains a large number of escape sequences to change character spacing. As these codes are non-printing but merely instructions for the microprocessor they can be processed much more rapidly. Therefore, it is advantageous to increase the transmission speed above that at which the machine can print. Although, the standard speed settings are 110,150 and 300 baud it can be strapped to run at 1200 baud. At this speed, it will print at the maximum printing speed of about 45 cps and the escape codes that cause changes in spacing for justification go through without diminishing that speed. To avoid buffer over-runs it is necessary to use a special protocol to stop transmission when the Diablo buffer fills. This protocol is implemented by the program DIABLO.

The command string for this program is

*Diablo line name:=input file specification

If the output specification is omitted a default line will be used. Besides implementing the appropriate protocols, Diablo also sets the line characteristics to those required by STENO or CICERO. We have experienced some problems with DIABLO which we suspect were due to rub out characters appended to the device control codes in escape sequences. We now transmit to the Diablo in image mode and it seems to have fixed the problem. DIABLO is written in MACRO. The program has been tested driving a Qume Sprint Micro 5 and the protocol works quite satisfactorily.

17.2.4. The Programs MULTI and DOUBLO

MULTI is a PASCAL program to perform two passes through a file distinguishing the text for each pass on the basis of the ribbon colour specified through the Diablo control codes. Using this convention it is possible to print the file normally with a two colour ribbon for checking purposes or in two passes changing typewheel between passes with MULTI. Unfortunately MULTI only works at 300 baud. We have also experienced some alignment problems with long reports. DOUBLO will be a program to do the same thing but implementing the necessary synchronisation protocol to allow transmission at 1200 baud. It will be a MACRO program.

17.3. Program Parameters

CICERO and where applicable STENO have been implemented with these limitations:

UNDMAX	16	- Maximum number of underline types.
UPMAX	16	- Maximum number of superscript types.
DOWNMAX	16	- Maximum number of subscript types.
CBMAX	24	- Maximum number of typefaces.
PNTMAX	24	- Maximum number of pointsizes.
MAXCOL	10	- Maximum number of columns.
MAXISB	6	- Maximum number of index sublevels.
MSPMAX	16	- Maximum macro nesting.
MAXNAME	15	- Maximum length for a font name descriptor.
NULDMAX	16	- Maximum number of special character macros.
SPMMAX	16	- Maximum number of special character macros.

SPMAX 16 - Maximum depth of stacking for font names or point sizes.

MAXFNCAR 6 - Maximum number of different printers marks for footnotes.

For some of these the parameter can be changed simply by redefining the symbol and recompiling. However, there are other implications for some. For instance, increasing any of the first six would require redefinition of most of the special internal characters SUBMSP to MAXSPNT.

17.4. DECsystem-10 PASCAL

Both CICERO and STENO are written in DECsystem-10 PASCAL. This is a fairly standard implementation of PASCAL with a LOOP statement and DECsystem-10 orientated file handling. Temporary storage is allocated on a heap with a heap pointer. For this particular application it is convenient to allocate a fixed area from the heap which can be overwritten after each page has been set. We do this by manipulating the heap pointer through MACRO procedures, MARK and RELEASE. These allow us to allocate storage at two different points in the heap. One of these is a one time storage and is used for index and contents entries and for figures and footnotes that do not fit on the page on which they are declared. The other area is called HASH and is used for data that can be overwritten after the page is set. The HASH area is set at the start of the program at the time the first line is ready to be stored in it. The size of the hash area is calculated by a fairly generous algorithm. However, the storage allocated can be inadequate under certain circumstances, eg if one subsequently reduces the point size. The /HASH switch can be used to increase the hash area allocated at the start of the program. For instance,

/HASH:8000

would set a hash size of 8000 words. Both programs print the hash size they actually used on completion.

17.5. Special Features for Compatability with TYP2 Input

TYP2 was an earlier program for creating formatted typescript. Essentially, the switch structure and local control facilities are a subset of those provided by STENO. There is no macro facility and some of the switch names are different. There is no support for the Diablo printer nor for the phototypesetter. One of the incompatabilities with TYP2 is that the contents, index and reference tables are now forced out with predefined macros instead of switches as happened for TYP2.

The control of indentation in free text by counting leading spaces on an input line is a convention derived from TYP2. With hindsight, the system using predefined macros <LM> and <ZM> is much preferable. In early versions of TYP2 each input line of indented text had to start either with leading spaces or a comma. The indentation was cancelled by omitting the spaces or the comma. It soon became apparent that automatic indentation could be provided and cancelled by exiting from shiftable text. This became the default action even for TYP2. The earlier behaviour can still be selected by specifying the /NOAUTOIND switch. (/AUTOIND cancels this switch).

TYP2 allowed page headings but not footings. With the provision of footings and allowing the page number to be included in the heading or footing, the page number line is not really necessary. It is included for compatability and because it is a moderately convenient feature.

17.6. Running CICERO Programs on a DECsystem-20

Programs compiled on a DECsystem-10 can be run on a DECsystem-20 with some slight loss of efficiency. The operating system on the -20 loads a compatibility package into the job's virtual address space when it fields the first UUO. Using this compatibility package, DEC-10 programs saved as .EXE files will run directly on the -20 without the need either to compile or re-load the program. There are some problems if one tries to load a PASCAL program saved as a .REL file on the -10. When the job is started the error message:

```
HEAP OVERRUNS STACK
```

results. The reason for this is that PASCAL tries to locate the HEAP and STACK between the top of the programs data area and the top of the low segment. It gets the latter information from location 44 which is where it would be on a -10. There are no physical segment sizes on the -20 and one could happily locate the heap and stack in any pages that are not write protected. Location 44 is probably in the data area. For all the programs mentioned in this chapter it contains zero but should be patched to a large number eg 377777 so that PASCAL will start its heap there, rather than in the accumulators. Location 44 is not protected as it is on the -10 so one can do this. These complications are necessary only if one has to load DEC-10 PASCAL .REL files on the -20. One of the reasons one might wish to do this is that the current PASCAL compiler on the -10 is a more recent version than that for the -20. It can be run on the -20 with the compatibility package but obviously will produce DEC-10 relocatable files.

APPENDIX A

Default Files

This appendix contains suggested or recommended default file switch settings.

BASIC.TTY - to set switches for the basic subset of chapter 2

```
./STD:# /LIT:& /UFCR:_ /SUB:*
./FSPERS:10 /SPBMIN:7 /SPBMEAN:10 /TABSIZ:8
./LFLEAD:8 /SPACING:8 /PARASP:4 /BLANKSP:6
./LINE:56 /PNO /RPAGE:58 /PAGE:58 /LISTLEAD:1
./FNB
#
./KILL:#
```

CICERO.TTY to set 10 pitch on the Diablo

```
./STD:# /CTD:@ /RTD:$ /SUB:* /IND:| /TCD:` /LIT:& /MACROD:<>
./PRD:!,8 /FND:" /NMD:' /UFCR:_
./xsp:6
./FSPERS:12 /SPBMIN:8 /SPBMEAN:12 /TABSIZ:8
./LFLEAD:8 /SPACING:8 /PARASP:4 /BLANKSP:6
./SCRIPTOP:/:\:4
./LINE:62 /PNO,8 /RPAGE:58 /PAGE:58 /LISTLEAD:1
./FNB
```

ELITEM.TTY to set 12 pitch on the Diablo

```
./STD:# /CTD:@ /RTD:$ /SUB:* /IND:| /TCD:` /SCRIPTOP:/:\:4 /LIT:&
./MACROD:<>
./PRD:! /FND:" /NMD:' /UFCR:_ /UFCR:~:_ /FIGDEL:% /ISC:+
./FSPERS:10 /SPBMIN:7 /SPBMEAN:10 /TABSIZ:8
./LFLEAD:7 /SPACING:7 /PARASP:3 /BLANKSP:3
./LINE:72 /PNO,7 /RPAGE:71 /PAGE:71 /LISTLEAD:1 /MARGIN:8
./FNB /HYPHEN
```

PICA.TTY to set output for an ASCII terminal or line printer

```
./STD:# /CTD:@ /RTD:$ /SUB:* /IND:| /TCD:` /LIT:& /MACROD:<>
./SCRIPTOP:/:\:1
./PRD:! /FND:" /NMD:' /UFCR:_
./TABSIZ:8
./LINE:72 /parind:10 /PNO /PAGE:58 /LISTLEAD:1
./FNB
```

DELITE.TTY to set 12 pitch for the Diablo using program DIDOT

```
./STD:# /CTD:@ /RTD:$ /SUB:* /IND:| /TCD:` /SCRIPTOP:/:\:4 /LIT:& /MACROD:<>
./FND:" /NMD:' /UFCR:~:_ /FIGDEL:% /ISC:+
./FSPERS:10 /SPBMIN:7 /SPBMEAN:10 /TABSIZ:8
./LFLEAD:8 /SPACING:8 /PARASP:3 /BLANKSP:6
```

```
./LINE:72 /PNO,7 /RPAGE:62 /PAGE:62  /MARGIN:8 /TOC IND:12
./FNB /HYPHEN
./SPMAC:_:<BIG><TNRM>
./FNOPTION:MARK:
```

ELITEM.SET to produce photosetter output

```
./STD:# /CTD:@ /RTD:$ /SUB:* /SCRIPTOP:/:\:5 /LIT:& /MACROD:<>
./FND:" /NMD:' /GOLF:[] /SIZE:() /PRD:!,11  /SPMACRO:_:<BIG><NORM>
./SUPERSHIFT:^
./FSPERS:10 /LFLEAD:7
./LINE:34,87  /PAGE:60 /SPBMIN:4 /SPBMEAN:8 /GALLEY:4
./FNB /EMPARA /PARASP:4 /BLANKSP:6
./SPACING:12 /LISTL:1 /HYPHEN /PNODEL:--
./ASSIGNF:autolig[HR] A 3
9,9
7,7                        (<=== A width table)
7,7
7,7
7,7
7,7
6,6
10,5
5,5
10,12
18,16
16,16
16,16
16,16
5,6
4,4
9,9
9,10
10,10
9,13
10,9
10,10
4,10
5,10
5,4
10,4
10,6
15,9
9,10
11,12
5,12
13,14
13,12
5,5
11,10
13,9
13,13
15,13
12,14
12,17
12,12
10,10
10,10
```

```
10,10
10,10
10,10
11,12
12,14
5,10
18,9
./PNO,12(11)[HR]
[HR](11)
./ASSIGNF:AUTOLIC[HI] A 4
9,9
7,7
7,7
7,7
7,7
7,7
6,6
10,5
5,5
10,12
9,10
10,16
16,16
10,7
5,6
4,4
9,9
9,11
11,11
9,14
10,9
10,10
4,10
5,10
5,4
11,4
10,6
15,10
9,11
10,11
5,12
13,14
13,12
5,5
11,10
13,9
13,13
16,13
12,14
11,16
11,13
10,10
10,10
10,10
10,10
10,10
11,11
12,14
```

```
5,10
18,9
./ASSIGNF:AUTOLIG[HB] A 5
11,11
7,7
7,7
7,7
7,7
7,7
6,6
10,5
6,6
11,13
18,16
16,16
16,16
16,16
6,6
5,5
9,10
10,11
11,11
10,14
11,10
11,11
5,10
6,10
6,5
11,5
11,7
16,10
10,11
11,12
5,13
13,14
13,12
5,5
11,11
13,10
13,13
15,13
12,14
12,17
12,13
10,10
10,10
10,10
10,10
10,10
10,10
11,12
13,14
5,10
18,9
./ASSIGNF:NOAUTOLIG[CTR] C 1
11,0
11,11
0,11
11,11
```

```
0,0
11,11
11,0
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
11,11
18,9
./ASSIGNF:NOAUTOLIC[T G] C 2
10,10
15,18
10,10
15,15
15,15
15,15
15,15
10,15
15,18
18,15
15,10
```

```
4,15
15,4
15,6
10,4
15,15
9,9
10,15
11,9
10,15
10,10
9,10
6,11
8,8
12,9
10,11
13,10
10,9
13,10
12,13
7,15
14,15
15,12
10,10
15,14
14,15
16,11
17,14
15,11
14,15
15,13
15,15
15,15
15,15
15,15
15,15
13,12
14,15
4,5
18,9
./ASSIGNF:NOAUTOLIG[TCI] C 3
18,15
10,6
15,15
15,18
15,15
10,15
15,15
15,15
15,15
15,15
15,15
15,18
8,15
10,15
12,15
10,18
9,9
10,18
```

```
11,9
10,10
10,10
9,10
6,11
8,8
12,9
10,11
13,10
10,9
13,10
12,13
7,14
14,15
15,12
15,15
15,14
13,15
15,12
17,14
15,11
13,15
15,13
15,18
18,15
15,15
15,6
18,15
13,12
14,18
5,10
18,9
./UFLF:~:_,4[HI]
```

APPENDIX B

The /LIST Switch

The purpose of the /LIST switch is to list the current settings of switches and the names of macros. For instance, it can be used to list default settings resulting from particular default files. The switch takes arguments to select the display required. If no argument is provided it does everything except list the help prompt. The permitted arguments are:

CHARACTERS	- List all special characters.
FLAGS	- List all boolean settings.
FONTS	- List all typeface names.
MACROS	- List all macro names.
NAMES	- List all switch names.
POINTSIZES	- List all point sizes available on the phototypesetter.
TITLES	- List page titles currently defined.
VALUES	- List all numeric values.
HELP	- List the permitted arguments.

As with the switch names, only the characters necessary to identify the argument uniquely need be typed. The HELP argument causes these to be typed in upper case. Similarly, for switch names the NAMES argument causes the necessary characters to be typed in upper case.

APPENDIX C

Facilities Deleted from STENO in Order to Make DIDOT

DIDOT is a cutdown version of STENO. It is experimental and at present undocumented. It was made from STENO by cutting out these features:

1. The /SHIFT switch and support for case conversion. This improves efficiency but requires a full case keyboard.

2. The /PRD and /PRIND switches and everything to do with the page reference line.

3. <FON> and <FOFF> predefined macros. All they did was to cause the Diablo to replace printing with spacing and in fact that probably does not work anyway for STENO.

4. Aligning tabs and their associated predefined macros <TFIX>, <ZTFIX>, <ALIGN> and <XPND>.

5. /AUTIND and /NOAUTIND switches since <ZM> does the same thing better. The leading comma convention also goes so a comma is now a normal character in all circumstances.

6. Left margin indentation by counting leading spaces.

7. /FORGET and /NULDELIM switches since CICERO compatible files that can be set as typescript or photoset can be written without them using the special macro facility.

8. Less used underlining switches /UCR, /ULF and /UBS that do not underline spaces and /UNS and /UFNS that do flexowriter style non-spacing underlining.

9. Maintenance switches /NOCHKCOL and /CHKCOL.

10. Redundant switches provided for TYP2 compatibility.

    /CMARGI - use /GUTTER instead.

    /FILL - use /JUSTIFY instead.

    /GOLF see 13 below.

    /LISTLEAD - not needed anyway.

11. /QUEUE switch is not needed for the Diablo and is really a more appropriate function for the Diablo dumping program.

12. KEY and DOWN options of the /FNOPTION switch.

13. All support for multiple typefaces, since that does not work well for the Diablo and presumably when the dual wheel Qume comes we can code the change of typeface the way we do ribbon colour at the moment.

14. /LIST switch except for /LIST:HELP, /LIST:MACROS and /LIST:NAMES since these are useful to tell what is left in DIDOT.

15. The reference pre-scan to create a sorted reference table has been cut. This means numeric references will be numbered in the order of their occurrence.

16. Sidenotes are not implemented in DIDOT.

17. DIDOT's error reporting is not as good as STENO's.

18. The /FIGFIT and /FIGCARRY switches are not implemented in DIDOT and, at present, the default action when positioning figures corresponds to the "FLOAT" argument for these switches.

19. There is no line number stripping.

20. The <NCHAP(NAME)> predefined macro is not implemented.

DIDOT is about 11P shorter than STENO and runs about 20% faster.

INDEX

Aligning tabs 69
Alphabetic indexes 58
Arabic numbering 48
ASCII 9
Autoligature substitution 84
Backspace 34
Basic subset of CICERO 14
Blank lines
    At top and bottom of page 80
    Before headings 39, 79
    Between paragraphs 37, 79
    Between tabulated records 80
Bold typing 71, 81
Calling macros with special characters 75
Carriage return 34
Case conversion 31
Cassettes 10
Central processing unit 9
Centre text delimiter 28
Change of typeface 32
Changing
    Contents heading 62, 73
    Contents leader character 61
    Index heading 60, 73
    Number of columns 43
    Ribbon colour 71
    Text processing mode 25, 28
Chapter descriptor 48, 74
Chapter numbers 48
Character mapping when photosetting 100
Character size 85
Cicero 92
Columns 43
Comma 35
Command strings 22
Composition 11
Computing devices 9
Contents 60
Contents
    Appending text to the end of the contents 62
Contiguity 46
Contiguity
    For headings 72
Continuation character 32
Control action 20
Control concepts 19
Control structure 22
Controls
    Global 19
    Local 19
Counters
    Alphabetic 67
    Autoincrementing 67
Counters
    Numeric 67
CRLF 16
Daisy wheel printers 11, 77
Decimal tabs 69
DECsystem-10 18
DECsystem-20 106
Delimiters 21
Delimiters
    Centre Text 28
    Null mode 28
    Right text 28
    Shiftable text 28
Diablo 1610/1620 printer 18, 77
DIDOT 102
Didot point 92
Disks 9
Dot 35
Double column format 43
Double space 22
Drums 9
EBCDIC 9
Editor programs 13
EM space 86, 93
EN leader 61, 68
EN space 93
End notes 63
Errors
    Figure positioning 52
    Figure size 53
    Inadvertent indentation 17
    Overflow of hash area 106
    Unexpected blank lines 58
Escape sequences 77
Extra blank lines 49
Figure captions 53
Figure text 53
Figures 51
File sequence 19
File structure 19
Film advance 91
Floppy disks 10
Font change messages 32
Font names 32
Footnotes 54
Footnotes in multiple columns 56
Forcing a new line 15
Form feed 34
Fractions 98
Full stop 22, 35
Global controls 19, 21, 22
Global specification 22
Golf ball 11
Golf ball identifiers 32
Gutters 43
Hanging indent 16

Hard copy 8
Headings 39, 72
Hierarchy of contiguity
    requirements 46
Horizontal tab 34
Hot metal typesetting 11, 91
Hyphenation 40, 72
Immediate action markers 22
Indentation 17
Indentation
    Continuation lines in indexes
        59
    Continuation lines in the con-
        tents 61
    Left margin 68
    Page reference line 63
    Sublevels in indexes 59
Index level separator character 58,
    61
Index pages 59
Index
    Appending text to the index 60
    Changing the heading 60
Indexes 58
Ink jet printers 11
Intercolumn gutters 43
Intercolumn margin 43
Interfacing 13
Justification 39, 80
Leader characters 61
Leading 91
Left margin 16, 42, 68
Ligatures 84
Line feed 34
Line length 36
Line numbers 104
Line spacing 37
Linotype 91
Lists of short items 16
Literal marker 17, 33
Local action markers 22
Local controls 19, 21
Location of controls 20
Location of
    Contents 62, 72
    Index 60, 72
    Reference tables 64, 72
Logical page length 42
Logical page number 48
Lower case 91
Macro delimiters 65
Macros 65
Macros to
    Append text to index or con-
        tents 73
    Cause bold typing 71
    Change chapter descriptor 48,
        74
Macros to
    Conserve special characters 73
    Control ribbon colour 71
    Control sidenote column reser-
        vation 74
    Control text processing sub-
        modes 74
    Control the left margin 68
    Facilitate macro coding 72
    Locate indexes etc. 72
    Set special characters 68
    Tabulate 68
Magnetic tape 9
Margin 16
Margins 42
Mathematics 97
Mergenthaler V-I-P phototypesetter
    100
Monotype 91
Multiple columns 38, 43
Murray code 91
Name delimiters 65
Named tabstops 98
New paragraphs 16, 37, 86
New
    Column 49
    Page 49
Null mode and indexes 58
Null mode delimiter 28
Operating system 13
Operators
    Down 97
    Subscript 97
    Superscript 97
    Up 97
Page
    Headings and footings 44
    Length 42
    Number 44, 48
    Number delimiters 48
    Number line 80
    Reference line 49, 62, 80
Pagination 42
PASCAL 8, 106
Period 35
Photocomposition 12
Phototypesetters 12
Phototypesetting 84
Physical location of typefaces 84,
    101
Physical page length 42
Pica 92
Point 92
Point size 85
Positioning figures 51
Positioning the page number 48

Predefined macros
<3USP> 68
<ADDPLD> 72
<ALIGN> 69
<BLACK> 71, 81
<CONTENTS> 62, 72
<CTD> 73
<EMSP> 68
<ENSP> 68
<FF> 34
<FIGD> 73
<FND> 73
<FOFF> 71
<FON> 71
<HYPHEN> 40, 72
<IND> 73
<INDEX> 60, 72
<INDFOOT> 60, 73
<INDHEAD> 60, 73
<ISC> 73
<L3USP> 68
<LEMSP> 68
<LENSP> 68
<LIT> 73
<LM> 68
<LTSP> 68
<LVTBS> 68
<LVTSP> 68
<NCHAP(NAME)> 74
<NCHAP> 43
<NLDR> 68
<NMD> 73
<NOHYPHEN> 40, 72
<NOJUST> 72
<PRD> 73
<QIN> 74
<QOUT> 74
<RED> 81
<REF> 71
<REFA> 64, 72
<REFERENCE> 64, 72
<REFN> 64, 72
<RFAD> 73
<RFND> 73
<RSF> 73
<RSS> 73
<RTD> 73
<SND> 73
<SNOFF> 57, 74
<SNON> 57, 74
<STAB> 68
<STD> 73
<SUB> 73
<SVF> 73
<SVS> 73
<TAB> 68
<TBLD> 71, 81
Predefined macros
<TCD> 73
<TFIX> 69
<TNEXT> 44, 70
<TNRM> 71, 81
<TOCFOOT> 62, 73
<TOCHEAD> 62, 73
<TOFF> 44, 70
<TON> 44, 70
<TSP> 68
<TSTART> 44, 70
<TXBLD> 71, 81
<UPGNL1> 47, 72
<UPGNL> 47, 72
<VTBS> 68
<VTSP> 68
<XPND> 69
<ZM> 68
<ZTAB> 68
Printers 10
Printing 11
Printing measures 92
Programs
CICERO 102
DIABLO 104
DIDOT 102, 116
DOUBLO 105
MULTI 105
PUNCH 104
STENO 102
TTSSPY 104
TYP2 106
Qume Sprint 5 printer 77
References
Alphabetic 64
Numeric 63
Right text delimiter 28
Roman numerals 48
Saving and restoring
Point size 73
Typeface 73
Selectric typewriters 11
Separators 22
Shiftable text 15
Shiftable text delimiter 28
Sidenotes 54, 56, 74
Slave setters 12
Spaceband parameters 39
Spaces
3 unit space 68
EM space 68
EN space 68
In the input 35
Thin space 68
Unit space 68
Special macro character 81
Special symbols 99

Specification of control action 20
STENO 102
String 13
Stripping line numbers 104
Structure
    Global controls 22
    Local controls 23
    Switches 23
Sub-modes
    Free text 74
Sublists within indexes 58
Subscripting 97
Substitute characters 68
Substitute space 17, 31
Superscripting 97
Supershift
    Characters 85
    Operator 86
Switch names 23
Switch structure 23
Switches 14, 19
Switches
    /ARABIC 48
    /ASSIGNFONT 23, 32, 84, 101
    /AUTOIND 106
    /BLANKSP 39, 79, 89
    /CLASHES 37, 98
    /COLOURFONT 87
    /COLUMN 43
    /CONCAR 32
    /CONTENTS 62
    /CTD 28
    /CTONL 46
    /EMLIMITS 86
    /EMPARAGRAPH 86
    /FF 34
    /FIG 51, 79, 88
    /FIGCARRY 51
    /FIGDELIMIT 53
    /FIGFIT 51
    /FIGURE 51, 88
    /FNB 56
    /FND 54, 89
    /FNK 56
    /FNOPTION 55
    /FNT 56
    /FNZK 56
    /FONT 32
    /FOOTING 23, 45
    /FORGET 33
    /FSPERSPACE 77
    /FTONL 46
    /GALLEY 90
    /GBMSG 32, 87
    /GNL 47
    /GO 103
    /GOLF 32
Switches
    /GUTTER 43, 78, 87
    /HASH 106
    /HEADING 23, 45
    /HYPHEN 40
    /IND 58
    /INDEX 59
    /INDIND 59, 78
    /INDINDENT 87
    /INDSUB 59, 78, 87
    /ISC 58
    /JUSTIFY 39
    /KILL 33
    /LFLEAD 37, 42, 78
    /LINE 36, 78, 87
    /LIST 103, 115
    /LISTLEADIN 87, 89
    /LITERAL 33
    /MACRODELIM 65
    /MARGINSIZE 42, 78, 87
    /MAXADDLEAD 39, 82, 89
    /NMD 28, 58
    /NOAUTOIND 106
    /NOCLASHES 37, 98
    /NOFF 34
    /NOHYPHEN 40
    /NOJUSTIFY 39
    /NOPNO 48
    /NULDELIM 34
    /NUMBER 48
    /PAGE 42, 79, 88
    /PARASP 37, 79, 89
    /PARINDENT 37, 86
    /PNO 48, 80, 89
    /PNOBOTTOM 48
    /PNODELIMIT 48
    /PNOTOP 48
    /PRD 62, 80, 89
    /PRINDENT 63, 78, 87
    /REFA 64
    /REFN 63
    /ROMAN 48
    /RPAGE 42, 79, 88
    /RTD 28
    /SCRIPTOP 97
    /SDEFAULT 14, 26
    /SHIFT 32
    /SIDENOTES 57
    /SIZEDELIMIT 85
    /SND 57
    /SNSPACE 57
    /SPACING 37, 79, 89
    /SPBMEAN 40, 80, 89
    /SPBMIN 39, 80, 89
    /SPMACRO 75, 81
    /STD 28
    /SUB 31

Switches
/SUPERSHIFT 86
/TABONL 46
/TABSIZ 78
/TABSIZE 87
/TABULATE 44, 70, 87
/TCD 61
/TOCIND 78
/TOCINDENT 61, 87
/TOCLEADER 61
/TOCNOS 61, 87
/TOCNUM 78
/TSPACINC 44, 80, 89
/UBS 29
/UCR 29
/UFBS 29
/UFCR 29
/UFLF 29, 79
/UFNSP 29
/ULF 29, 79
/UNSP 29
/XFOOTINC 45
/XSP 49, 80, 89
Symbolic name 65
Symbols 65
System software 13
Table of contents 60
Tables 53
Tabstops
Aligning 69
Named 66
Simple incremental 68
Tabulated columns 43, 70
Tabulation
Simple incremental 68
Tapes 9
TECO 18
Terminal 9
Terminal command strings 20
Text Processing 7
Text processing modes 25
Text processing modes
Centre text 25
Null mode 25
Right text 25
Shiftable text 15, 24, 36
Thin space 93
Time-sharing 13
TTS 9
TTS code 91
TTS encoded paper tape 12
TYP2 8, 106
Typeface names 32
Typesetting 7
Typography 95
Underlining 17, 29
Unit backspace 93
Unit space 93
Upper case 91
Utility programs 13
V-I-P 100
VDU 8
Vertical tab 34
Widow lines 47
Width table 84, 101
Word Processing 7

DISTRIBUTION

Copy No.

AUSTRALIA

DEPARTMENT OF DEFENCE

Central Office

Chief Defence Scientist	1
Deputy Chief Defence Scientist	2
Superintendent, Science and Technology Programs	3
Australian Defence Science and Technical Representative (UK)	4
Counsellor, Defence Science (USA)	5
Defence Library	6
Joint Intelligence Organisation	7
Assistant Secretary, DISB	8-24
First Assistant Secretary, Computing Services	25
Director, Instructions, Orders and Manuals	26

Aeronautical Research Laboratories

Chief Superintendent	27
Principal Officer, Computer Centre	28
Superintendent, Structures Division	29
Divisional File, Structures Division	30
Authors: R. C. Adams	31
G. A. Cleave	32
Library	33
Superintendent, Aerodynamics Division	34
Superintendent, Systems Division	35
Superintendent, Materials Division	36
Superintendent, Mechanical Engineering Division	37
Principal Engineer, Engineering Facilities Division	38
Administrative Officer	39

Materials Research Laboratories

Library	40
Mr W.G. Thege	41-60

Defence Research Centre, Salisbury

Library	61
Principal Officer, Computing Services, ERL	62

Central Studies Establishment

Information Centre	63

Engineering Development Establishment

Library	64
Principal Technical Officer, Technical Information Group	65
Mr T. Duell	66

RAN Research Laboratory

Library	67

Navy Office

Naval Scientific Adviser	68

Copy No.

Army Office
- Army Scientific Adviser 69
- Office of the Military Secretary, Mr C.E. Roach 70

Air Force Office
- Air Force Scientific Adviser 71
- Engineering (CAFTS) Library 72
- HQ Support Command (SENCsO) 73

Defence Printing Establishment
- Manager 74

DEPARTMENT OF PRODUCTIVITY

Government Aircraft Factory
- Manager/Library 75
- D. Bryant 76

C.S.I.R.O.
- Central Information, Library and Editorial Section 77
- Division of Computing Research, Chief 78

AUSTRALIAN BROADCASTING COMMISSION
- Director, A.D.P. 79

UNIVERSITIES AND COLLEGES

Adelaide
- Barr Smith Library 80
- Professor of Computing Science 81

Australian National
- Library 82
- Coombs Computing Services, Director 83

Deakin
- Professor of Computing Science 84
- Manager, Computer Centre 85

Flinders
- Library 86
- Director, Computer Centre 87

James Cook
- Library 88
- Director, Computer Centre 89

La Trobe
- Library 90
- Director, Computer Centre 91

Melbourne
- Library 92
- Professor of Computing Science 93
- Director, Computer Centre 94

	Copy No.
Monash	
Library	95
Professor of Computing Science	96
Director, Computer Centre	97
New England	
Library	98
Director, Computer Centre	99
Sydney	
Engineering library	100
Library, Basser Dept of Computer Science	101
Professor J.M. Bennett	102
Director, Computer Centre	103
New South Wales	
Physical Sciences Library	104
Professor M.W. Allen	105
Director, Computer Centre	106
Queensland	
Library	107
Professor of Computing Science	108
Director, Prentice Computer Centre	109
Tasmania	
Library, Department of Information Science	110
Professor A. Sale	111
Director, Computer Centre	112
Western Australia	
Library	113
Director, Western Australia Regional Computing Centre	114
W.A.I.T	
Director, Computer Centre	115
Library	116
R.M.I.T.	
Library	117
Library, Computing Science Department	118
Director, Computer Centre	119
CANADA	
University of Western Ontario	
Library	120
Director, Computer Centre	121
Professor of Computer Science	122
NEW ZEALAND	
Department of Health	
Computer Centre, Wellington	123

Copy No.

UNITED KINGDOM

Royal Aircraft Establishment, Farnborough
Library 124

National Physical Laboratories
Library 125

British Library
Science Reference Library 126
Lending Division 127

Rolls-Royce (1971) Ltd.
Bristol Siddeley Division T.R. & I. Library Services 128

University of Oxford
Director, Computing Laboratory 129

Hatfield Polytechnic
Director, Computer Centre 130

UNITED STATES OF AMERICA

University of Pittsburgh
Director, Computer Center 131

N.A.S.A.
Scientific and Technical Information Facility 132

SPARES

Copies 133 to 300

DOCUMENT CONTROL DATA SHEET

Security classification of this page: Unclassified

1. Document Numbers
   (a) AR Number:
   AR-001-743
   (b) Document Series and Number:
   Structures Note 453
   (c) Report Number:
   ARL-Struc-Note-453

2. Security Classification
   (a) Complete document:
   Unclassified
   (b) Title in isolation:
   Unclassified
   (c) Summary in isolation:
   Unclassified

3. Title: CICERO Typesetting Manual

4. Personal Author(s):
   Adams, R.C.
   Cleave, C.A.

5. Document Date:
   July, 1979

6. Type of Report and Period Covered:

7. Corporate Author(s):
   Aeronautical Research Laboratories

8. Reference Numbers
   (a)Task:

9. Cost Code:
   616450

   (b) Sponsoring Agency:

10. Imprint
    Aeronautical Research Laboratories, Melbourne

11. Computer Program(s)
    (Title(s) and language(s)):
    CICERO Pascal
    STENO
    DIDOT

12. Release Limitations:
    Approved for public release

12-O. Overseas:	N.O.		P.R.	1	A		B		C		D		E	

13. Announcement Limitations: No limitation

14. Descriptors:
    Text processing
    Word processing
    Composing
    Printing
    Computer typesetting

15. Cosati Codes:
    0502
    1405

16.

ABSTRACT

CICERO is a scheme for coding typesetting information into natural language text. Very powerful text processors may be designed to produce fully paginated typescript or photoset copy from text files containing a free format source text and CICERO format coding. Such files may be edited very easily and subsequently reformatted automatically. The separation of the editing function

from formatting is one of the principal advantages of CICERO. CICERO provides means to control justification, hyphenation, pagination and tabulation, the location of figures and indexes, the setting of multi-line mathematical expressions, footnotes, sidenotes and the contents of indexes.

The manual is mainly a users guide to the coding scheme. Several computer programs are available to process files with CICERO coding. All these programs are written in Pascal and run on a DECsystem-10 or -20 computer. One of these implements all the features described in this manual and is capable of producing TTS encoded output to drive a V-I-P phototypesetter.

This manual has been set as typescript using CICERO.