An introduction to text processing using SGCAL

by

Philip Hazel

Second edition
June 1994

Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG
United Kingdom

Telephone: +44 223 334600

1. Introduction

SGCAL is a text formatting program which is a direct descendent of the GCAL program. Although it works in a similar way from the user's point of view, its internals are totally different, and there are also a number of incompatibilities at the basic user level, though at the level of the standard styles these do not show very much, other than in the revised text strings used for font flags.

SGCAL can do most of the things that GCAL can do, and several of the things that GCAL cannot do. There are some GCAL features that are not implemented in SGCAL, but in many cases this is simply because they have not yet been needed, rather than any fundamental problem.

SGCAL and the associated programs SGTYPE (for previewing output) and SGTOPS (for converting the output into PostScript) are written in ANSI Standard C. At present all three programs run under Acorn's RISC OS operating system and under the SunOs version of Unix (where the X window system is used by SGTYPE), while SGCAL and SGTOPS also run on Phoenix/MVS.

This document contains introductory material describing the use of SGCAL. It is aimed at users who have not used a text processor before. The descriptions it contains are in some cases incomplete and simplified. If a particular facility cannot be found herein it does not necessarily mean that SGCAL does not have it.

The full specification should be consulted for the definitive definition of all SGCAL facilities. It also contains a description of the differences between GCAL and SGCAL, and some suggestions for converting GCAL input files for use with SGCAL.

Another document entitled *Specification of Aspic* describes an additional program for generating line drawings that can be used in conjunction with SGCAL.

2. Basic text formatting with SGCAL

2.1 What is text processing?

The terms *word processing* and *text processing* are often confused. The former normally refers to a system or program that permits the user to lay out text on a screen more or less in the form in which it is subsequently printed. The latter is a more involved process in which the text of a document (often called the *copy*) is entered into a computer system together with additional information as to how it should be laid out and printed. A word processor or text editor can be used to do this. The extra information mixed in with the copy is called the *markup*, and it has the same function as the marks added to a manuscript by a copy editor before sending it to be typeset. The mixture of copy and markup is read by a text processing program, which formats the copy as requested and generates instructions for the device on which it is to be printed.

While the added complication imposed by a text processor may not always be appropriate for short documents, for longer ones there are several advantages. The device on which the input is prepared can be very much simpler than the ultimate printing device; a small microcomputer can be used to prepare text for the most sophisticated phototypesetter, for example. Another advantage is that it is easy to format the same input text in different ways or for different output devices, provided care is taken in the marking up. Text processors, particularly on small machines, can also offer more facilities in the way of automatic chapter, section and footnote numbering, floating inserts, creation of indexes and so on.

2.2 Specific and generic markup

Many text processors allow the user to include very specific instructions in the markup, for example *'leave 12 points of white space and indent the next line by 24 points'*, which might be used at the start of a paragraph. (A *point* is a traditional unit of length used in the printing industry; it is approximately 1/72 of an inch.) Including such *specific markup* in an input file is not a good idea because if there is any need to change it for some reason, every occurrence in the file must be sought out and changed.

The alternative is to use *generic markup*, which indicates the logical structure of the document without specifying how this structure is to be represented on the page. For example, the start of each paragraph is indicated by a markup instruction *'start of paragraph'*, and the start of a chapter by *'start next chapter with title such-and-such'*.

Of course, the text processor has ultimately to be told what *'start of paragraph'* actually means in layout terms. This is done by a series of definitions that can either appear at the start of the input, or in a separate file which is referenced from within the main input. Such a set of definitions specifies a *document style*. It is then easy to alter the layout parameters if the need arises, and, what is perhaps more important, the style is guaranteed to be consistent throughout the document.

2.3 Coding the markup in SGCAL

Because the input to a text processor is a conventional file of characters that can be typed on a keyboard, there has to be some way of distinguishing which characters are copy and which are markup. In SGCAL there are two distinct forms of markup encoding:

- *Directives* are major instructions which always occupy an entire input line by themselves. The line begins with a full stop, followed by the name of the directive and possibly other information. Some examples of SGCAL directives are

  ```
  .display
  .section Coding the markup in SGCAL
  .index directives
  ```

  Directive names are normally in lower case (small letters) – in SGCAL, upper case and lower case letters are considered as distinct.

- *Flags* are the other form of markup; they normally appear mixed up with the copy, and normally consist of a character that is neither a digit nor a letter, possibly followed by other characters. Upper and lower case letters are distinct in flags as well as in directives, so, for example, '`$f`' and '`$F`' are two completely different flags.

  Flags are used to encode instructions that apply to the immediately surrounding text (for example, to change font) or to cause the insertion of a character that is not available on the normal keyboard. Some examples of SGCAL flags are

`$it{`	to change to italic text
`#`	to insert an 'exact space'
`$alpha`	to insert a Greek alpha

When input is being prepared for SGCAL it is important that the copy and the markup not be confused. The typist must take special action if any line begins with a full stop (which is not very likely) or if any of the special characters that begin a flag appears in the copy. The set of such characters can be changed by SGCAL directives, but the default set which is used with the standard styles is

`~`	tilde
`_`	underline
`#`	sharp sign, or 'hash'
`$`	dollar
`}`	closing curly bracket

If it is necessary to include one of these characters as part of the copy, it should be preceded by a 'commercial at' characters (@). Thus

```
for ~ type @~
for _ type @_
for # type @#
for $ type @$
for } type @}
```

The flag sequence '`@`' is called the *quote flag*, and it can also be used to insert a full stop at the start of a line, should this ever be necessary.

As well as these flag characters, the '–' character has special significance in that it is treated as a possible hyphenation point. To prevent hyphenation in an individual instance, '`@-`' can be used.

2.4 Additional markup for typesetting devices

If the output is ultimately to appear on a device which uses typographic fonts, such as a laser printer or phototypesetter, there are some additional points to note when preparing the copy.

There is normally a distinction between opening and closing quotation marks on these devices, the normal computer 'quote' character producing a closing single quote. For an opening single quote the character called 'grave accent' in the ASCII character set is commonly used.

Double opening and closing quotes are obtained by typing two successive 'grave accents' or 'quote' characters respectively. Here is an example of some copy marked up for a typographic device:

```
He said `I shall write to ``The Times'' tonight'.
```

If a text which has been marked up in this way is processed for a simpler device with only 'typewriter' quotes, appropriate conversions are made.

Typographic fonts may also distinguish between a hyphen, an en-dash and an em-dash, which are all different lengths of short horizontal line. In SGCAL input, a single '–' character is treated as a hyphen, while en-dashes and em-dashes are entered as two and three successive '–' characters respectively. For example,

```
An en-dash is used for ranges, such as
19--42, and a spaced en-dash is used -- as
here -- to set off parenthetical comments.
The use of an em-dash---without
spaces---for this purpose is going out of
fashion.
```

Again, when a device does not have this variety of characters, appropriate substitutions are made.

2.5 SGCAL's standard macros

The SGCAL program implements a number of basic text processing facilities, including a number of particular directives and flags. It also provides means by which these basic facilities can be combined into higher level operations. A directive which is built from more primitive operations is known as a *macro directive*, and a collection of macro definitions is called a *macro library*. Additional flags can also be defined.

SGCAL is intended to be used in conjunction with a macro library, and the use of 'raw' SGCAL without any macro directives is exceptional. A standard library containing definitions for a number of standard document styles is part of all implementations. The directives and flags that are described in subsequent sections of this document include many that are in fact part of the standard library rather than 'raw' SGCAL. In the list of directives that appears at the end, the primitive ones are marked with an asterisk.

3. SGCAL Input File Structure

An SGCAL input file is divided into two parts, the first of which is normally only a few lines long. This selects the style for the document and possibly makes some changes to the standard options. The remainder of the input file consists of the marked up copy, as described in the remainder of this document.

3.1 Selecting a standard style

The first line of an SGCAL file must consist of a **library** directive specifying the name of the style, enclosed in quotes. For example

```
.library "a4ps"
```

selects the style that is designed for PostScript output on A4-sized paper. A style can alternatively be specified as a parameter on the command that invokes SGCAL. This single line is all that is needed in many cases.

Certain features of some styles can be varied by setting parameters before using the **library** directive. For example, for the `a4ps` and `a5ps` styles, the size of type and the typeface family can be specified in this way. No other directives should normally appear before **library**.

3.2 Special character flags

A number of standard flags are provided to give access to certain special characters that are not in the normal computer character set. These standard flags, and the characters they represent, are as follows:

`-->`	→	right arrow
`<--`	←	left arrow
`<->`	↔	two-headed arrow
`($)`	£	pound sterling
`(c)`	©	copyright sign
`$'`	′	feet (or minutes)
`$.`	•	bullet

An example of the use of these special characters is

```
.library "a4ps"
6$' of pipe cost ($)7.85.
```

SGCAL also contains support for letters from the Greek alphabet. However, these are not available by default, and it is necessary to obey the directive

```
.usegreek
```

to gain access to them. The names of the flags are '`$alpha`', '`$beta`', etc. for lower case letters, and '`$Alpha`', '`$Beta`', etc. for the capitals. For devices which do not have these characters directly available, approximations are generated using overstruck characters.

Additional device-specific characters are made available in a similar way for certain output devices. In particular, for the PostScript styles, the directive **useaccents** defines a set of flags for printing accented characters.

The more advanced user may wish at this point to include directives to vary the standard style, for example to indent all the displays, or change the amount of white space preceding each section. Some of the possibilities are described later in these introductory chapters. If this text is of any length, it may be convenient to keep it in a separate file which is inserted into the main file by means of the **include** directive, as for example

```
.include "header"
```

The **include** directive can also be used to put together a single document from a number of separate files.

3.3 An example of SGCAL input

The next page contains an example of a complete input file for SGCAL which illustrates the general style. The markup items are described individually later in this document. The output produced from this file follows on the subsequent page.


```
.library "a5ps"
.chapter An Example of SGCAL Output
This is an example of the output produced by SGCAL
when it   processes   the  input on the previous
page using the style definition 'a5ps'.

.section A short story
Once upon a time there was a $it{beautiful}
princess who often used to go for long walks by
herself in the woods near her castle.

.subsection The plot thickens
One day, while she was out walking, she was
confronted by a fierce
.display
$c $bf{D R A G O N}
.endd
which was spitting fire and flame.

.subsection The dilemma
Fortunately the princess had grown up in the
electronic age, and knew all
about dragons and other monsters.
.footnote
In her palace she had a huge collection
of home computers.
.endf
Should she
.numberpars
Zap it with her laser cannon?
.nextp
Lure it to the
bottomless pit just around the next corner?
.nextp
Utter the magic spell given to her by the
Great Binary Wizard?
.endp

.subsection Conclusion
The dragon, seeing that it had met its match,
surrendered, and they both lived
happily ever after.
```

An SGCAL input file

4. An Example of SGCAL Output

This is an example of the output produced by SGCAL when it processes the input on the previous page using the style definition ‘a5ps’.

4.1 A short story

Once upon a time there was a *beautiful* princess who often used to go for long walks by herself in the woods near her castle.

4.1.1 The plot thickens

One day, while she was out walking, she was confronted by a fierce

D R A G O N

which was spitting fire and flame.

4.1.2 The dilemma

Fortunately the princess had grown up in the electronic age, and knew all about dragons and other monsters.[1] Should she

(1) Zap it with her laser cannon?

(2) Lure it to the bottomless pit just around the next corner?

(3) Utter the magic spell given to her by the Great Binary Wizard?

4.1.3 Conclusion

The dragon, seeing that it had met its match, surrendered, and they both lived happily ever after.

[1] In her palace she had a huge collection of home computers.

5. SGCAL Markup for Running Text

In a previous chapter the general form of SGCAL markup was described. In this chapter the particular ‘marks’ relevant to sections of running text are defined and explained.

5.1 Paragraphs

Most paragraphs of running text can be typed verbatim. The only times when markup is required are

- When one of the characters that SGCAL uses to introduce a flag is part of the copy;
- When a character not in the normal printing set is encountered;
- When a change of font or underlining state is required;
- When some change from the normal spacing or line splitting rules is wanted.

5.1.1 Characters that introduce flags

These are the characters ‘`$`’, ‘`#`’, ‘`~`’, ‘`_`’, ‘`}`’, and a full stop at the beginning of an input line; any occurrences that are part of the copy must be preceded by ‘`@`’, for example:

```
... the @$20,000 question ...
```

Hyphens in the copy are taken as possible line-splitting places unless preceded by ‘`@`’.

5.1.2 Special characters

The availability of special printing characters is very dependent on the printing device that is ultimately used. The style definition files for particular devices contain appropriate flag definitions. Tables of characters for the various printing devices should be consulted when a particular character is sought. All the standard styles make use of the convention of using ‘grave accent’ for an opening quote and the ordinary ‘quote’ for closing quote (with doubling where required), and ‘`--`’ and ‘`---`’ for en-dash and em-dash respectively.

5.1.3 Font changes and underlining

The following standard flags are defined for changing font:

```
$rm{     to select the roman font
$it{     to select the italic font
$sl{     to select the slanted roman font
$bf{     to select the bold font
$tt{     to select the typewriter font
$ss{     to select the sanserif font
$sc{     to select the 'small caps' roman font
```

Whenever there is a change of font, the previous font is remembered on a 'stack', from where it can be recalled by means of a flag consisting of a single closing curly bracket. An example of a sentence that uses several fonts is

```
These words are $it{italic}, $bf{bold}, and roman.
```

Advanced users who make use of other fonts are recommended to create suitable mnemonic flags.

Sections of text to be underlined are bracketed by the *underline flag*, which consists of a single underline character. For example, the input

```
Here is an _underlined_ word.
```

produces as its output

Here is an <u>underlined</u> word.

SGCAL is very flexible in the way its flags are defined, and it is possible to change the meaning of any flag sequence. For example, if a document has been marked up with underlines, these can be changed to, say, italic *without changing the main input*, by re-defining the meaning of the flag sequence '_'. In general it is best to mark up for the most sophisticated device that is ever likely to be used for the document, as it is easier to re-define the flags for a simpler device than *vice versa*.

5.1.4 Formatting paragraphs

When SGCAL is formatting a paragraph it tries to fit as many words onto each output line as possible. The length of input lines is of no account, nor is the number of spaces between words. For example, the input for the present paragraph could be as follows:

```
When SGCAL is formatting a paragraph
it tries  to  fit  as many words onto
each output line as possible.  The length of input
lines is of no account,
nor    is   the number of spaces between words.
For example, the input
for the present paragraph could be as follows:
```


Each line of a paragraph is stretched out to reach the right hand margin if SGCAL is operating with both left and right justification enabled (the default). The stretching is done by including extra spaces between words. For printing devices that cannot produce spaces that are significantly narrower than the printing characters, the result is not very pleasing. Right-justification can be turned on or off at any time by means of the **justify** directive. An appropriate setting is selected in the style definition file for any given printing device.

It is occasionally necessary to tell SGCAL not to split a line at a particular place. The flag '$>' is used to specify a *non-splitting space*, for example

```
The author's name is A.N.$>Other.
```

Such a space can, however, be stretched if necessary. Another sort of space that behaves just like a printing character and neither stretches nor is a possible splitting point is called an *exact space*. It is indicated by the flag '#'.

A new paragraph is started whenever an empty input line or the **newpar** directive is encountered. SGCAL reads the entire text of a paragraph before splitting it up into lines. The first line of a paragraph is never placed at the bottom of a page, nor the last line at the top of a page, except in the case of single-line paragraphs.

When a paragraph does not start at the top of a page, an amount of vertical space is output above it such that the new paragraph is preceded by at least the amount specified by the **parspace** directive. Style definition files normally select a default value appropriate to each printing device, but this can be changed any number of times. For example,

```
.parspace 36
```

specifies that, from now on, at least 36 points of space are to precede each paragraph.

A temporary indent is set for the first line of each paragraph. The size of this is controlled by the **parindent** directive, and again the style definition files select an appropriate default. It is also possible to override the indent for one individual paragraph by means of the **tempindent** directive (see below). Normal typographic convention is to use *either* blank space *or* an indent to signify a new paragraph, but not both.

If it is necessary to force a new line of output without treating it as a new paragraph, the **newline** directive can be used. This can optionally be followed by the word 'justify', which requests that the line just terminated be right-justified, that is, stretched out to end flush with the right-hand margin.

5.2 Chapters, sections, subsections and sub-subsections

Paragraphs of text are usually grouped into larger units, and SGCAL has provision for up to four different levels, not all of which need be used. For example, a novel may consist only of chapters, whereas a technical note may be divided only into sections. The start of each portion of the text is indicated by one of the following directives:

```
.chapter <title>
.section <title>
.subsection <title>
.subsubsection <title>
```

For example

```
.chapter SGCAL Markup for Running Text
.section Paragraphs
```

The way in which the titles are printed is determined by the macros and flags in the style definition file: For example, when formatting for a line-printer, chapter titles are printed in upper case, but on a laser printer they are printed in large type. In both cases they are centred and placed at the start of a new page.

If a chapter or section title starts with a double quote character, it is necessary to include the entire title in double quotes, and to double up any double quote characters that are in it. For example:

```
.section """Special"" features"
```

Titles can always be enclosed in double quotes, but they are mandatory only in this special case.

5.3 Indentation

Automatic indentation at the start of each paragraph is controlled by the **parindent** directive, as described above. Paragraph indents are relative to the overall indentation, which is set by the **indent** directive. For example,

```
.indent 6 em
```

specifies an indentation of six ems in the current font, as has been done for this text. An *em* is a traditional printer's unit of horizontal width, approximately the size of a capital 'M'. In SGCAL it is taken as the width of an exact space. Indents may also be specified in points, inches, or centimetres:

```
.indent 36
.indent 1 in
.indent 2.54 cm
```


The paragraph indent as set by **parindent** is taken relative to the overall indent, and when there is a positive indentation set, the paragraph indentation may be set to a negative value, giving the effect demonstrated here. Other effects can be obtained by using the *temporary indent* facilities, of which the paragraph indent is really just a special case.

It is sometimes necessary to specify indents relative to the current indent. This can be done by writing an arithmetic expression as part of the **indent** directive, and using the text '`~~sys.indent`' to represent the current indent. For example,

```
.indent ~~sys.indent + 24
```

This is an example of the use of the insert flag '`~~`' to insert the value of an SGCAL *variable* into the text. The name of this particular variable is **sys.indent**. Variables whose names begin with **sys.** are *system variables*, and their contents are automatically maintained by SGCAL. The contents of **sys.indent** are defined to be the current indentation value, expressed in points.

The directive

```
.tempindent <e1> <e2>
```

where **<*e1*>** and **<*e2*>** are arithmetic expressions, specifies an indentation of value **<*e1*>** which lasts for the next **<*e2*>** output lines. If **<*e2*>** is omitted, the temporary indent lasts for one line only. For example, the directive

```
.tempindent 4 em 2
```

has been used here. In effect, the directive

```
.tempindent ~~sys.indent + ~~sys.parindent
```

is automatically inserted at the start of every paragraph. A subsequent explicit **tempindent** directive can override this, making

```
.newpar
.tempindent <e1> <e2>
```

have a completely different effect to

```
.tempindent <e1> <e2>
.newpar
```

in which the **tempindent** directive is cancelled by the subsequent **newpar**.

It is often necessary to place text in the indent space at the start of paragraphs. This can be done by specifying a temporary indent of zero. Following the special text, the flag '`$i`' is used to restore the alignment.

This is one of SGCAL's *tab flags*, more details of which are given in a later section; it tabs to the current overall indent. An example of its use is as follows:

```
.indent 6 em
.tempindent 0
(1)$i This is the start of a numbered paragraph.
```

5.3.1 Enumerated paragraphs

Three common cases of paragraph indentation have been 'wrapped up' into a macro command which allows paragraphs to be numbered, lettered, or 'bulleted' with an arbitrary character, respectively. The directive **numberpars** indicates the start of a sequence of 'numbered' paragraphs, indented relative to the current indent. Each subsequent paragraph must be marked by the directive **nextp**, and the sequence is ended with **endp**. If **numberpars** is used on its own, the paragraphs are numbered with ordinary arabic numerals which are placed in brackets in the indent of the first line of each paragraph. For example, the input

```
.numberpars
This is the first numbered paragraph. If
it is long enough to require more than one line,
they are all indented by the same amount.
.nextp
The is the second numbered paragraph.
.endp
```

produces the following output:

(1) This is the first numbered paragraph. If it is long enough to require more than one line, they are all indented by the same amount.

(2) This is the second numbered paragraph.

Blank space is always left between the paragraphs, and there is additional space before the first and after the last. If roman numerals are required, one of the words 'ROMAN' or 'roman' can be added to the **numberpars** directive:

```
.numberpars roman
```

This gives lower case roman numerals; 'ROMAN' gives upper case ones. Similarly, to obtain lettered rather than numbered paragraphs, the **numberpars** directive is followed by 'alpha' or 'ALPHA'. Finally, if an arbitrary character sequence follows **numberpars** it is used to mark each paragraph. For example

```
.numberpars *
```


marks each paragraph with an asterisk. Calls to **numberpars** can be *nested*, causing 'inner' paragraphs to be further indented, as shown in this example:

```
$bf{Check List for Monday, 10th June}
.blank
.numberpars
Check that
.numberpars *
Luggage contents are correctly listed;
.nextp
The luggage is labelled.
.endp
.nextp
Include a first-class stamped postcard.
.nextp
$bf{Arrival at school:}
.numberpars roman
Children should arrive at School between
8.30 and 8.45 am.
.nextp
If the coach is not already outside the gates,
luggage should be deposited outside the Craft Room.
.endp
.nextp
The coach will depart at 9.00 am.
.endp
```

The output produced is as follows:

Check List for Monday, 10th June

(1) Check that

  * Luggage contents are correctly listed;

  * The luggage is labelled.

(2) Include a first-class stamped postcard.

(3) **Arrival at school:**

  (i) Children should arrive at School between 8.30 and 8.45 am.

  (ii) If the coach is not already outside the gates, luggage should be deposited outside the Craft Room.

(4) The coach will depart at 9.00 am.

Care must be taken to match each **numberpars** with an **endp**, as otherwise strange effects occur.

5.4 White space

There is a directive called **blank** which is followed by a number, and which leaves that many units of vertical white space, unless SGCAL happens to be at the top of a page, in which case it has no effect. If no number is given, a single unit of space is left. The size of the unit depends on the style being used (and can be varied); it is normally about two-thirds of the depth of a line. There is a similar directive called **space** which leaves a fixed amount of white space, even at the top of a page, and whose meaning cannot be varied. Examples of these two directives are

```
.blank
.blank 3
.space 2 in
.space 5 cm
```

5.5 Doublespaced output

The directives **doublespace** and **singlespace** can be used to specify double and single spacing of the output, respectively. Single spacing is the default. For many of the standard styles, **doublespace** does not give literally double the spacing by default, as this often looks too deep.

5.6 Hyphenation

If a word contains a hyphen character, it is a candidate for splitting across a line break. There is also a *conditional hyphen* flag, '~'. Its occurrence in a word indicates an allowable splitting point. If the word is in fact split, a hyphen is inserted; if it is not, the word is closed up. Words may contain any number of hyphens or conditional hyphens, for example

```
...  photo~type~setter type-faces ...
```

To prevent a word from being split at a particular hyphen character, precede it with the quote flag '`@`'.

Words which contain neither real nor conditional hyphen characters may be split across linebreaks by means of SGCAL's hyphenation dictionary. Hyphenation occurs only when the line is sufficiently 'loose', that is, when the amount of space in the line is large relative to the number space positions.

5.7 Horizontal lines

Printers call straight lines (whether horizontal or vertical) 'rules'. SGCAL has flags for generating both kinds of rule, though vertical rules are not supported for plain output. For the simple case of a horizontal line right across the page, there is a directive called **rule**.

6. Notes, emphasis and indexing

6.1 Footnotes

Footnotes are placed in the input text at the point they are referenced, and SGCAL automatically supplies numbers for them. The numbers are reset at the start of each chapter. The text of each footnote is enclosed between the directives **footnote** and **endf**, as in the following example:

```
Footnotes are normally printed at the bottom
of each page
.footnote
Like this.
.endf
and separated from the text by a short line.
```

Footnotes are normally printed at the bottom of each page[1] and separated from the text by a short line. The styles in which the reference numbers are printed, in the body of the text and at the start of the note, are specified by flags which can be changed by the user. Details are given below in the chapter on changing the standard style.

6.2 Emphasis

A particular form of marginal annotation commonly found in manuals and other documents that undergo revision is the 'emphasis bar', which is a short vertical line printed in the margin to indicate where a document has been changed. SGCAL is set up so that any output lines containing sections of the text appearing between the directives **em** and **nem** are marked in this way. For example,

```
.em
The line containing this text will be emphasized.
.nem
```

6.3 Indexing

The directive **index** causes the text that follows it on the same line, together with the page number, to be written to the index file. This must subsequently be sorted and consolidated by some means before being reprocessed by SGCAL for printing.

[1] Like this.

7. Displayed Text

Display is a printer's word for material that is not part of the running text of a document, but instead is 'displayed' in some special fashion.

7.1 In-line displays

In SGCAL, displayed material that must be printed in sequence with the main text is enclosed between the two directives **display** and **endd**. When processing a display, SGCAL does not perform its normal line filling, but instead copies the input to the output, line for line. In addition it ensures that the display does not cross a page break.

At the start of a display the current line length, indent, and so on are the same as in the immediately preceding text, but any changes that are made inside the display are automatically cancelled when it is completed. The font is automatically changed to a suitable fixed pitch font, but this can be changed by the use of '`$rm{`', '`$it{`', etc. At the end of a display the font reverts to what it was previously. Here is an example of the input to generate a display:

```
.display
  1 + 2 + 3 + 4 + 5 = 15
.endd
```

By default, SGCAL flags are still interpreted in displayed material, so changes of font can still be made and tabs (see below) can be used. However, it is sometimes useful to be able to disable this processing, for example, when including text from other sources. A common case is the inclusion of computer program fragments, which often make use of the special characters that introduce SGCAL flags. If the word 'asis' is added to the **display** directive, the displayed text is not scanned for embedded flags. However, SGCAL directive lines are still recognized – this is necessary, of course, in order that **endd** should terminate the display. Here is an example that displays a fragment of a BCPL program. Without the use of 'asis', the '`$`' and '`#`' characters would need to be preceded by '`@`'.

```
.display asis
LET hypot(a, b) = VALOF
$(
LET hh = a#*a + b#*b
RESULTIS realsqrt(hh)
$)
.endd
```

It is sometimes desirable to allow very long displays to flow over page breaks. To do this, add the word 'flow' to the **display** directive:

```
.display flow
<many lines of input>
.endd
```

The first six lines of such a display always appear on the same page. If both 'flow' and 'asis' are needed for a display, 'asis' must come second.

```
.display flow asis
<many lines of verbatim input>
.endd
```

Even though displayed material is copied line for line from input to output, it is possible to request that individual lines be justified, that is, stretched out to end flush with the right-hand margin, by increasing the sizes of the spaces in the line. The **newline** directive with the **justify** option achieves this.

7.2 Figures and tables

The form of display described above appears in the output at the same point as it appears in the input. For large amounts of display text it is sometimes more convenient to use a *floating display* which appears near but not necessarily at the point of input. Such displays are normally figures with captions, and SGCAL has two directives for processing them which are used as in the following example:

```
.figure This is the figure's title.
*****************************************************
The text for the figure, normally quite a few lines.
*****************************************************
.endfigure
```

The output from this example appears as figure 1 below.

****
The text for the figure, normally quite a few lines.
****

Figure 1: This is the figure's title.

Notice that SGCAL has supplied a figure number automatically. The reference to the figure was generated by including the following text *before* the figure:

```
... appears as figure ~~figurenumber below.
```


This is an example of the use of the insert flag '~~' to insert the value of an SGCAL *variable* into the text. The name of this particular variable is **figurenumber**.

The **endfigure** directive can be followed by a dimension. This specifies an amount of extra white space that is to be left below the figure title.

If there is room on the current page for the figure, it is output immediately. Otherwise it and any subsequent figures are held over to the top of the next page (even if the subsequent figures would in fact fit).

Another pair of SGCAL directives is **table** and **endtable**. They work in exactly the same way as figures, except that tables have their own sequence of numbers, held in the variable **tablenumber**. The fact that a figure is being held over to the next page does not force tables to be held over (and *vice versa*).

7.3 Subscripts and superscripts

Displayed material often requires the use of subscripts and superscripts. Three SGCAL flags are available for this purpose:

(1) '`$U`' is the up flag, which causes subsequent text on the line to be raised with respect to what went before;

(2) '`$D`' is the down flag, which causes subsequent text on the line to be lowered with respect to what went before;

(3) '`$L`' is the level flag, which causes subsequent text on the line to be printed at the normal line level.

The ability to print characters above and below the line is only available on certain output devices. If the **printer** style is being used, then these flags have no effect. Here is a typical example of their use:

```
.display
H$D2$UO + SO$D$$3$L --> H$D2USO$D$$4
.blank 2
e = mcU$$2
.endd
```

The output is

$H_2O + SO_3 \rightarrow H_2SO_4$

$e = mc^2$

Note the use of the null flag '`$$`' to terminate these flags when a digit follows. This is necessary because if a number follows the up or the down flag, it is taken as the number of points to move up or down. If no number is present, the distance moved is one third of the current line depth.

7.4 Tabs

Several different kinds of tab are available in SGCAL. The *indent tab* has already been described above: it tabs to the current indentation position, and is represented by the flag '`$i`'.

A conventional tab operation is represented by the flag '`$t`' in the input. Tab positions are set by the **tabset** directive, which is followed by a list of column widths. Here is an example of some tabbing input:

```
.display
.tabset 7em 11em 7em 9em
1234567890123456789012345678901234567890
X $t XX $t XXX $t XX $t X
.endd
```

The output that is produced is:

```
1234567890123456789012345678901234567890
X      XX          XXX    XX        X
```

Any space characters in the input preceding or following the tab flag are ignored. The tabs in the example above are *left* tabs, because the portions of text are set with their left-hand edges at the tab positions. SGCAL also supports *right* tabs and *centred* tabs, and these are set in the **tabset** directive by typing the letters 'R' or 'C' following the tab position, respectively. (It is also permitted to type 'L' to indicate a left tab explicitly.) Here is an example showing all three kinds of tab:

```
.display
.tabset 7em 13em R 11em C 14em
12345678901234567890123456789012345678901234567890
X $t XX $t XXXX $t XXX $t XX
.endd
```

which produces the following output:

```
12345678901234567890123456789012345678901234567890
X      XX        XXXX          XXX              XX
```

Finally, there are two special tabs that are useful in headings and footings, the *centring* tab and the *ending* tab, represented by the flags '`$c`' and '`$e`', respectively. Any text following the ending tab on the input line is moved to the right until it ends at the right-hand margin. Any text following the centring tab, up to the next tab if present, or to the end of line if not, is centred in the current line width. Thus the input


```
.display
the left $c the middle $e the right
.endd
```

produces the following output line:

the left   the middle   the right

7.5 Heads and feet

Headlines are lines that appear at the top of a page, above the main text, and similarly *footlines* appear at the bottom. They are used for page numbers and titles of various sorts. The standard arrangement in SGCAL is to have two footlines, the first being blank and the second containing a centred line number, and no headlines.

The directive **nofoot** can be used to suppress the footlines altogether. Alternatively, the size of the headline or footline areas can be changed by the **headdepth** and **footdepth** directives. Thus, for example,

```
.headdepth 3 ld
.footdepth 4 ld
```

specifies three headlines and four footlines instead of the default zero and two. The abbreviation 'ld' after a number stands for 'line depths'. The material that appears in the headline area is defined between the directives **head** and **endhead**, for example

```
.head
The $c First $e Headline
.newline
Another $e Headline
.endhead
```

and similarly footlines are defined between **foot** and **endfoot**. There may be more than one occurrence of these directives in an SGCAL input file; at the start or end of a page the most recent definition is used. If there are not enough headlines to fill the headline area, the bottom is left blank; if there are not enough footlines to fill the footline area the top is left blank.

The most common use of headlines and footlines is for printing the page number, and this can be done in SGCAL by including the text '`~~sys.pagenumber`' where the page number is required.

Another useful system variable is **sys.date**, which can be used to insert the date of processing into the text, as in the following example of footline definition:

```
.footdepth 3 ld
.foot
$c [~~sys.page] $e ~~sys.date
$bf{FIRST DRAFT}
.endfoot
```

It is possible, by using the more advanced features of SGCAL, to cause the page numbers to be printed in roman numerals, to specify different headlines and footlines for odd-numbered and even-numbered pages, and to create 'running' headlines and footlines.

8. Advanced Features

The facilities described in previous chapters are enough to cope with many text processing jobs. There is, however, a lot more to SGCAL. In this chapter some ways of changing the standard actions described above will be covered, as well as some additional features. New users are advised to become familiar with the preceding material before reading further.

8.1 Variables

The directives for handling chapters, sections, footnotes and so on make use of *variables* for counting. Many other features of the standard styles are controlled by variables. Additional variables can be defined and used for many purposes, some examples of which are given below. The **set** directive is used to define a variable and its contents, which may be a number or a string of characters. One simple use of a variable is to hold a long string that appears many times in the document, for example:

```
.set scal "supercalifragilisticexpialidocious"
```

This can then be inserted whenever required by simply typing

```
~~scal
```

wherever it is needed.

8.2 Changing a standard style

In this section some of the variables and flags which control the standard styles are described. The may all be changed by the user in order to achieve a different effect.

8.2.1 Numbering chapters and sections

The variables which contain the current chapter, section, subsection and sub-subsection numbers are called **chapter**, **section**, **subsection** and **subsubsection**. These can be used in several ways:

(1) If any of them is set to contain a negative number, the numbering of the relevant item is suppressed. Thus

```
.set chapter -1
```

switches off chapter numbering, while leaving section, subsection and sub-subsection numbering on. It is sometimes useful to turn chapter numbering off for a preface, then turn it on again (by setting the variable **chapter** to zero) just before the call to the **chapter** directive for chapter 1.

(2) A long document can be processed in parts by setting the appropriate chapter or section number at the head of each part.

(3) A current number can be saved in a different variable for later use in a backwards reference, or output as part of an index entry.

8.3 Display indentation

Displays are automatically indented by an amount specified by the variable **displayindent**, which defaults to zero. Thus to indent all displays by 36 points, the following directive would be used:

```
.set displayindent 36
```

The indent is always reset on exit from a display.

8.3.1 White space

The amount of vertical white space surrounding headings can be controlled by changing the following variables:

`chapspaceb`	space after chapter headings
`sspacea`	space before section headings
`sspaceb`	space after section headings
`ssspacea`	space before subsection headings
`ssspaceb`	space after subsection headings
`sssspacea`	space before sub-subsection headings
`sssspaceb`	space after sub-subsection headings
`fnspace`	space between footnotes

8.3.2 Heading styles

The styles in which headings appear (large type, bold face, etc.) are controlled by flags which are placed before each heading text. They are:

`$chead{`	chapter heading
`$shead{`	section heading
`$sshead{`	subsection heading
`$ssshead{`	sub-subsection heading
`$fkt{`	footnote key in text
`$fkn{`	footnote key for note
`$ftitle{`	figure title
`$ttitle{`	table title

The heading texts are always terminated by a closing curly bracket. Before re-defining one of these flags, it is first necessary to cancel it using the **cancelflag** directive. Thus, for example, to arrange for section headings to be in sanserif type:

```
.cancelflag $shead{
.flag $shead{ "$ss{"
```


The variable **hnspace** contains a string to be inserted between the number of a chapter, section or subsection and its title. By default this string is a single space, but it can be changed as required.

8.4 Number formats

Page, chapter, section and subsection numbers are normally printed in arabic numerals. Roman numerals can be requested by means of the **format** directive, which is followed by the name of the relevant variable and one of the words 'roman' or 'ROMAN' for lower or upper case numerals, respectively. The page number is held in a *system variable* called **sys.pagenumber**, because it is incremented automatically inside the SGCAL program. For example,

```
.format chapter ROMAN
.format sys.pagenumber roman
```

would number chapters in uppercase roman numerals, and pages in lower case. For numbers in the range 1–26, an alphabetic format is also available, again in either case, specified by the letters 'alpha' or 'ALPHA' in a **format** directive.

8.5 Varying heads and feet

Two common requirements for headlines and footlines are the inclusion of the current chapter or section title, and varying the text on alternate pages. The first of these can easily be done by making use of the variables **chapname**, **sectname** and **ssectname**, which are automatically maintained by SGCAL. The second requires the use of SGCAL's conditional directive, **if**. This is a powerful facility which can be used to test many conditions; the simple requirement here is to test whether the current page number is odd or even. Its use is best demonstrated by an example, which places the current chapter title on the left at the top of even-numbered (left hand) pages, and the current section title at the top of odd-numbered pages. In both cases the titles are forced to be in the sanserif typeface.

```
.head
.if even ~~sys.page
$ss{~~chapname}
.else
$e$ss{~~sectname}
.fi
.endhead
```

The two parts of the conditional construction are separated by **else**, and the whole thing is terminated by **fi**. Nested conditions are permitted.

When a new chapter or section is being started, SGCAL sets the name variable to the null string until the title has been output. The above example would therefore result in a null headline at the start of a chapter and whenever a section started at the top of a page (which is normally what is wanted).

8.6 Thin and wide spaces

The thin space flag, '`$<`', can be used to generate a small amount of horizontal space, with a width approximately one sixth of a normal space (except of course for line-printers and other devices that do not have variable spaces). There is also an 'extra-stretchy' space flag, '`$<>`', which has the width of an ordinary space, but which, if it appears in a line which is being right-justified, absorbs all the stretch, leaving the other spaces unaltered. This can be used to force text to the right hand side of the page at the end of a paragraphs, as follows:

```
This is the text of the paragraph.
The quick brown fox jumps over the lazy dog.
Use the extra-stretchy space at the end,$<>thus.
```

The output produced is:

This is the text of the paragraph. The quick brown fox jumps over the lazy dog. Use the extra-stretchy space at the end,   thus.

* * *