SCRIBE

INTRODUCTORY USER'S MANUAL

FIRST EDITION

Brian K. Reid

July 1978

CARNEGIE-MELLON UNIVERSITY
COMPUTER SCIENCE DEPARTMENT

This manual corresponds to SCRIBE version CMU 0E(51).

The research that produced SCRIBE was funded in part by the Rome Air Development Center under Contract No. F306-2-75-C-0218, in part by Army Research Contract No. DAAG29-77-C-0034, and in part by the Defense Advanced Research Projects Agency under contract No. F44620-73-C-0074.

Disclaimer

This is a preliminary version of the SCRIBE manual. It is being circulated so that there will be some manual available, however incomplete, and to get comments from users about its clarity and content.

If you have comments about the accuracy of this manual, you may comment by sending mail to Reid@CMU-10A. If you have comments about its readability, clarity, content, or style, please send comments to the same address.

Some of the information in this manual describes system features that are not fully implemented as the manual goes to press. Let's just call these "lies". As far as I know, this is an exhaustive list of lies in the manual:

- Use of @k to get special characters on the XGP does not work yet. It only works on the Diablo and the LPT.
- SCRIBE does not yet produce the promised list of "faked" special characters to tell you where to write these characters in by hand.
- The LetterHead and MemoHead document types do not work on the XGP. They do work on the Diablo.
- The bibliography mechanism does not work as advertised. An interim bibliography mechanism is available; contact me for details.
- The @Part command is not fully debugged.

I hope to get all of these features fully operational before the final draft of this first edition manual is produce in September 1978.

Brian K. Reid

Preface

This manual is a beginner's manual for the SCRIBE document production system developed in the Computer Science Department at Carnegie-Mellon University.

The reader is expected to have used (however briefly) a timesharing computer, to know how to use a text editor, and to understand the notion of a timesharing "file system". No particular computer expertise or knowledge of programming is needed. If you feel that you don't have that background, see the CMU Computer Science Department's Computer Primer. On the other hand, if you have a lot of programming experience and would like to know more, there is a SCRIBE Expert's Manual available, as a companion to this one.

SCRIBE is still under development. This first edition of the Introductory User's Manual was written for internal use in the Computer Science Department, and the examples are taken from the documents that computer scientists produce.

This manual was produced with SCRIBE and printed on the Computer Science Department's Xerox Graphics Printer.

1. Introduction

SCRIBE is a document production program. It produces documents (designed to be read by humans) from manuscript files which are primarily to be read by the computer. The manuscript files that SCRIBE processes are created by human beings using a text editor; the document files that SCRIBE produces are printed on printing devices to produce paper copy.

SCRIBE is a big program. It has many exotic capabilities, most of which you will probably never need. But as the people who sell cars with big engines are fond of saying, it's nice to have the power there in case you need to use it.

Unlike many big programs, SCRIBE looks simple. You don't need very many commands, and the ones that you do need are not complicated. If you have used computer text editors before, you should be able to learn enough about SCRIBE in an hour to be able to produce your first document. As you learn more about document production and find that you want more "features" or more power, just read farther in the manual and you will probably find what you are looking for.

The SCRIBE Expert's Manual, a companion volume to this one, describes how to do complicated things. We have made it a separate manual to emphasize the fact that SCRIBE is quite useful to people who aren't experts, and also to save printing costs, since not everybody wants to be an expert.

1.1. Some Explanation for Non-Programmers

The preparation of a document requires, in addition to the actual

text, an understanding of how the text is to be formatted. When a secretary types a document, he applies his knowledge of typing and formats and his understanding of what the words and sentences mean, and chooses a reasonable format for various pieces of the text. When a typographer typesets a document for publication, he needs editor's marks, in colored pencil, to show him what to do.

Computers are at a double disadvantage in the production of documents. They are not clever, like secretaries, nor can they read penciled proofreader's marks, like typographers. Computers can recognize the 95 characters on their keyboard, but not penciled notes. Somehow, using only those 95 characters, we must devise a means of communicating to the computer all of the various kinds of information that secretaries can figure out for themselves and typographers get from the pencil marks. SCRIBE has such a scheme, using special sequences of characters to represent formatting commands, and if it sometimes seems to you as though a certain construct is too baroque or too mathematical, please understand why it is that way.

1.2. Some Explanation for Programmers

SCRIBE is a one-pass program that is table-driven and configurable; the language that it processes is non-procedural. It can be run in an idempotent processing mode or in regular "compilation" mode.

From its input files (manuscript files), SCRIBE can produce two outputs: an output file, suitable for printing on the appropriate printing device, and an updated version of the input manuscript file. The idempotence property prescribes that if SCRIBE is used to update

the same manuscript file twice in a row, then the second update will not actually change the file. The line and page breaks in the updated manuscript file will correspond to the line and page breaks in the finished output file.

SCRIBE does not have "commands" in the usual sense of the word: its commands are not procedural. A SCRIBE command specifies the result that is desired rather than the method to achieve it. For example, to enter a quotation in running text, it is customary to switch to single spacing, indent the left and right margins, and then change back again to normal margins and spacing. In SCRIBE, this would be achieved by placing a "@begin(quotation)" command at the front of the quotation's text, and a "@End(quotation)" after its text. The details of the knowledge of how to print a QUOTATION are encoded in SCRIBE's internal tables.

SCRIBE is configurable, in the sense that users may change, replace, or augment the formatting knowledge contained in its tables. While radical changes to SCRIBE's behavior cannot be achieved by changing the tables, nearly any conventional text effect can be achieved. Most users with uncomplicated needs will never want to use the configuration mechanisms. However, for those with complicated needs or the urge to tinker, there is a companion manual, the Scribe Expert's Manual, that explains all the many details of modifying and configuring the system.

2. A Beginner's Guide

This chapter is for new users, who have never used SCRIBE before and who would like to learn how to use it for simple things.

2.1. Preparing Input for Scribe

A manuscript file prepared for input to SCRIBE consists primarily of plain text. Paragraphs are separated by blank lines, and commands, if present, are flagged with an "@" sign. If you want an "@" sign to appear in your output, you must type two of them: "@@". The various commands are described in this and following chapters. A file containing no commands at all will work just fine; it will yield simple justified paragraphed text. On the PDP-10, a manuscript file should be given the extension ".MSS"; our example that we discuss here will be called TRIAL.MSS.

2.2. Running Scribe

Suppose that we have built a manuscript file named TRIAL.MSS, with the following contents:[1]

```
00100 So when he came to the churchyard Sir Arthur alighted and
          tied his
00200  horse to the stile.  And so he went to the tent and found
          no knights
00300 there for they were at jousting.  And so he handled the
          sword by the
00400  handles and lightly and fiercely pulled it out of the
          stone, and took
00500 his horse and rode his way until he came to his brother,
          Sir Kay, and
00600 delivered him the sword.
00700
00800  And as soon as Sir Kay saw the sword he wist well it was
          the sword
00900 from the stone, and so he rode to his father, Sir Ector,
```

---

[1] From Le Morte Darthur, by Sir Thomas Malory.

```
            and said,
01000 `Sir, lo! here is the sword of the stone, wherefore I must
            be King of
01100 this land.'
```

To process this file, we run SCRIBE; it will prompt us for input by typing an asterisk:

```
R SCRIBE
*
```

In response to the prompt, type the file name of the manuscript file that is to be processed.

```
R SCRIBE
*TRIAL.MSS
```

SCRIBE will then process the file. It will produce as output a file called TRIAL.LPT. This TRIAL.LPT file that it produces, when typed on the terminal, will look something like this:

> So when he came to the churchyard Sir Arthur alighted and tied his horse to the stile. And so he went to the tent and found no knights there for they were at jousting. And so he handled the sword by the handles and lightly and fiercely pulled it out of the stone, and took his horse and rode his way until he came to his brother, Sir Kay, and delivered him the sword.
>
> And as soon as Sir Kay saw the sword he wist well it was the sword from the stone, and so he rode to his father, Sir Ector, and said, `Sir, lo! here is the sword of the stone, wherefore I must be King of this land.'

The reader is invited to copy the file TRIAL.MSS[4,5621][1] and go through this exercise.

2.3. Printing Devices

When SCRIBE processes a manuscript file into a document file, it does so with a particular printing device in mind. The device characteristics determine the number of characters that SCRIBE tries

---

[1] Only at SRI.

to put on a line, the number of lines that it tries to put on a page, the methods used to accomplish underlining and backspacing, and so on.

If you don't tell SCRIBE anything one way or another about printing devices, it will assume that you are preparing a file for the line printer. If you aren't, you must put a @DEVICE command at the beginning of your manuscript file, telling SCRIBE what device you are using. The format is "@DEVICE(what kind)"[1]; some examples follow:

@DEVICE(LPT) (the default device) Prepare for a line printer

@Device(XGP) Prepare document for Xerox Graphics Printer

@DEVICE(DIABLO) Prepare for Diablo printer

@device(FILE) Prepare as a computer file

To return to our example of the previous section, let's change it to produce output for the XGP. The updated manuscript file TRIAL.MSS would now look like this:

```
00050 @device(XGP)
00100  So when he came to the churchyard Sir Arthur alighted and
           tied his
00200 horse to the stile.  And so he went to the tent and  found
           no knights
00300  there  for  they were at jousting.  And so he handled the
           sword by the
00400 handles and lightly and fiercely  pulled  it  out  of  the
           stone, and took
00500  his  horse and rode his way until he came to his brother,
           Sir Key, and
00600 delivered him the sword.
00700
00800 And as soon as Sir Kay saw the sword he wist well  it  was
           the sword
00900  from  the stone, and so he rode to his father, Sir Ector,
           and said,
01000 "Sir, lo! here is the sword of the stone, wherefore I must
```

---

[1] Upper and lower case may be used interchangeably in any SCRIBE command; varying case will be used throughout this manual to emphasize that.

```
                be King of
01100 this land.'
```

If this file were now run through SCRIBE, it would produce an output file named TRIAL.XGO suitable for printing on the XGP, instead of the TRIAL.LPT that it produced before.

The complete set of DEVICE codes known to SCRIBE is shown below.

Code	Page Length	Page Width	Description
LPT	53 lines	132 cols	Line printer (standard)
XGP	11 inches	8.5 inches	Xerox Graphics Printer
DIABLO	11 inches	8.5 inches	Diablo HyType printer
SOS	50 lines	69 cols	SOS file
FILE	50 lines	79 cols	Standard DOC file
CRT	24 lines	79 cols	Pages match CRT screen size
LA36	66 lines	132 cols	LA36 DECwriter full width
DECwriter	66 lines	80 cols	LA36 DECwriter narrow width
TI700	50 lines	80 cols	Texas Instruments 700

We expect to be able to add a device type for a photocomposing machine as soon as one becomes available to us.

2.4. Characters and Keyboards

SCRIBE's normal mode of operation is to take the characters that you type into the manuscript file and see to it that they appear in the finished document. If SCRIBE did this faithfully for all characters, then it would have no way of recognizing commands, for the "@" sign would be just another character and not a special flag saying "a command follows". The special-character commands cause SCRIBE to insert a special character into the text in place of the command; another command will leave space for you to write in the character by hand.

2.4.1. Getting SCRIBE to Print Special Characters

Remember that a computer keyboard normally has only 95 characters on

it. People frequently want to include special characters, whether a Greek letter like P or z, or a mathematical symbol like ?< or ?7. These special characters are not available on the keyboard as individual keys. SCRIBE lets you specify special characters via multiple-character "escape sequences". They are called "escape sequences" (a piece of computer jargon) because you escape from the normal meaning of the character. For example, if your manuscript file contains

```
if A @k(leq) B then @g(L) must be 0
```

then your finished output will contain

```
if A ?< B then L must be 0
```

The @g(L) to print a lambda is a "font-change" code, which changes to the Greek font. These are described in section FONTCODES. The @k(leq) is a special-character escape sequence to generate the less-or-equal character. Appendix SPECIALCHARS has a table of special characters and what you must type to get them.

2.4.2. Faking Special Characters

If the printing device that you are using cannot print the character you have asked for, SCRIBE will leave a blank spot there so that you can put the character in by hand. If you don't want to learn how to produce special characters, you can do the same thing: just leave a blank spot in your manuscript and then write the character in with a pen. Anywhere you put the sequence "@#", SCRIBE will leave a blank space large enough to write one character. Thus, if your manuscript file contains

```
Kaiser Wilhelmstra@#e
```

then SCRIBE will replace that "@#" by a wide space, printing:

    Kaiser Wilhelmstra e

whereupon you would take pen in hand and write in the "?#" character:

    Kaiser Wilhelmstra?#e

It you will need to write special characters in by hand, SCRIBE will produce a list that shows you where this must be done; you need just sit down with the list and a black pen and write the characters in.

3. Preparing a Manuscript

You now know how to build a manuscript file that contains plain text, and then run it through SCRIBE to produce a document. In this section we will tell you how to fancy it up with italics, underlining, subscripts, superscripts, and so forth.

Let's start with an example, then get to a more general explanation. To get italics,[1] surround the letters that you want italicized with @i[...], like this:

The inscription read, "@i[De minimus non curat lex]."

which will produce output that looks like this:

The inscription read, "De minimus non curat lex."

Before going on with more explanations, we need to talk some nuts and bolts.

3.1. Delimiters

The square brackets [] used in the example above are called delimiters. They delimit the characters being italicized. Computer keyboards contain several pairs of matching left and right delimiters, and any of them may equally well be used. For example, you could have put @i(De minimus..) or @i<De minimus..>. It doesn't really matter which kind of delimiter you use as long as the closing delimiter matches the opening delimiter. If you tried @i<De minimus...], SCRIBE will just keep italicizing until it comes to a ">" character or the

---

[1] If this document was printed on a device not capable of italicizing letters, then you will see underlining or overstriking where we intend italics.

end of the paragraph. The delimiter pairs that you may use are (...), [...], {...}, <...>, '...', `...', and "...".

Suppose that you wanted boldface and underlining of the same text. To get boldface, you use @b[...] around the text to be boldfaced. To get underlining, we put @u[...] around the text to be underlined.

Why would anybody want @b[@i[boldfaced underlining]]?

This is called nesting of the delimiters. The concept of nesting should be familiar to everyone in the notion of quotations inside quotations; the inner quotation is delimited with single quotes and the outer quotation is delimited with double quotes, like this:

Macomber said, "But the sign said `No Hunting.'"

SCRIBE is clever enough to let you nest like delimiters without getting itself confused, but you can use different delimiters if you want:

Why would anybody want @b<@i[boldfaced underlining]>?

If you want to include an un-nested delimiter character inside a pair of delimiters, you have to be careful. Suppose, for example, that you wanted to put an underlined ")" character in your text. If you used parentheses as delimiters, and put @u()), SCRIBE would treat that as "@u()" followed by ")", and would not do what you wanted. This might seem like a useless warning, but the situation comes up in mathematical formatting. Equations and formulas are frequently printed with italicized variable names, and frequently contain parentheses. To get the sequence "2*(4+y)-b" correctly italicized, we must be careful not to use parentheses as delimiters: @i[2*(4+y)-b] will work, but @i(2*(4+y)-b) will not work.

3.2. Printing Device Considerations

Not every feature provided by SCRIBE can be realized on every output device. The line printer, for example, cannot print italics or even underline, but it can generate boldface by overstriking. The XGP at CMU can italicize and underline and boldface, but cannot do italics and boldface on the same line. In fact, our XGP is restricted to no more than two fonts in any pair of lines. Some of the features in SCRIBE cannot be achieved on any of the printing devices currently available to it, but were installed in anticipation of a photocomposing machine or better Xerographic printers. For example, the overbar code @o[] and the script code @s[] cannot be printed on even the XGP without special tinkering that is beyond the scope of this manual.

3.3. Italics, Underlining, Etcetera

SCRIBE recognizes all of the following font-change codes. Whether it will actually produce the requested font change depends on the nature of the final printing device being used. If SCRIBE cannot produce a particular code on a particular printing device, it will attempt a compromise. In general, if a final printing device cannot print a particular effect, SCRIBE will generate underlining or overstriking instead.

SCRIBE recognizes this set of font-change codes:

@i[phrase]	Italics
@u[phrase]	Underline non-blank characters
@ux[phrase]	Underline all characters
@un[phrase]	Underline alphanumerics

@b[phrase]	Boldface
@r[phrase]	Roman (the normal typeface)
@t[phrase]	Typewriter font
@+[phrase]	superscript
@c[phrase]	SMALL CAPITALS
@-[phrase]	subscript
@g[phrase]	Greek (Ellen)
@o[phrase]	Overbar

3.4. Inserts

You now know how to produce paragraphed text with various fancy effects like italics and underlining. In this section, we will show you how to insert various things into the middle of running text. Anything put into running text is called an insert. Inserts can be quotations, examples, equations, tables, blank space, or even pictures. The text will resume in the same paragraph after the insert.

3.4.1. Specifying an Insert

There are two ways of specifying an insert. The long form requires more typing, but is immune to problems with delimiters. Both yield the same result. Consider this example: to put a quotation into running text, we mark its beginning with "@BEGIN(QUOTATION)" and its end with "@End(QUOTATION)". Thus:

```
@Begin(Quotation)
Body of quotation.
@End(Quotation)
```

This is the long form. The short form of the same thing would be:

```
@Quotation(
Body of quotation.
)
```

Obviously, the short form requires that the body of the quotation not contain a right delimiter. If your quotation is short, you can specify it all on one line:

```
@Quotation(Body of quotation)
```

SCRIBE will produce the same output from any of these three forms.

A quotation in running text comes out single spaced, with the left and right margins pulled in a little, like this:

> The fact is, that civilization requires slaves. The Greeks were quite right there. Unless there are slaves to do ugly, horrible, uninteresting work, culture and contemplation become impossible. Human slavery is wrong, insecure, and demoralizing. On mechanical slavery, on the slavery of the machine, the future of the world depends[1].

3.4.2. Simple Inserts

SCRIBE knows about the following collection of simple insert types. Any one of them may be used in any document type. They may be used in the long form (using BEGIN and END), or the short form, @NAME(body of insert). ENTER is a synonym for BEGIN and LEAVE is a synonym for END. Each of these insert types is described below; examples of the use of each can be seen in Appendix EXAMPLES.

QUOTATION	Text quotation
VERSE	Verse quotation
EXAMPLE	Example of computer type-in
DISPLAY	Non-justified insert. Each line is separate.
CENTER	Each line centered
VERBATIM	Characters copied exactly, without formatting

---

[1] From Oscar Wilde, The Soul of Man under Socialism, 1895

FORMAT	Hand-formatted text
ITEMIZE	Paragraphs with a tick-mark in front of each
ENUMERATE	Paragraphs with a number in front of each
DESCRIPTION	Outdented paragraphs; single spacing
EQUATION	Equation, with a number put in the right margin
THEOREM	Numbered theorem

3.4.2.1. QUOTATION and VERSE

The QUOTATION insert has already been described (previous section). It inserts prose quotations in running text.

The VERSE insert differs from the QUOTATION insert in the way it handles line overflow. If a line in VERSE mode doesn't fit, it will be split and the second part of the line will be indented a bit from the left. A VERSE insert comes out looking like this:

> Oh, East is East and West is West, and never the twain shall
> meet,
> Till Earth and Sky stand presently at God's great Judgment
> Seat;
> But there is neither East nor West, border, nor breed, nor
> birth
> When two strong men stand face to face, though they come from
> the ends of the earth[1].

3.4.2.2. EXAMPLE and DISPLAY

The EXAMPLE and DISPLAY insert types are very similar; they differ only in the type face that will be used. In either EXAMPLE or DISPLAY inserts, each line of the manuscript file will produce one line in the document. Since EXAMPLE is for showing examples of computer type-in

---

[1] Rudyard Kipling, The Ballad of East and West.

and type-out (useful in user's manuals), it will appear in a typeface that is designed to look like computer output. DISPLAY inserts will appear in the normal body type face. Thus:

This is an EXAMPLE insert

This is a DISPLAY insert

The examples were generated like this:

```
@Begin(Example)
This is an EXAMPLE insert
@End(Example)
@Begin(Display)
This is a DISPLAY insert
@End(Display)
```

If your copy of this manual was printed on a device that cannot change fonts (such as the Diablo printer), then those two lines will look nearly identical.

3.4.2.3. CENTER

A CENTER insert is similar to a DISPLAY, except that it centers its lines rather than left-justifying them. Like DISPLAY, CENTER produces one line in the document for each line of the manuscript file that is inside the CENTER.

This is a CENTER insert

If there is more than one line in a CENTER insert, each line will be centered individually: for the example we just gave, the text to generate it was

```
@Begin(center)
This is a CENTER insert
@End(center)
```

although we could just as well have used

```
@center(This is a CENTER insert)
```

If you think that you want boldface centering, and find yourself

wondering whether you should say @b(@center(Mynah Birds)) or @center(@b(Mynah Birds)), the chances are that what you are really doing is making a heading, and you should use @Heading(Mynah Birds) instead; see Chapter HEADINGCHAP.

3.4.2.4. VERBATIM and FORMAT

A VERBATIM insert is printed exactly as you type it; no justifying or moving of the text will be done. SCRIBE will switch to a fixed-width font for the VERBATIM text, in order that columns will line up properly. The @ codes for underlining, italicizing, and the like will all be processed inside VERBATIM.

FORMAT mode differs from VERBATIM only in that it uses a variable-width character set if one is available on the final printing device. In FORMAT mode, you would use the tabbing and formatting commands (described in Chapter TABSTOPCHAPTER), rather than the space bar on your terminal, to align columns and achieve the desired effects.

3.4.2.5. ITEMIZE and ENUMERATE

ITEMIZE and ENUMERATE each expect a sequence of paragraphs (separated by blank lines, as usual); and each justifies those paragraphs into inset margins and puts a mark in front of each. ITEMIZE puts a tick-mark ("- ") in front of each paragraph, while ENUMERATE puts a number in front of each. If you are not going to refer back to the numbers that ENUMERATE generates, you should use ITEMIZE instead. An ITEMIZE comes out looking like this:

- First item

- Second item

- Third and last

An ENUMERATE looks like this:

1. First item

2. Second item

3. Third and last

The input text that made those examples was identical save for a substitution of name: the itemization was:

```
@Begin(itemize)
First item

Second item

Third and last
@End(itemize)
```

while the enumeration was

```
@Begin(enumerate)
First item

Second item

Third and last
@End(enumerate)
```

3.4.2.6. DESCRIPTION

A DESCRIPTION insert is designed for the "command description" style that is so common in reference manuals. The first line of a DESCRIPTION insert will be at the left margin, and the second and remaining lines will be indented substantially, so that the word or phrase at the head of the description stands out. If three or more blanks in a row are found on the first line of a DESCRIPTION, they will be taken as a signal to tab to the left margin provided that the left margin has not been passed. Thus, this input:

```
@Begin(description)
Segment         One of the parts into which something naturally
                separates or is divided; a division, portion,
                or section.
Section         A part that is cut off or separated; a distinct
                part or  subdivision of anything, as an object,
                country, or class.
@End(description)
```

will produce this output:

Segment One of the parts into which something naturally separates or is divided; a division, portion, or section.

Section A part that is cut off or separated; a distinct part or subdivision of anything, as an object, country, or class.

3.4.2.7. EQUATION

An EQUATION insert is almost identical to a DISPLAY insert: each line in the manuscript file generates one line in the document, the left and right margins will be inset a bit, and it is printed in the standard body font. The difference is that the EQUATION insert will cause an automatically generated equation number to be printed in the right margin of each line in which a @TAG command (see section XREFS, page XREFS) appears. The @TAG command is used to mark cross-reference points, and only those equations that will be cross-referenced need be numbered.

This input text:

```
@begin(equation)
@i[x]@-[n+1] = @i[x]@-[n]-@i[F](@i[x]@-[n])/@i[F]'(@i[x]@-[n])
@r(or)
@tag(Newton)@i[x]@-[n+1] = (@i[x]@-[n]+@i[c]/@i[x]@-[n])
@End(equation)
```

might produce output that looks like this (notice the equation number over at the right):

$$x_{n+1} = x_n - F(x_n)/F'(x_n)$$

or

$$x_{n+1} = (x_n + c/x_n) \qquad (3\text{-}12)$$

3.4.2.8. THEOREM

A THEOREM insert is almost identical to a QUOTATION insert: it will produce a single-spaced justified block of text, with the left and right margins pulled in a bit. The difference is that each time you do a @Theorem() or a @Begin(Theorem), SCRIBE will add the prefix text "Theorem" and then print the current theorem number. Thus, this input text:

```
@Theorem(The closed interval [a,b] is compact)
...
@begin(Theorem)
A closed bounded subset of R@+[n] is compact.
@End(Theorem)
```

might produce this output text:

> Theorem 1-4: The closed interval [a,b] is compact.
>
> ...
>
> Theorem 1-5: A closed bounded subset of R^n is compact.

See section XREFS for a discussion of how to cross-reference theorem numbers using the @Tag and @Ref commands.

3.5. Footnotes

To get a footnote[1] into running text, simply insert the sequence

```
@foot(body of footnote)
```

into your text at the point where the superscripted footnote marker

---

[1] Like this one

should appear. The footnote in the previous sentence was generated by

```
To get a footnote@foot(Like this one) into
running text, simply
```

Notice that SCRIBE automatically generates footnote numbers and puts them in the proper places; don't put the numbers in yourself.

In providing such a simple footnote mechanism, we feel a responsibility to advise you to use it sparingly. Footnotes seriously interfere with the readability of a paragraph, and their excessive use will distract the reader rather than help him.

3.6. Indexing

An index could be built using the cross-reference mechanism, as it is a form of cross reference. However, SCRIBE has a built-in indexing mechanism that is simpler and that will do the alphabetization for you. It's really quite simple. At any point in the text, you may put an entry of the form

```
@index(Text to be indexed)
```

Nothing will appear in the running text output at that point, but "Text to be indexed" will appear in the index with the correct page number:

```
Text to be indexed    23
```

You may put SCRIBE commands into the index text if you want, but if you include any that cause a paragraph break you will get an unsatisfactory result. If you want an @ sign to appear in the index entry, you have to use two:

```
@Index(@@index command)
```

4. Document Types and Styles

Remembering our view that SCRIBE aspires to be a substitute for a secretary, consider the case in which a secretary is handed a rough draft of something and told to "make a letter" or "make a memo." Part of his job is to know about basic document types and to be able to produce them as needed. SCRIBE also knows about basic document types, and will produce the kind of document that you tell it to.

Each time SCRIBE is run, it produces a document of a particular "Document Type" for a particular printing device. A Document Type includes a specification of page format, paragraphing style, margins, heading and subheading style, and so forth. Some document types are very specialized, such as the one named "IEEE", which produces documents in the style and layout demanded by the various IEEE journals. Other document types are very general, such as the one named "TEXT", which produces paragraphed, justified text on numbered pages.

Novice SCRIBE users should use the TEXT document type first (it's the document type that you get if you don't ask for any particular one). As they gain more experience with the system or as they find the need to produce more complicated documents, they should try other document types.

4.1. SCRIBE's Database of Document Types

Document types are defined by entries in SCRIBE's database. When you request SCRIBE to produce a particular document type, it searches its database for the definition and then reads in the definition, thereby setting itself up to produce the requested document. If SCRIBE

cannot find in its database a document type with the name that you ask for, it will tell you and then just halt; it can't really go on.

4.2. Specifying a Document Type: the @Make command

The @Make command specifies document type. @Make(LETTER), for example, specifies the "LETTER" document type; @Make(TEXT) specifies the TEXT document type. The @Make command must come at the beginning of a manuscript file or it will not be honored. By "beginning" we mean before any text. The order of @Make, @Device, @Style, @Font, and other setup commands is not important.

SCRIBE knows about the following types of documents, which is to say that this is the list of names you can put inside a @Make command:

LETTERHEAD	A business letter with CMU letterhead.
LETTER	A personal letter; you provide the return address
REPORT	A document divided into numbered chapters, sections, subsections, and the like; an automatic table of contents and outline will be generated.
MANUAL	Like REPORT, but with an index.
REPORT1	Like REPORT, but the sections will not be numbered.
ARTICLE	A sectioned document without chapters; SECTION is the highest-level of sectioning.
ARTICLE1	Like ARTICLE, but the sections will not be numbered.
POSTER	A single-page poster or announcement
SLIDE	An overhead-projector slide. Font and line spacing have been chosen to maximize readability.

Since SCRIBE is still under development, we expect that the set of available document types will be expanded in due time. The @MAKE command must be at the beginning of your manuscript file.

In the Expert's Manual that is a companion to the one you are now reading, you can find out how to define your own document types or modify the appearance of existing ones. We recommend that you not attempt this until you are an experienced SCRIBE user. In the meantime, you may use the @Style and @Font commands that are described in chapter STYLECHAPTER to get small changes in format and style and type face.

5. Titles, Headings, Sections, and the Table of Contents

Almost every document has a title. Some have a title page. Others have headings and subheadings here and there in the text. In this chapter, you will find out how to do various kinds of titles and headings.

As usual, we'll start with some vocabulary. A document has one and only one title. This title goes on the title page, at the top of the first page, or something like that. A document can have many headings. Sometimes you might want to put a few subheadings inside a heading. If you give numbers to headings and subheadings, then they become sections. At the top (and/or bottom) of every page are the page headings, sometimes known in the printing business as "running heads".

Different document types use different schemes to produce headings or sections. This manual, for example, was produced using the document type "MANUAL"; it is divided into numbered chapters, sections, and subsections and has a table of contents. A smaller document might have sections but not chapters; a short piece of text might have just a heading or two. The particular SCRIBE commands that you should use to get headings in your document depend on what kind of a document you are producing.

5.1. Headings in a Document Without a Table of Contents

There is a simple set of heading commands to produce headings for documents without tables of contents. They are simple because all they have to do is print the heading: they don't have to give it a number or put it in the table of contents. Use

- MajorHeading to get large letters, centered.
- Heading to get medium-size letters, centered.
- SubHeading to get normal-size boldface letters, flush to the left margin.

Thus, this input sequence:

```
@MajorHeading(This is a Major Heading)
@Heading(This is a Heading)
@Subheading(This is a Subheading)
```

will produce this output:

THIS IS A MAJOR HEADING

THIS IS A HEADING

This is a Subheading

That's all there is to it. Since these heading commands are really just environment names, you can use the "long form" call on them also:

```
@begin(MajorHeading)
This is a Major Heading
@End(MajorHeading)
@begin(Heading)
This is a Heading
@End(Heading)
@begin(SubHeading)
This is a Subheading
@End(SubHeading)
```

will produce the same results as did the previous example.

5.2. Headings in a Document Having a Table of Contents

A document having a table of contents must use a different set of commands for headings. They are different because they need both to print the headings in the text and to make an entry in the table of contents. Usually these commands will automatically number the chapters and sections, though in some document types they will not.

The use of these sectioning commands is as simple as can be:

```
@Chapter(Chapter Title)
```

declares (and prints) a chapter title;

```
@Section(Section Title)
```

declares (and prints) a section title.

In document types REPORT and MANUAL, you get the following sectioning commands:

```
Chapter, Section, SubSection, and Paragraph
Appendix and AppendixSection
UnNumbered
PrefaceSection
```

In document type ARTICLE, you get the same list except "Chapter"; the highest-order sectioning command in an ARTICLE is "Section".

The easiest way to show you what these commands do is to draw your attention to the table of contents at the front of this manual, and to reproduce here the first few sectioning commands that were used in it:

```
@PrefaceSection(Preface)
@Chapter(Introduction)
@section(Some Explanation for Non-Programmers)
@section(Some Explanation for Programmers)
@chapter(A Beginner's Guide)
@section(Preparing Input for Scribe)
@section(Running Scribe)
@Section(Printing Devices)
@Section(Characters and Keyboards)
@Subsection(Getting SCRIBE to Print Special Characters)
@SubSection(Faking Special Characters)
@Chap(Preparing a Manuscript)
```

Notice that Chapter, Appendix, UnNumbered, and PrefaceSection all cause SCRIBE to start at the top of a new page.

5.3. Numbering Pages and Page Headings

The information printed at the top or bottom of each page is called

Left heading center heading Right heading

a page heading (or page footing). Normally SCRIBE will number the pages for you, putting a page number centered at the top of all pages after the first. You may, if you like, change the page headings to any text that you want.

The page heading and footing areas are divided into three parts: a left part, a center part, and a right part. These parts are printed The headings and footings of this page are labeled to show you whe position of these six fields.

Page headings and footings are declared with the @Pageheading and @PageFooting commands. The commands for this page said:

```
@Pageheading(left "Left heading",center "center heading",
        right "Right heading")
@Pagefooting(left "Left footing",centert "center footing",
        right "Right footing")
```

The text fields inside the quotes can contain anything, including SCRIBE commands. For example, the @PageHeading command used to print most of this document (all but this page, in fact) was

```
@pageheading(left "@c[SCRIBE User's Manual]",
        right "@c[Page @ref(page)]")
```

The @c[...] generates small capitals (see section 3.3). The @ref(Page) causes the current page number to be printed in that spot (see section XREFS).

The @Pageheadinng and @Pagefooting commands may take an optional parameter, IMMEDIATE. If the IMMEDIATE parameter is present:

```
@PageHeading(Immediate,Left="Left heading",....)
```

then the heading of the current page will be changed. In the absence of the Immediate parameter, the newly-declared heading or footing

Left footing center footing Right footing

won't take effect until the following page.

5.4. Title Pages

The document types REPORT and MANUAL (and their variations) include a simple mechanism for generating title pages.

Surround the title page text with a @Begin(TitlePage) and a @End(TitlePage). These will cause the title page to be on a page by itself, and text within the title page to be centered.

At CMU, the title page for a technical report should have the title, author, and date of publication visible within a "box" that will be reproduced on the cover of the report. To put text in this box, use the environment "TitleBox". If you do a @Begin(TitleBox), the next line of text will be positioned at the very top of the title box region. When you @End(TitleBox), text assembly will shift down to a region of the page that is no longer in the title box. To get large or medium letters inside the title box, use the "MajorHeading" or "Heading" environments described in section 5.1.

The environment "CopyrightNotice" is used to put a copyright notice on the title page. If you type this text into the manuscript file:

```
@CopyrightNotice(L. Frank Baum)
```

then SCRIBE will put the following line on your title page, at an appropriate spot near the bottom:

Copyright -C- 1978 L. Frank Baum

for the current year.

The environment "ResearchCredit" is used to put a research funding

credit at the bottom of the title page. Any text that you put between a @Begin(ResearchCredit) and @End(ResearchCredit) will be placed in an appropriate spot at the bottom of the page, justified and single spaced.

See appendix TITLEPAGEEXAMPLE for an example of a title page.

6. Format Control: Tabs, Columns, and Cursor Movement

SCRIBE provides several ways to get text or numbers formatted into columns. For simple formats or small amounts of tabular material, the VERBATIM environment serves admirably. For more complex formatting, typewriter-style tab stops, set in fixed columns, are available; they may be used in any environment, but FORMAT is a frequent choice.

6.1. VERBATIM

For small amounts of tabular material, it might be easiest to format "by hand" at the terminal and use the VERBATIM environment. In VERBATIM, SCRIBE copies each line of the input manuscript file to one line in the output, and uses a fixed-width font so that characters will line up correctly.

For example, if the input manuscript file contains this text:

```
@Begin(Verbatim)
        Directory listing       26-Jun-78        0:56:57
Name Extension Len  Prot     Access      ---Creation---

ARTICL  MAK     25  <155>   24-Jun-78    11:59   22-Jun-78
TEXT    MAK      3  <055>   24-Jun-78     0:56   24-Jun-78
MCFREP  MAK      9  <055>   26-Jun-78    18:30   24-Jun-78
MANUAL  MAK     48  <055>   25-Jun-78    22:48   24-Jun-78
  Total of 85 blocks in 4 files on DSKC: [C410BR10]
@End(Verbatim)
```

then the finished document, when printed, will look like this:

```
        Directory listing       26-Jun-78        0:56:57
Name Extension Len  Prot     Access      ---Creation---

ARTICL  MAK     25  <155>   24-Jun-78    11:59   22-Jun-78
TEXT    MAK      3  <055>   24-Jun-78     0:56   24-Jun-78
MCFREP  MAK      9  <055>   26-Jun-78    18:30   24-Jun-78
MANUAL  MAK     48  <055>   25-Jun-78    22:48   24-Jun-78
  Total of 85 blocks in 4 files on DSKC: [C410BR10]
```

The fixed-width font is important so that columns will line up correctly: if we printed that same text in our normal font (which has

different widths for different characters), it would come out looking like this:

```
        Directory listing       26-Jun-78        0:56:57
Name Extension Len  Prot     Access      ---Creation---

ARTICL  MAK     25  <155>   24-Jun-78    11:59   22-Jun-78
TEXT    MAK      3  <055>   24-Jun-78     0:56   24-Jun-78
MCFREP  MAK      9  <055>   26-Jun-78    18:30   24-Jun-78
MANUAL  MAK     48  <055>   25-Jun-78    22:48   24-Jun-78
  Total of 85 blocks in 4 files on DSKC: [C410BR10]
```

6.2. Tabs and Tabs Stops

SCRIBE has a tab-stop mechanism that works pretty much the way tabs on a typewriter work. Tabs may be set in any horizontal position, and when a "tab" command is found, SCRIBE will move right to the next tab stop and start formatting there.

6.2.1. Setting Tab Stops

The setting of SCRIBE tabs is very simple. The @TabClear command will clear all tabs, and the @Tabs command will set them. Thus, the sequence

```
@Tabclear
@Tabs(1inch,2inches,3inches)
```

will clear all tabs and then set new tabs 1, 2 and 3 inches from the left margin of the current environment.

The @TabDivide command will divide the formatting area into a particular number of columns, making them however wide they must be in order to fit that many columns on the page. Thus,

```
@TabDivide(5)
```

will set 4 tab stops, each 1/5th of the way across the page. The right margin serves as the final tab stop.

The @. command sets a tab in the current cursor position. Thus, the sequence

```
@tabclear
This line @. is for setting @. tab stops.
```

will clear all tabs and then set two new tabs. The exact location of these new tabs depends on the font in use. The first one will be set at the distance that the letters "This line" occupy on the page.

6.2.2. Tabbing to a Tab Stop

The @\ command tells SCRIBE to move its formatting cursor to the right until it encounters the next tab stop. If there is no tab stop to the right of the cursor position, then it will be moved to the right margin.

The ^I or TAB character on your terminal's keyboard does not generate a SCRIBE tab, but rather is converted into a series of blank spaces.

6.3. Fancy Cursor Control

This section tells you how to do more elaborate cursor control. These fancy cursor-control features can be used, for example, to generate wierd mathematical formatting. If you don't see how these features would be useful to you, then they aren't.

6.3.1. The Return Marker

SCRIBE maintains a "return marker", which is a fixed memory of a particular horizontal position on the page. You set the return marker by putting "@!" in your text. Wherever "@!" occurs, the return marker will be set to the current horizontal cursor position.

Whenever SCRIBE finds an "@/", it moves the cursor to the return marker. This will work regardless of whether the cursor was to the left or right of the return marker.

One use of the return marker is building super-subscripts:

```
A@i@+[i]@/@-[j]
```

generates

$$A^{i}_{j}$$

6.3.2. Centering, Flush Left, and Flush Right

SCRIBE has commands that make it take pieces of text and center them or flush them right. Left flush is no big trick.

To flush a text fragment right, it must be flushed "against" something. Usually text is flushed to the right margin,

like this

but text can be flushed right against any tab stop:

```
                    text flushed
                           right
            against a tab stop
```

To flush a text fragment to the right, you have to delimit the fragment to be flushed and you have to tell SCRIBE what to flush it against. The code "@>" (looking vaguely like a right arrow) says "begin a flush-right operation". If there is a tab sequence (@\) later in the line, then SCRIBE will find the column that the tab sequence tabs to, and then flush right against that column position. If there is no tab sequence later in the line, then SCRIBE will flush the rest of the line against the right margin.

Time for an example. This input:

```
@tabdivide(3)
@>Flushed to right margin
@>to tab stop@\
@>longer line flushed to different tab stop@\
This line starts in the middle @>and flushes from here
```

produces this output:

```
                                                Flushed to right margin
        to tab stop
                        longer line flushed to different tab stop
This line starts in the middle                    and flushes from here
```

Before talking about the commands that center, let's talk about what we mean by centering. Centering requires that there be a left and right marker of some kind, and that the text be centered between them. For example,

this line is centered in the page

but

this is centered in the left half of the page

this is centered in the right half of the page

SCRIBE uses tab stops and margins as the reference points for centering. If there are no tab stops set, it will always use the left and right page margins as centering guides. If there are tab stops, then SCRIBE will use them instead. More detail in a minute.

To center something, you have to mark the left and right ends of the thing to be centered. Mark the left end always with "@=". You may mark the right end with "@\" (a tab), with another "@=", or with nothing. If you mark the right end with nothing, then SCRIBE will center the entire rest of the line and use the right margin for a right-hand centering guide.

When SCRIBE finds a @= (begin centering) code or a @> (begin flushright) code, it first checks to see if there was a previous unclosed @= or @> on the same line. If there was, then SCRIBE simulates a @\ code (tab), which closes the previous center or rightflush code. If this all sounds much too complicated, then contemplate the following example. This input:

```
@begin(format)
@TabDivide(4)
a@\b@\c@\d
@=e@\@=f@\@=g@\@=h
@=i@=j@=k@=l
Left@=Center@>right
Left@=Center@>right@\
@TabClear
Left@=Center@>right
@End(format)
```

produces this output:

```
a                   b                   c                   d
        e                   f                   g                   h
        i                   j                   k                   l
Left  Center                                                        right
Left  Center right
Left                                Center                          right
```

Notice that the "e f g h" line and the "i j k l" line produce the same result. Notice also that you may combine a @= centering code and a @> flush-right code on the same line, to produce the "Left Center right" line shown.

6.4. Controlling Word, Line, and Page Breaks

SCRIBE sometimes decides to break a line at a place that you don't like; sometimes it might break a page between two lines that really ought to be together on the same page. This section tells you how to get things kept together. It also tells you how to force word breaks, line breaks, and page breaks.

6.4.1. Controlling Word and Line Breaks

SCRIBE breaks lines between words. To change where it breaks a line, we must change what it thinks is a word. Words are separated by blanks, so what we need is a means of making SCRIBE think that a blank is part of a word rather than a separator between two of them. A blank that is part of a word is called a significant blank. The need for significant blanks arises often in mathematical formatting. If you are going to write x = y, you don't want the x on one line and the = y on another: you want to make the blanks into significant blanks.

There are two ways to do this. The sequence "@ " (an @ sign and then a space) generates a significant blank, which is treated by SCRIBE exactly as if it were a letter. Thus, you could write

```
x@ =@ y
```

to make a single word out of that equation. For long and complicated equations, it gets a bit tedious to put an "@ " instead of every blank; it's also quite hard to read. SCRIBE therefore provides a command that makes all blanks in its range into significant blanks: @w. So

```
@w[x = y]
```

produces the same results as our previous example.

The @| command is used to tell SCRIBE that it may break a word at that point. If your text contains "Abracadabra", for example, then the output document might contain "Abracadabra" printed together as a single word, or it might contain "Abra" on one line and "cadabra" on the next line, depending on how the line filling happened.

If you want to force a line break to occur before it otherwise might, use the "@*" command. When SCRIBE is justifying text and comes across an "@*", it will start a new line without justifying the old one. If SCRIBE encounters a "@*" while processing unjustified text, it will be treated as if it were an end-of-line.

6.4.2. Controlling Page Breaks

Normally SCRIBE will start a new page when the old one is full, regardless of what it is doing at the time. If SCRIBE happens to be processing an insert at the moment that the page becomes full, the insert will be split across two pages. It is often desirable to ensure that an insert will not be split across a page. There are two ways to proceed if an insert will not fit on a page: start a new page (leaving blank space at the bottom of the previous page), or "float" the insert to the top of a nearby page and continue the text uninterrupted at that point. Both of these options are available in SCRIBE. To ask for them, you need to use the long-form insert specification (with @Begin and @End).

To allow an insert to float if it won't fit on the current page, add the keyword FLOAT to the @Begin marker:

```
@Begin(Verse,Float)
Text of verse
@End(Verse)
```

If the verse insert won't fit, the text will continue without interruption, and the insert will appear on the top of the next page.

To require that a new page be started if an insert doesn't fit on the current page, use the keyword GROUP:

```
@Begin(Display,Group)
Text of display insert
@End(Display)
```

The @HINGE command may be put into a GROUPed insert to allow it to be broken at that point. If one or more @Hinge commands are in a GROUPed insert, it will be treated as a sequence of smaller inserts, each one being GROUPed. One way of viewing the @HINGE command is to view it as being equivalent to a @End followed by a @BEGIN. @HINGE will be ignored if it appears anywhere but inside a GROUPed insert.

If you would like to force a page break to occur prematurely, use the @NEWPAGE command. "@NEWPAGE" causes SCRIBE to start at the top of a new page. "@NEWPAGE(how many)" causes it to leave <how many> blank pages, then start at the top of a new page.

7. Cross Reference, Bibliography, and Citation

SCRIBE does automatic bookkeeping of cross references of various kinds. One kind of cross reference is the bibliographic citation, a reference to something mentioned in the bibliography. Another kind of cross reference is the text reference, which is a reference to some other part of the document.

7.1. Cross References

A cross reference is a notation like "see page 13" or "see chapter 5". If you are typing a document by hand, it is a major feat to get cross references right; and if you change anything, they are suddenly wrong again. The Joy of Cooking, an 800-page kitchen classic, averages 7 to 8 cross references per page. We shudder to think of the work that it took to get all 5000 to 6000 of them right.

SCRIBE will do cross referencing automatically, by letting you define codewords to mark places in the text, then filling in the correct page or section number wherever you reference the codeword. There are two separate kinds of cross references, but their use is nearly identical. You may refer to the page or section number containing a piece of text, or you may refer to a specific equation or theorem or example by number. These are called text references and object references respectively.

7.1.1. Text References: the @LABEL Command

The mechanism for text references is quite simple. If you put the sequence @LABEL(Codeword), for some codeword that you choose, anywhere in a document, SCRIBE will note the page and section number at the

spot where the @Label appeared. If you put @REF(Codeword) somewhere else, SCRIBE will fill in the section number of the text that contains the corresponding @Label; if you put @PAGEREF(Codeword), SCRIBE will fill in the page number of the corresponding label.

To take an example from this manual, we said on page 21 that

> ... each line in which a @TAG command (see section 7.1, page 42 appears).

The manuscript file text that generated that line actually contains:

> ... each line in which a @TAG command (see section @ref(Xrefs), page @pageref(Xrefs) appears).

Someplace in chapter 2 there is a @Label(Manuscripts) command. "Manuscripts" is a codeword that was picked more or less arbitrarily to stand for that chapter. You must choose your own codewords and spell them the same everywhere.

If a @REF at the beginning of a document references a @LABEL at the end, SCRIBE will use the page number that was generated the last time the document was processed.

If you have made changes to a document, some of the cross references might be wrong the first time you SCRIBE it after making those changes. If this happens, SCRIBE will tell you; you then have the option of printing the file as it stands (with the errors, but knowing that it's not your final copy and that you don't need perfection), or of re-running SCRIBE to get the cross references right. They will be right the second time because the correct values for all labels will have been stored in an auxiliary file, which SCRIBE will read in before the second processing. Since the normal evolutionary

development of a document involves alternate editing and reprocessing, the references will be correct almost all of the time, and if you want perfection on a particular run, it is simple enough to run it through twice.

SCRIBE builds an "AUX" file that contains the various pieces of information that it must remember from one run to the next. Cross-reference labels are included in this AUX file. If you are processing a file named, say, "RECIPE.MSS", then SCRIBE will store its auxiliary information in a file named "RECIPE.AUX".

7.1.2. Object References: the @TAG Command

One of the most valuable features of SCRIBE is its ability to number things for you, and thus to renumber if changes are made. Automatic page numbering is pretty much taken for granted; we have already mentioned the automatic numbering of sections (Section 5.2) and of equations or theorems (Section 3.4.2). SCRIBE will number anything it can count; you can also attach a cross-reference tag to anything that SCRIBE can count.

Let's talk about the difference between the @Label command and the @Tag command; this difference is a bit subtle. A sectioned document has many different sequences of numbers that identify the pieces of it. Pages are numbered sequentially; chapters and sections and such are usually numbered independently of the pages. Figures and tables are often numbered within a chapter, ignoring section numbers. If you want to attach a codeword to the section number of the section that contains a particular theorem or example or figure or table, then use

@Label. If you want to attach a codeword to the theorem number or example number or equation number, rather than to the section that it is contained in, use the @Tag command.

It's time for some examples. You should remember from section 3.4.2 that the THEOREM and EQUATION environments are used to print theorems and equations and automatically to assign numbers to them. You may place an @Tag command inside a THEOREM or an EQUATION to define a label so that you may reference the equation or theorem number. You may also place a @Label command inside a THEOREM or an EQUATION so that you may reference the section number or page number on which the theorem or equation occurs.

This unlikely input text:

```
@begin(Theorem)
@tag(imbed)
@label(ImbedSec)
If G is a non-self-embedding context-free grammar, then
L(G) is a regular set.
@End(Theorem)
It follows immediately from Theorem @ref(Imbed), that ...
   [and in a later section...]
We showed in section @ref(ImbedSec) in Theorem @ref(imbed) ...
```

might produce this output text:

> Theorem 4.10: If G is a non-self-embedding context-free grammar, then L(G) is a regular set.
>
> ...It follows immediately from Theorem 4.10, that ...
>
> We showed in section 4.6 in Theorem 4.10 ...

7.2. Bibliography and Citation

A bibliography is a labeled list of books, articles, papers, and the like. A citation is a marker in the text that refers to an entry in the bibliography. Thus, if a section of your bibliography looks like

this:

[15] E. W. Dijkstra. Co-operating Sequential Processes. In F. Genuys, editor, Programming Languages, pages 43-110. Academic Press, 1968.

[16] Mary-Claire van Leunen. A Handbook for Scholars. Alfred A. Knopf, 1978.

[17] A. van Wijngaarden, B. J. Mailloux, J. E. L. Peck, and C. H. A. Koster. Report on the Algorithmic Language ALGOL 68. Mathematisch Centrum, Amsterdam, 1969.

then text with citations might look like this:

> M. van Leunen, in her lively and detailed book, argues against the bibliographic footnote [16].
>
> The original language, which was rather more cumbersome to learn and to use, is defined in van Wijngaarden et al [17].

The SCRIBE bibliography facility does three things:

1. Selects from a database the bibliographic entries that are actually cited,
2. Formats them into a bibliography and assigns a number or label to each, and
3. Causes the correct numbers to be placed in the citations in the text.

7.2.1. Bibliography Files

Because SCRIBE selects and prints only those entries in a bibliography that are actually cited in the text, you may use a common bibliography "database" for many different documents. This database is in a separate file which we will call the bibliography file. Each entry will be given an identifier, or name, by which it is cited. These identifiers work in very much the same way as do the @label/@ref cross-reference keywords (see section 7.1): an identifier is defined in one place (the bibliography) amd cited in others (@cite commands in the text).

The bibliography file thus consists of a series of entries, each with an identifier. Each entry also has an entry type and a list of data fields. For example, the bibliography file entries for our examples of the previous section might be

```
@InBook(Processes, Author "E. W. Dijkstra",
                   Title "Co-operating Sequential Processes",
                   Editor "F. Genuys",
                   Title "Programming Languages",
                    Publisher "Academic Press",
        )
@Book(HANDBOOK,Author "Mary-Claire van Leunen",
               Title "A Handbook for Scholars",
               Publisher "Alfred A. Knopf",
               Date "1978"
        )
@Report(ALGOL68,Author "A. van Wijngaarden, B. J. Mailloux,
                        J. E. L. Peck, and C. H. A. Koster",
             Title "Report on the Algorithmic Language ALGOL 6
             Publisher "Mathematisch Centrum, Amsterdam",
             Date "1969"
        )
```

In this example, the names "Processes", "HANDBOOK", and "ALGOL68" are the identifiers by which these bibliography entries will be cited. The text of the document would contain @Cite commands that reference those identifiers.

Any text following a bibliography file entry is assumed to be an annotation or commentary on that entry, and will be included in the bibliography printed in your document.

7.2.2. Citations

Citations are simple. Just put a @cite command into your text at the spot where the citation should appear. @Cite takes one or more identifiers to tell it what to cite:

```
@cite(Knuth59)
@cite(Guarino78,Cooprider78)
```

These will be processed by SCRIBE into standard form citations:

[12]
[5,16]

7.2.3. Alternate Citation Styles

Some people don't like straight numeric citations, though it is the form most frequently needed in preparing papers for publication. SCRIBE tries to have something for everyone, though, and therefore provides a style parameter to change citation style. You may choose:

1. Straight numeric citation. This is normally the default: [6].
2. Author's last name and year: [Guarino78].
3. First 3 characters of author's last name, and year: [Gua78].
4. Print the identifier instead of a generated label.

You may set the bibliography style using the CITATION parameter to the @Style command:

@Style(CITATION=1) or @Style(CITATION=4), etc.

Some document types may include a specification of CITATION style of other than 1; this is why we said that style 1 is normally the default rather than style 1 is the default.

7.2.4. Nuts and Bolts of Bibliography Entry Format

As we said earlier in this chapter, a bibliography file is a sequence of bibliography entries, each possibly followed by an annotation. This section describes the detailed format of these entries.

Each bibliography entry takes the form

@what(identifier, information list)

where "what" is one of BOOK, ARTICLE, THESIS, INBOOK, REPORT, or MISC. The "information list" is a list of data about the entry, each datum consisting of a parameter word and a quoted string. Each type of entry needs a different set of information. For example, an ARTICLE needs a JOURNAL parameter, while a THESIS needs a SCHOOL parameter. The complete set of parameters is:

AUTHOR	The name of the author or authors. Put names in the order "First I. Last" and not "Last, First I.".
JOURNAL	The title of the journal or its standard abbreviation.
ISSUE	The number within volume of the journal issue.
VOLUME	For journals, the journal volume number. For multi-volume books, the book volume number.
DATE	The publication date.
TITLE	The title of the article, paper, book, or thesis being cited.
BOOKTITLE	If a chapter of a book is being cited, the book title must also be given.
EDITOR	If a chapter of a book is being cited, the name of the editor of the book mus be given.
PAGE	If desired, a page number or numbers may be provided.
SCHOOL	The name of the school at which a thesis was submitted.

If an unnecessary parameter is included in an entry, it will be ignored. If a necessary parameter is omitted from an entry, a warning message will be issued when the file is processed.

8. Figures and Tables

Figures and tables are inserted into the manuscript file in very much the same way as any other kind of insert: you mark the beginning with a @Begin or @Begin, and mark the end with an @End or @End. Between those delimiters you place the commands necessary to produce the figure.

A figure has three parts:

1. A figure body. The figure body can be something you paste onto the finished document. It can be an image picture printed on the XGP. It can be text produced with SCRIBE. How you produce it is your own business.

2. A caption. All figures must have captions. You provide the text of the caption, and you decide how it is to be placed with respect to the figure body.

3. A figure number. Figures are numbered sequentially. SCRIBE will automatically assign numbers to figures, and you can use the standard SCRIBE cross-reference mechanism (see section 7.1) to refer to them.

8.1. Generating Figure Bodies

You may generate the body of a figure using any of the standard SCRIBE environments, such as FORMAT, VERBATIM, or EXAMPLE. You may also generate the body of a figure using the special commands @Blankspace or @Picture. The @Blankspace command leaves a blank space in your document; the @Picture command (which works only on the XGP, and then only on the CMU XGP) inserts an image picture into the document [1].

Figure 8-1: Sample Image Picture

---

[1] Image pictures are binary files produced by the SPACS program, the PLOT program, and others. These programs are mentioned in the CMU Introductory Users' Manual and are fully documented in other manuals.

8.1.1. The @Blankspace Command

The @Blankspace command makes a blank space of a specified size. @Blankspace(3 inches) makes a blank space about 3 inches high; @Blankspace(16 cm) makes a blank space about 16 centimeters high. We say "about", because SCRIBE will always leave a small margin above and below the blank space that you request. You may use the distance units "inches", "in", "inch", "cm", "mm", or "lines". If you specify the amount of blank space in lines, then the amount that you actually get will depend on the font and printing device being used.

8.1.2. The @Picture Command

The @Picture command needs two arguments: the size of the image picture, measured in XGP scan lines, and the name of the PDP-10 file that will contain the image picture at the time you are printing the document on the XGP. A typical @Picture command might be:

```
@Picture(622,File "TEMP:IOBUS.IMG[F100CH00]")
```

This command will generate an image picture that is 622 scan lines high (about 3.4 inches), and whose definition can be found in the file "TEMP:IOBUS.IMG" on the account F100CH00[1].

8.2. Generating Figure Captions

Figure captions are generated with the @Caption command. The call is just

---

[1] CMU Computer Science Department users please note: the file name and PPN specified here are evaluated on CMU-10B, regardless of which machine you run SCRIBE on. If you have different account numbers on different machines, make sure that you use the 10-B account numbers in the @Picture command

```
@Caption(Text of Figure Caption)
```

SCRIBE will add the word "Figure" and the correct figure number; what will get printed in the finished document will be something like:

Figure 3-6: Text of Figure Caption

.

8.3. Figure Numbers

Figures are numbered automatically during the processing of the @Caption command. If you don't put a caption on a figure, it won't get a number. If you want to reference the figure in the text of the document, you will want to put a@Tag command into the body of the figure. The best place to put the @Tag is after the @Caption. If you put it before the @Caption, you will get the wrong figure number assigned to the tag, because the number will not yet have been incremented.

8.4. Full-page figures

If you want to make sure that a figure occupies an entire page, use the FLOATPAGE attribute to the @Begin(Figure):

```
@begin(Figure,FloatPage)
<figure body>
@Caption(figure caption)
@End(Figure)
```

The figure will be given a page to itself. If you want the caption at the bottom of the page, you must put in the right amount of spacing yourself.

If you want to leave a blank figure page in the document, use the @Blankpage command:

```
@Blankpage(1)
```

will leave one blank figure page. This figure page will have page headings and footings, and will be given a page number, but nothing will be printed on it.

You may remember that there is a @Newpage command (section 6.4.2), which starts a new page and can leave a specified number of blank pages. @Blankpage differs from @Newpage in a subtle but important way. When SCRIBE encounters a @Newpage command, it skips immediately to the top of the next page. When SCRIBE encounters a @Blankpage command, it makes a note about the blank page request but continues on the current page; then when it finally comes to the end of the current page, it will put in the extra blank page(s) at that point.

The sequence

```
@Begin(Figure,FloatPage)
@End(Figure)
```

will yield exactly the same effect as

```
@Blankpage(1)
```

.

9. Making Changes to Standard Formats

Although SCRIBE's basic approach to document production is to provide its users with a large menu of document types and discourage them from tinkering with details, we recognize that the urge to tinker is incurable. SCRIBE therefore provides two commands that may be used to make small changes in the format and style of a document.

9.1. The @Font Command

The @Font command tells SCRIBE to use a particular type font. To us, a type font is a collection of type faces, chosen to look harmonious when used together. A font will often contain 10 to 15 different type faces in diferent sizes and styles. For example, a typical font will contain Roman, Boldface and Italic in both body text size and footnote text size; it will also contain various sizes of heading and title fonts, usually without italics and boldface. A font, like a document type, is defined by an entry in SCRIBE's database. This font database entry is a list of information of the form "when I say @B, I mean Times Roman Bold; when I say @R, I mean Times Roman Medium." An entry like that would be found in a font whose name was "Times Roman", which would be specified by the command @Font(TimesRoman).

9.2. A List of Available Fonts

At CMU at the time of this writing (July 1978), the following fonts are available for use with SCRIBE. Many different character-set files exist for the XGP, but very few of them are attractive enough and readable enough to warrant printing a document in them. Each SCRIBE font uses 10 to 15 different character-set files.

Here is the list; a sample of each is in appendix FONTSAMPLES:

```
NewsGothic10
NewsGothic12
Nonie12
Nonie9
Meteor12
Meteor9
Bodoni10
```

The number associated with a font is its height in "points", which is a unit of printer's measure. A 10-point type like News Gothic 10 is about the same size as a standard Elite typewriter font.

For Diablo printers, a font simply specifies a type wheel and a list of special characters to be constructed with it; SCRIBE knows about these fonts (typewheels) for the Diablo:

```
Elite
Pica
```

Although users may create their own fonts by creating their own database entries, we defer description of this process to the Expert's Manual.

9.3. The @Style Command

The @Style command changes the settings of certain of SCRIBE's internal parameters. These parameters form a sort of "genetic code"; each parameter is somewhat like a gene. There is a parameter that controls whether or not SCRIBE makes a flush right margin. There is a parameter that controls how much space SCRIBE leaves between lines; another that controls the distance that each paragraph is indented.

The database entry for a document type presets the value of all SCRIBE parameters. You may use the @Style command to override these values. All @Style commands used in a document must come at the

beginning; you can't change styles in mid-document.

Let's consider a small example. The document type TEXT specifies no paragraph indentation; it adds an extra blank line between paragraphs instead. If you wanted to change to standard typewriter-style indentation (5 spaces), you could put the command

```
@Style(Indentation,5)
```

into your manuscript file. SCRIBE would then indent paragraphs 5 spaces. If you also wanted to suppress the extra blank line between paragraphs, you could say

```
@Style(Indentation 5, Spread 0)
```

or alternatively you could say

```
@Style(Indentation 5)
@Style(Spread 0)
```

The word "Spread" was chosen to mean inter-paragraph spacing solely because it would be pretty tedious to type @style(Inter-ParagraphSpacing 0).

Some style parameters expect names instead of numbers; others expect quoted strings. For example:

```
@Style(Justification Off)
```

will turn off right-margin justification, while

```
@Style(Footnotes "*")
```

tells SCRIBE to use asterisks instead of numbers when counting footnotes.

9.4. A List of STYLE Parameters

This table is a partial listing of SCRIBE's various style parameters

and an explanation of what kind of value they expect. The complete list, which contains many obscure and exotic parameters, is in the Expert's Manual. These parameters are used in the @Style command, which is explained in the previous section.

Indentation — Controls paragraph indentation. A horizontal distance should be provided:

```
@Style(Indentation 5)
```

means to indent 5 spaces. If you want backwards indentation, give it a negative number.

Spacing — Controls inter-line spacing. A vertical distance should be provided: "2" means "2 lines", i.e. double space. If you want, say, one-and-a-half spacing, then say

```
@Style(Spacing 1.5)
```

The default spacing that you get depends on the document type being produced.

Spread — Controls the inter-paragraph spacing, i.e. the extra space, over and above the line spacing, that will be put between paragraphs.

Justification — A "Yes", "No", "On", or "Off" tells SCRIBE whether or not to justify the right margin. Don't put the word in quotes:

```
@Style(Justification Off)
@Style(Justification On)
```

LeftMargin — This parameter will change the size of the left margin of the pages. For example,

```
@Style(LeftMargin 1.3 inches)
```

will cause the page to have a 1.3-inch left margin.

RightMargin — This parameter will change the size of the right margin. The RightMargin parameter and the PaperWidth parameter both affect the length of the printed lines.

TopMargin — Changes the top margin of the page.

BottomMargin — Changes the bottom margin of the page. The BottomMargin parameter and the PaperLength parameter both affect the size of the printing area on the page.

PaperLength — Used to specify the length of the paper that you are using. For example,

```
@Style(PaperLength 14 inches)
```

specifies legal-size paper.

PaperWidth — Used to specify the width of the paper that you are using.

Footnotes — Controls the way that footnotes are numbered. The normal numbering scheme is numeric: the first footnote is numbered "1", the second one "2", and so forth.

```
@Style(Footnotes "*")
```

will cause the numbering to be "**", "***", and so forth. If you want some other numbering scheme used, specify a template. See the Expert's Manual to find out what that means.

10. Producing Large Documents

SCRIBE has several features that make it easier to work on large documents. The @File command allows a document to be split into multiple files, which makes editing and storage easier. The @String command allows you to store frequently-used text strings and reference them by name. The @Part command facilitates separate compilation of parts of a larger document. The @Use command tells SCRIBE to use a particular set of database files or a particular bibliography or .AUX file.

10.1. Including Subfiles in a Manuscript File

SCRIBE normally reads sequentially through a manuscript file, processing text and commands as it comes to them. The @File command makes SCRIBE suspend processing of the main manuscript file, process a second file, then resume processing of the original manuscript file. For example, if your manuscript file contains

```
00100 The Wicked Witch was both surprised and worried when
00200 she saw the mark on Dorothy's forehead, for she knew
00300 well that neither the Winged Monkey nor she, herself,
00400 dare hurt the girl in any way.
00500 @File(SUBFIL.MSSIC410BR103)
00600 At first the With was tempted to run away from Dorothy;
00700 but she happened to look into the child's eyes and saw
00800 how simple the soul behind them was, and that the
00900 little girl did not know of the wonderful power the
01000 Silver Shoes gave her.
```

and if the file whose name is SUBFIL.MSSIC410BR103 contains

```
00100 She looked down at Dorothy's feet, and seeing the
00200 Silver Shoes, began to tremble with fear, for she knew
00300 what a powerful charm belonged to them.
```

then the finished document will contain the text

> The Wicked Witch was both surprised and worried when she saw the mark on Dorothy's forehead, for she knew well that neither the Winged Monkey nor she, herself, dare hurt the girl in any way. She looked down at Dorothy's feet, and seeing the Silver Shoes, began to tremble with fear, for she knew what a

> powerful charm belonged to them. At first the With was tempted to run away from Dorothy; but she happened to look into the child's eyes and saw how simple the soul behind them was, and that the little girl did not know of the wonderful power the Silver Shoes gave her.

@File commands may be nested, i.e. a file that is processed because of a @File command may itself contain other @File commands naming yet more files.

10.2. Defining and Using Text Strings

The @String command provides a mechanism for storing a series of characters under a particular name, then including that series of characters in the document by referring to the name in a @Ref command.

10.2.1. The @String Command

@String defines names to equal a quoted string:

```
@String(CACM="Communications of the ACM")
                2              2
@String(EQ4="x +4*x*y-17*y ")
@String(Name="J. Alfred Prufrock", Agency="Rutabaga, Inc.")
```

You need not use quotation marks; any of the delimiters described in section 3.1 will work. To use these text strings that you have defined, call them inside a @Ref command. For example, if the @String commands of the previous example have been processed, then this input:

```
Prior work was published in @ref(CACM).
The solution is @ref(EQ4).
@Ref(Name) no longer is employed by @ref(Agency).
```

will produce this output:

```
Prior work was published in Communications of the ACM.
                    2              2
The solution is x +4*x*y-17*y
J. Alfred Prufrock no longer is employed by Rutabaga, Inc.
```

If both a text string and a cross-reference label are given the same name, then the @Ref will refer to the text string and not to the

cross-reference label.

10.3. The @Part Command: Separate Compilation

10.4. The @Use Command: Changing the Processing Environment

I. A Potpourri of Examples

This appendix is full of random examples. All of them were motivated because early users of SCRIBE often asked, "How do I do such-and-so". There is no particular logic to the sequencing of these examples.

@AppendixSec(How to Make a Title Page) Document Types REPORT, MANUAL, and ARTICLE contain a title page facility; you can also say and produce just the title page.

This input file produced the title page for this manual:

```
@begin(TitlePage)
@begin(TitleBox)
@MajorHeading(SCRIBE

Introductory
User's
Manual)
@Heading(First Edition)

Brian K. Reid
@ref(Date)

@End(TitleBox)
@begin(Heading)
Carnegie-Mellon University
Computer Science Department
@End(Heading)
@CopyrightNotice(Brian K. Reid)
@Begin(ResearchCredit)
The research that produced SCRIBE was funded in part by the
Rome Air Development Center under Contract No. F306-2-75-C-021
and in part by the Defense Advanced Research Projects Agency
under contract No. F44620-73-C-0074.
@End(ResearchCredit)
@End(TitlePage)
```

The "TitlePage" environment will be on a page by itself; i.e. a page break will be taken before it and after it. The "TitleBox" environment is used to line up text with the box on CMU technical report covers.

Anything that you put inside the environment "TitleBox" will be positioned on the page so that it will show through the cutout box.

The "ResearchCredit" environment puts justified text at the bottom of the title page. The "CopyrightNotice" environment generates the centered notice, and positions it in a particular place on the page. @AppendixSec(Equations and Cross References) This example is taken from Volume 1 of The Art of Computer Programming by Donald E. Knuth. It shows the use of the SCRIBE equation numbering scheme, and it shows the use of cross-references to equations earlier in the text.

Here is the input file:

```
Let @i[P(n)] be some statement about the integer @i[n]; for
example, @i[P(n)] might be "@i[n] times (@i[n] + 3) is an even
number," or "if @i[n] ?> 10, then 2@+[@i[n]] > @i[n]@+[3]."
Suppose we want @i[to prove that P(n) is true for all positive
integers n]. An important way to do this is:
@Begin(Enumerate)
Give a proof that @i[P(1)] is true; @Tag(BaseStep)
Give a proof that "if all of @i[P](1), @i[P](2), . . . ,
        @i[P](@i[n])
are true, then @i[P](@i[n]+1) is also true"; this proof should
        be valid
for any positive integer @i[n]. @Tag(InductStep)
@End(Enumerate)
As an example, consider the following series of equations, which
many people have discovered independently since ancient times:
@equation(
1  = 1@+[2], 1 + 3 = 2@+[2], 1 + 3 + 5 = 3@+[2], 1 + 3 + 5 + 7 =
        4@+[2],
1 + 3 + 5 + 7 + 9 = 5@+[2].
)
We can formulate the general property as follows:
@equation{
1 + 3 + . . . + (2@i[n] - 1) = @i[n]@+[2] @Tag(Induction)
}
Let us, for the moment, call this equation @i[P(n)]; we wish to
prove that @i[P(n)] is true for all positive @i[n]. Following
the procedure outlined above, we have:
@Begin(Enumerate)
"@i[P](1) is true since 1 = 1@+[2]."
"If all of @i[P](1), . . . , @i[P](@i[n]) are true, then, in
        particular,
@i[P](@i[n]) is true, so Eq. @Ref(Induction) holds; adding
```

```
            2@i[n] + 1
to both sides, we obtain
@begin(equation)
1  +  3  +  .  .  . + (2@i[n] - 1) + (2@i[n] + 1) = @i[n]@+[2] +
            2@i[n] + 1 = (@i[n] + 1)@+[2]
@End(equation)
which proves that @i[P](@i[n] + 1) is also true."
@End(Enumerate)
We can regard this method as an @i[algorithmic proof procedure].
In  fact,  the  following  algorithm  produces  a  proof  of
            @i[P](@i[n]) for
any positive integer @i[n], assuming that steps @Ref(BaseStep)
and @Ref(InductStep) above have been worked out.
```

and here is the output that it produces:

Let P(n) be some statement about the integer n; for example, P(n) might be "n times (n + 3) is an even number," or "if n ?> 10, then 2^n > n^3." Suppose we want to prove that P(n) is true for all positive integers n. An important way to do this is:

1. Give a proof that P(1) is true;

2. Give a proof that "if all of P(1), P(2), . . . , P(n) are true, then P(n+1) is also true"; this proof should be valid for any positive integer n.

As an example, consider the following series of equations, which many people have discovered independently since ancient times:

$$1 = 1^2,\ 1 + 3 = 2^2,\ 1 + 3 + 5 = 3^2,\ 1 + 3 + 5 + 7 = 4^2,$$

$$1 + 3 + 5 + 7 + 9 = 5^2.$$

We can formulate the general property as follows:

$$1 + 3 + \ldots + (2n - 1) = n^2 \qquad (1)$$

Let us, for the moment, call this equation P(n); we wish to prove that P(n) is true for all positive n. Following the procedure outlined above, we have:

1. "P(1) is true since $1 = 1^2$."

2. "If all of P(1), . . . , P(n) are true, then, in particular,

P(n) is true, so Eq. 1 holds; adding 2n + 1 to both sides, we obtain

$$1 + 3 + \ldots + (2n - 1) + (2n + 1) = n^2 + 2n + 1 =$$

which proves that P(n + 1) is also true."

We can regard this method as an algorithmic proof procedure. In fact, the following algorithm produces a proof of P(n) for any positive integer n, assuming that steps 1. and 2. above have been worked out.

II. Special Characters

III. Font Samples

This section contains samples of some of the XGP fonts, to show you what they look like. Remember that exotic and intricate fonts, however showy they might be, are very hard to read unless you have only a very few words on the exotic font.

Index

---

1 like this one