AD-A064 808 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 5/2
SCRIBE INTRODUCTORY USER'S MANUAL.(U)
AUG 78 B K REID F44620-73-C-0074
UNCLASSIFIED AFOSR-TR-79-0045 NL

1 OF 2
AD
A064808

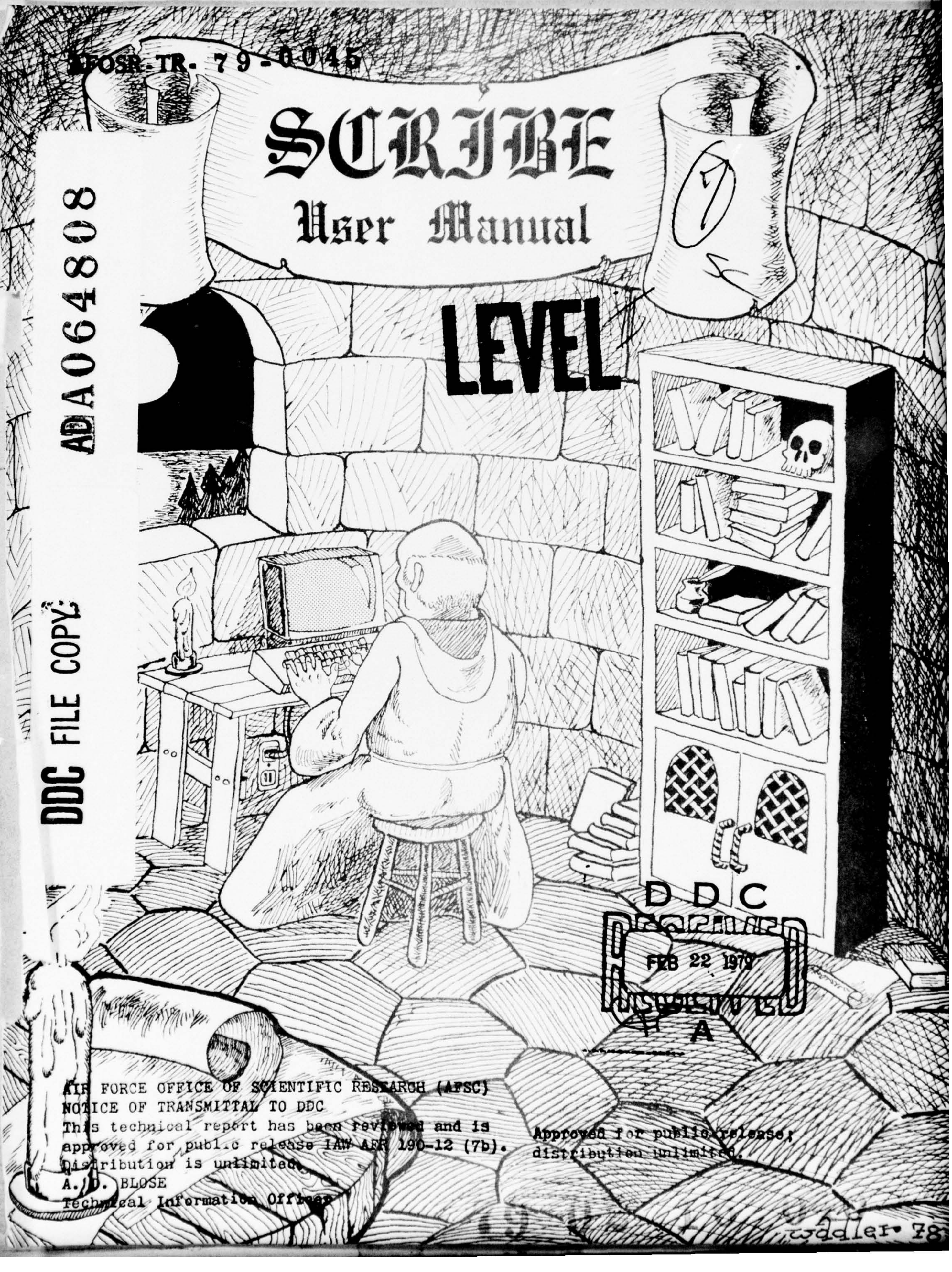
AFOSR-TR- 79-0045

SCRIBE User Manual

LEVEL

ADA064808

DDC FILE COPY



DDC
RECEIVED
FEB 22 1979
A

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is
approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

Approved for public release;
distribution unlimited.

20. Abstract continued.

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 particular set of formatting commands that have been devised and documented are designed around the sorts of documents that computer scientist produce. Users in non-academic or non-methematical environments will have very little use for the "Theorem" or "Equation" features, but may find something else desirable. In time there will be separate editions of the manual.

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

ACCESSION for
NTIS White Section [✓]
DDC Buff Section □
UNANNOUNCED □
JUSTIFICATION

BY
DISTRIBUTION/AVAILABILITY CODES

Dist.	AVAIL. and/or SPECIAL
A	

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (18) AFOSR-TR- 79-0045	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Introductory SCRIBE USER'S MANUAL	5. TYPE OF REPORT & PERIOD COVERED (9) Interim rept.,	
	6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s) (10) Brian K. Reid	8. CONTRACT OR GRANT NUMBER(s) (15) F44620-73-C-0074, DAAG29-77-C-0034	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Department of Computer Science Pittsburgh, PA 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101E A02~~46677~~ (16) ~~17~~ 66	
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wison Blvd. Arlington, VA 22209	12. REPORT DATE (11) 3 Aug[redacted]978	
	13. NUMBER OF PAGES 116	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Office of Scientific Research/NM Bolling AFB, Washington, DC (12) 119p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

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

The reader is expected to have used 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. (Continued on reverse side)

DD FORM 1 JAN 73 1473

403 081

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LEVEL II

AD A064808

SCRIBE

Introductory User's Manual

First Edition

Brian K. Reid
3 August 1978

DDC FILE COPY

Carnegie-Mellon University
Computer Science Department

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

This manual corresponds to SCRIBE version CMU 0E(110).

DDC
RECEIVED
FEB 22 1979
A

Copyright -C- 1978 Brian K. Reid

The research that produced SCRIBE was funded in part by the Rome Air Development Center under Contract No. F30602-75-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.

79 02 16 040

Table of Contents

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 Experts' 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 particular set of formatting commands that we have devised and documented are designed around the sorts of documents that computer scientists produce. Users in non-academic or non-mathematical environments will have very little use for the "Theorem" or "Equation" features, but may find something else desirable. In time there will be separate editions of the manual.

This manual was produced with SCRIBE and printed on the Computer Science Department's Xerox Graphics Printer. If you find anything wrong with it, please let me know. My ARPANET mail address is "Reid@CMU-10A"; my regular snailmail address is "Department of Computer Science, Carnegie-Mellon University, Pittsburgh PA 15213".

1. Introduction

SCRIBE is a document production program. It produces "documents", designed to be read by people, from "manuscript files", which are primarily to be read by the computer. The manuscript files that SCRIBE processes are created by people by 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 Experts' 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 needs 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

PRECEDING PAGE BLANK-NOT FILMED

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

The Scribe system was designed to make document production easy for the non-expert, and to allow ~~them~~ him to make small changes to the formats and styles without needing to learn much about how the program works. Scribe is not a programming language.

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 an "@End(quotation)" after its text. The details of the knowledge of how to print a QUOTATION are encoded in Scribe's internal tables.

The non-procedural nature of Scribe typically gives fits to people who have grown accustomed to procedural document formatters. Programmers are used to thinking in terms of procedural commands, and are used to having the full power of the document formatter available at the command level. Many expert programmers, upon first seeing Scribe, seem to think that it is weak and powerless and unflexible, when what they really mean is that it doesn't conform to their notion of what a procedural command language should do.

Scribe has been in use at Carnegie-Mellon since January 1978. We have found that even hard-core hackers can learn how to use Scribe effectively, but that they must learn to think differently before they will ever be comfortable with it. People who aren't experts at older document systems have had no difficulty learning to use Scribe.

2. *Getting Started*

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

The blank line (700) indicates the end of one paragraph and the beginning of another. To process this file, we run SCRIBE; it will prompt us for input by typing an asterisk:

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

PRECEDING PAGE BLANK-NOT FILMED

```
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[A800DP99][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 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

[1]Only at CMU Computer Science Department sites.

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)	Prepare for a line printer
@Device(XGP)	Prepare document for Xerox Graphics Printer
@DEVICE(DIABLO)	Prepare for Diablo HyType terminal
@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:

```
@device(XGP)
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 Key, 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.'
```

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.

A table of the DEVICE codes known to Scribe is shown in figure 2-1. We expect to be able to add a device type for a photocomposing machine as soon as one becomes available to us.

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

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
LA36	66 lines	132 cols	LA36 DECwriter
TI700	50 lines	80 cols	Texas Instruments 700
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

Figure 2-1: Device Types Known To SCRIBE

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 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[@u[boldfaced underlining]]?
```

This is called *nesting* of the delimiters. The output that it will produce will look like

[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.

this:

Why would anybody want <u>boldfaced</u> <u>underlining</u>?

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<@u[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 do what you want: it will print as "*2*(4+y*-b)", which is wrong.

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 font-change codes listed in figure 3-1. 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.

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, or what have you. 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 the

@i[phrase]	*Italics*
@u[phrase]	Underline non-blank characters
@ux[phrase]	Underline all characters
@un[phrase]	Underline alpha-numerics (but not punctuation or spaces)
@b[phrase]	**Boldface**
@r[phrase]	Roman (the normal typeface)
@t[phrase]	`Typewriter font`
@+[phrase]	print $^{\text{super}}$script
@c[phrase]	SMALL CAPITALS
@-[phrase]	print $_{\text{sub}}$script
@g[phrase]	Greek (Ελλεν)
@o[phrase]	Overbar
@p[phrase]	***Bold Italics***

Figure 3-1: Font-change Codes

right delimiter that closes the quotation.

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 the three forms shown above.

A quotation in running text comes out single spaced, with the left and right margins widened 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 abqut 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 inf figure 3-2; examples of the use of each can be seen in Appendix I.

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 lines. QUOTATION doesn't pay any attention to ends of lines in the manuscript file; it puts as many characters on each line as will fit there. VERSE starts a new line in the final document for each end-of-line in the manuscript file, regardless of whether the existing line is full. If VERSE needs to use more than one line of the document to hold the text from one line of the manuscript, the second and following lines 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[2].

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

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

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

Figure 3-2: Basic Insert Types

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 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 5.

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 will all be processed, so it's not completely verbatim.

FORMAT 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 7), 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 with slightly wider 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 of the page, 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 the sequence "@\" or a ↑I (TAB) character is found on the first line of a DESCRIPTION, it will be taken as a signal to tab to the indented margin provided that it has not yet been passed. Thus, this input:

```
@Begin(description)
Segment↑IOne of the parts into which something naturally
separates or is divided; a division, portion,
or section.

Section↑I          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 8.1, page 49) 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
$x_{n+1} = (x_n + c/x_n)$ (3-12)

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 8.1 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 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

Some document types request that an index be generated at the end of the document. This index does not magically become full of entries; rather, you have to tell SCRIBE what to put in the index. It's really quite simple. You use the @Index command, and provide the text of the index entry. SCRIBE will fill in the correct page number and see to it that the entries are in alphabetical order.

At any point in the text, you may put an index 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"

[1] Like this one

will appear in the index with the correct page number:

Text to be Indexed 22

@Index works like @Foot in that it doesn't interrupt the sequence of the text. For example, the input

```
Place the peaches and apples@index(peaches)@index(apples)
in a glass jug and pour in the wine.@index(wine)  A few
teaspoons of sugar may be added if a sweeter drink
is preferred, though it is not normally done.
```

will produce this output:

> Place the peaches and apples in a glass jug and pour in the wine. A few teaspoons of sugar may be added if a sweeter drink is preferred, though it is not normally done.

You may consult the index of this manual to see how those three entries look.

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. As usual, if you want an @ sign to appear in the index, you have to use two:

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

The index entries will be sorted into alphabetical order according to the standard "Ascii" code. If you've put an "@" sign at the front of an index entry (as we have in the example of the last paragraph), then it will be sorted as if "@" were the first letter. A facility exists whereby you can make things appear in the index out of normal alphabetical order; you will have to consult the *Experts' Manual* to learn how to use it.

3.7 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 Π or Γ, or a mathematical symbol like $\leq$ or $\equiv$. These special characters are not available on the keyboard as individual keys. If you are using an output device that is capable of printing special characters, you can get SCRIBE to print them for you. If you are not using such an output device, you can get SCRIBE to leave space for you to write in the characters by hand.

3.7.1 Getting SCRIBE to Print Special Characters

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 ?< B then @g(L) must be 0
```

then your finished output will contain

if A $\leq$ B then Λ must be 0

The @g(L) to print a lambda is a "font-change" code, which changes to the Greek font. Font-change codes are described in section 3.3; page 76 has a table of the Greek letters that you can get with the @g command. The "?<" is a special-character escape sequence to generate the less-or-equal character. A table of the special-character codes is on page 73.

3.7.2 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 "β" character:

Kaiser Wilhelmstraβe

If 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.

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 or commands that aren't "setup" commands. The five "setup" commands are @Make, @Device, @Style, @Font, and @Part. If you use any of these five commands, all of them must come before any of the other commands.

SCRIBE knows about the document types that are listed in figure 4-1, which is to say that it is the list of names that you can put in a @Make command. Since SCRIBE is still under development, we expect that the set of available document types will be expanded in due time.

REPORT	A document with a title page, 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. This manual was produced using document type MANUAL.
ARTICLE	A sectioned document like REPORT, but without chapters; SECTION is the highest-level of sectioning.
LETTERHEAD	A business letter with CMU letterhead.
LETTER	A personal letter; you provide the return address
POSTER	A single-page poster or announcement
SLIDE	An overhead-projector slide. Font and line spacing have been chosen to maximize readability.

Figure 4-1: SCRIBE Document Types

In the *Experts' 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 one of the variant document styles described in the next section, or you may use the @Style and @Font commands that are described in chapter 9 to get small changes in format and style and type face.

4.3 Style Variations

SCRIBE's standard document type for producing user manuals is called "MANUAL". This SCRIBE manual was produced in that style. Different people like different styles, though, and one particular "Manual" style isn't going to please everybody. SCRIBE therefore has a mechanism for changing the style of formatting without changing the basic document type.

This manual was produced in form 1 of the document type "Manual". The @Make command that was used to produce it was

```
@Make(Manual,Form 1)
```

If it had been instead

```
@Make(Manual,Form 2)
```

then the form-2 "Manual" format would have been used instead; if the @Make command had been just

```
@Make(Manual)
```

then the normal "Manual" format would have been produced. These "forms", or style variations, have been produced by various SCRIBE users who didn't like the standard style. Form-1 "Manual" format uses different type fonts, while form-2 "Manual" uses wider margins and wider line spacing.

Each time a new installation starts using SCRIBE, they find that they don't like certain things about its formats. The new users typically make small style changes to the document-type definitions in the SCRIBE data base. As those small changes filter back to CMU, they get installed in our master version of the data base as variant styles of our basic document types.

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

Almost every document has a title. Some have title pages. Others have headings and subheadings here and there in the text. In this chapter, you will find out how to get 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 headers".

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.

PRECEDING PAGE BLANK-NOT FILMED

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 insert 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 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 must both print the headings in the text and 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". **UnNumbered uses the same style and font size as Chapter, but it is not assigned a chapter number.**

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(Getting Started)
    @section(Preparing Input for Scribe)
    @section(Running Scribe)
    @Section(Printing Devices)
    @Subsection(Getting SCRIBE to Print Special Characters)
    @SubSection(Faking Special Characters)
    @Chapter(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 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 at the left, center, and right sides of the heading or footing area.
The headings and footings of this page are labeled to show you the 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",center "center footing",
        right "Right footing")
```

The text fields inside the quotes can contain anything, including SCRIBE commands. To get the current page number, use "@Value(Page)". To get the date, use "@Value(Date)". Thus, this command:

```
@PageHeading(left "Reference Manual",center "@value(Date)",
                                       ,right "@value(page)")
```

will cause the pages to have "Reference Manual" in the top left corner, the current date in the top center, and the page number in the top right corner. Section 10.4.1 will tell you more about the @Value command.

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 won't take effect until the following page.

You may declare different page headings to be used on odd and even pages. If you are going to be reproducing your document printed on both sides of the paper, then you will want odd and even headings to be the mirror image of one another. Use two different @PageHeading commands, one containing the keyword "odd" and the other containing the keyword "even". The commands that generated the page headings for this manual are:

```
@pageheading(even,right "@c[SCRIBE Users' Manual]",
                  left "@c[Page @ref(page)]")
@pageheading(odd,left "@c[@Title(Chapter)]",
                 right "@c[Page @ref(page)]")
```

The @Title command is described in section 10.4.2.

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 insert type "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" commands described in section 5.1.

The insert type "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 insert type "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 I.1 for an example of a title page.

6. *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(Figure) or @Enter(Figure), and mark the end with an @End(Figure) or @Leave(Figure). 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 8.1) to refer to them.

6.1 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].

6.1.1 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

[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.

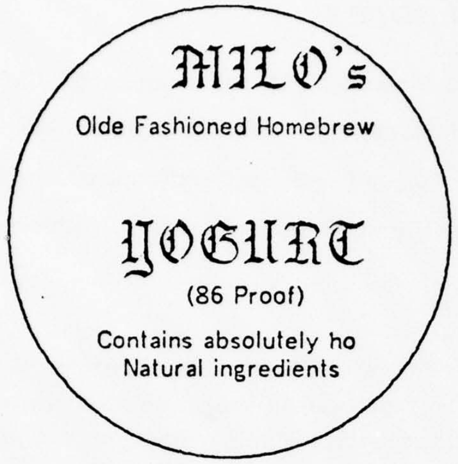


Figure 6-1: Sample Image Picture

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.

6.1.2 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].

6.2 Generating Figure Captions

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

```
@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

6.3 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 right 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.

6.4 Full-page figures

If you want to make sure that a figure occupies an entire page, use "FullPageFigure" instead of "Figure":

```
@begin(FullPageFigure)
<figure body>
@Caption(figure caption)
@End(FullPageFigure)
```

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.*

[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

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 7.4.2), 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(FullPageFigure)
@End(FullPageFigure)
```

will yield exactly the same effect as

```
@Blankpage(1)
```

7. 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.

7.1 VERBATIM

For small amounts of tabular material, it might be easiest to format "by hand" at the terminal and use a VERBATIM insert. 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]
```

7.2 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.

7.2.1 Setting Tab Stops

It's very simple to clear and set Scribe tabs. 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 document will show the lines

```
This line is for setting tab stops.
         ↑               ↑
```

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.

7.2.2 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.

7.3 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.

7.3.1 Overprinting

The @ovp command allows you to overprint things. @ovp[text] lays down "text" and then backspaces over it. Thus

```
Simple example of @ovp(=)/ overprinting.
```

will generate the output

Simple example of ≠ overprinting.

If you use the @ovp command, your document will most likely not give satisfactory results on a printing device other than the one you are used to, because the appearance of overstruck letters varies markedly from one printing device to another. If you want to overprint an underline character you would be better served to use the @U command, because different output devices use different characters for underlining.

You can use @ovp to fabricate an approximation to some special characters. The sequence "@ovp(⊂)/" will generate "⊄", which is a rough approximation to the mathematical symbol for "is not a subset of". If you are going to be producing a lot of special characters on the XGP, you would be better off making a private character set file that contains those special characters; see Section 11.2.3.

If you are so motivated, you can create real junk with the @ovp command:

```
    This line is @ovp(Carefully)overstruck
```

produces:

    This line is Ovaeefullyck

7.3.2 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]@/@-[j]
```

generates

A^{i}_{j}

7.3.3 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.

It's time for an example. This input:

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

produces this output:

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

Since there is no @\ in line 1 after the @> code, the remainder of the line is flushed against the right margin. Line 2 has a @\ tab code at the end, so the line is flushed right against the tab stop to which that @\ tabs. Line 3 is like line 2, except that by the time the @\ code is reached, the line has already passed the first tab stop, so the @\ moves to the second tab stop and the line is flushed to that second tab stop. Line 4 shows a fragment of a line being flushed right.

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              this is centered in
         the left half of                 the right half of
             the page                         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 current left and right 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 "@=", with a "@>", or with the end of the line. If you mark the right end with nothing, then SCRIBE will center the remainder 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)
1.a@\b@\c@\d
2.@=e@\@=f@\@=g@\@=h
3.@=i@=j@=k@=l
4.Left@=Center@>right
5.Left@=Center@>right@\
@TabClear
6.Left@=Center@>right
@End(format)
```

produces this output:

```
1.a                  b                   c                   d
2.        e                   f                   g                   h
3.        i                   j                   k                   l
4.Left  Center                                                    right
5.Left  Center   right
6.Left                                Center                      right
```

In line 1, the "b", "c", and "d" are positioned immediately after the three tab stops set by @TabDivide(4), so they will serve as markers to help you see the position of those tab stops. Line 2 has four letters, one centered in each of the four columns. Each of them is in a centering field marked on the left with a "@=" code and on the right with a "@\" (tab) code. Line 3 produces the same effect without the "@\" codes; the "@=" code serves like a "@\" then the "@\" is missing.

Compare lines 4, 5, and 6. In line 4, a centering zone is begun by a "@="; the next code on the line is the "@>". Since there is no "@\" to close the centered zone, the "@>" first simulates a "@\", thereby tabbing to the first tab column, and centering the "Center" word between the left margin and the first tab stop. There is no "@\" code to end the "@>" flush-right code, so the rest of the line (i.e. the word "right") is flushed to the right margin. Line 5 is identical to line 4 except that there is a "@\" code at the end, which causes "right" to be flushed to the appropriate tab stop and not to the right margin. Line 6 is identical to line 4, but the tab stops have been cleared by a @TabClear command. This makes the "Center" right marker be the right margin, since there is not a tab stop for it to settle on.

7.4 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.

7.4.1 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 "Abra@|cadabra", 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. "@|" is stronger than @w -- if you use "@|" inside a @w group, the "@|" will win, and a word break will be allowed there.

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.

7.4.2 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. @Hinge will be ignored if it appears anywhere but inside a GROUPed insert. @Hinge always forces a new output line.

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. These pages will not be totally blank; rather they will have page headings and/or footings. If you would like completely blank pages, just put blank paper in your document at that point.

8. *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.

8.1 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.

8.1.1 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 20 that

> ... each line in which a @TAG command (see section 8.1, page 49)

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 section 8.1 there is a @Label(Xrefs) command. "Xrefs" is a codeword that was picked more or less arbitrarily to stand for that section. You must choose your own codewords and spell them the same everywhere in the document.

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 the document has never been processed before, then there won't be any "last time", and SCRIBE will tell you to process the file a second time if you want the cross references right.

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 auxillary 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".

8.1.2 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 them 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 ...

8.2 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.

8.2.1 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 8.1): 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(Dijkstra69, Author "E. W. Dijkstra",
                   Title "Co-operating Sequential Processes",
                   Editor "F. Genuys",
                   Title "Programming Languages",
                    Publisher "Academic Press",
                   Date "1969"
        )
@Book(VanLeunen78,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 68",
               Publisher "Mathematisch Centrum, Amsterdam",
               Date "1969"
        )
```

In this example, the names "Dijkstra69", "VanLeunen78", 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.

Normally SCRIBE will look for a file with an extension of ".BIB" as the bibliography file for a document; i.e. if your manuscript file is "THESIS.MSS", SCRIBE will use "THESIS.BIB" as the bibliography file. You may use the @Use command, described in section 10.5, to request that SCRIBE use some different file for the bibliography file for your document.

8.2.2 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]
```

8.2.3 Alternate Citation Styles

Some people don't like straight numeric citations, though it is the form most frequently needed in preparing papers for publication. Some journals require other forms, 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.

9. *Changing Things*

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 Changing Type Font: The @Font Command

The @Font command tells SCRIBE to use a particular type font. For use with SCRIBE, a "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 (August 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 II:

PRECEDING PAGE BLANK-NOT FILMED

News Gothic 10
News Gothic 12
Nonie 12
Nonie 9
Meteor 12
Meteor 9
Baskerville 12
Bodoni 10

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 *Experts' Manual*. If you have a private character set for the XGP that you would like to use, see section 11.2.3.

The HELP text for SCRIBE, which you can see by typing HELP SCRIBE to the monitor, will give you instructions for finding out the current list of available fonts.

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 enumerating footnotes.

9.4 A List of STYLE Parameters

This table lists the style parameters that you can specify in the @Style command, and briefly describes what they do.

Indentation

Controls paragraph indentation in text. 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. The "margin" of a page is the white space at the edge of the paper, so if you specify a bigger number here, it will make a bigger white space and hence leave a smaller print line.

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. Remember that the "margin" is the white space at the edge of the paper, so increasing the size of the RightMargin parameter will increase the size of the margin, hence decreasing the amount of space taken by the print line.

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 *Experts' Manual* to find out what that means.

9.5 Miscellany

This section documents a few odds and ends that don't seem to belong anywhere else. They probably don't belong here either, but they've got to go somewhere.

9.5.1 Changing Command Names

If you really don't like SCRIBE's name for some command, you may use the @Equate command to define your own equivalent. The format is

```
@Equate(New=Old,New=Old,...)
```

Let's suppose, for example, that you don't like the name "Enumerate". You think it's too long. You would like to be able to say "@En" instead of "@Enumerate". Then enter at the front of your file the command

```
@Equate(EN=Enumerate)
```

From then on, you will be able to say

```
    @begin(en)
    @end(en)
```

or even just

```
    @en{...}
```

You may not use @Equate to change the meaning of an existing name. If you try, say,

```
    @Equate(Itemize=Enumerate)
```

SCRIBE will complain mightily and refuse to execute the @Equate.

9.5.2 Changing Counter Values

SCRIBE counters are things like "page", "chapter", and "TheoremCounter". They are incremented from time to time from inside SCRIBE, with values ranging from 1 to whatever.

Sometimes people don't like the counter numbers to be so well-behaved. SCRIBE therefore provides a command to let you meddle with counter values. For example, the command

```
    @set(Page=8,Chapter=4)
```

will set the page number to 8 and the chapter number to 4. If you would like to add or subtract some quantity to or from a counter, use a + or - sign: the command

```
    @Set(Page=+3)
```

adds 3 to the current page number.

We would like to refer you to the separate compilation facility, described in section 10.2, as a better way of doing the deeds that you are probably trying to accomplish when you go about meddling with counter values.

10. Producing Large Documents

Big documents are proportionately much harder to produce than small ones. The amount of work needed to produce one 200-page document is several times that needed to produce ten 20-page documents. Bookkeeping becomes a chore. The amount of file space used becomes huge. It can take an hour or two of elapsed time just to process a 200-page document through SCRIBE when the system is heavily loaded.

SCRIBE has a large-document facility, with several independent mechanisms, that make it easier to produce large documents. The large-document facility includes:

- An **@include** command, that allows a document to be split into multiple files, each one of which is of manageable size.
- A **@part** command that lets you separately process one component file of a document, yet have page numbers, section numbers, chapter numbers, and cross references still come out right.
- A **@string** command that lets you store commonly-used text sequences and recall them by name when they are to appear in the document.
- A **@use** command to request that SCRIBE use some private or custom edition of its database.
- An automatically-generated *outline* of your document, created in a separate file, to help you manage its organization.

10.1 Breaking a Manuscript into Several Smaller Files

SCRIBE normally reads sequentially through a manuscript file, processing text and commands as it comes to them. The @Include 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 @Include(SUBFIL.MSS[C410BR10])
```

```
00600 At first the Witch 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.MSS[C410BR10]* 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 Witch 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.

@Include requests may be nested, i.e. a file that is processed because of an @Include command may itself contain other @Include commands naming yet more files. The name "@File" is a synonym for "@Include", and may be used interchangeably with it.

To break up a large document into component manuscript files, it is best to build a small "root" file that contains the @Make, @Device, and @Style commands and then has a series of @Include commands that include all of the text. Each of these @Include commands includes one part of the document. It is usually convenient to divide a document into separate chapter files, with one @Include for each chapter, but any division that is convenient for you is acceptable.

Figure 10-1 shows the root file for the CMU *Computer Primer*, a large document produced with SCRIBE. It prints the title page, then includes in turn each of the chapter files.

The file CHO.MSS contains the preface, introduction, and such. CH1.MSS contains the text of chapter 1, CH2.MSS contains the text of chapter 2, and so forth. The

```
@device(XGP)
@make(Primer)
@pageheading(left "@c[CMU Computer Primer]",Right "@value(page)")
@begin(TitlePage)
@begin(TitleBox)
@begin(MajorHeading)
COMPUTER
PRIMER
@end(MajorHeading)
@begin(Heading)
For Computer Science Department
Computer Users

@end(Heading)
by
Brian K. Reid
@value(Month) @value(Year)

@end(TitleBox)
@begin(Heading)
Carnegie-Mellon University
Computer Science Department
@end(Heading)
@CopyrightNotice(Brian K. Reid)
@end(TitlePage)
@include(ch0.mss)
@Include(ch1.mss)
@Include(ch2.mss)
@Include(ch3.mss)
@Include(ch4.mss)
@Include(ch5.mss)
@Include(ap1.mss)
@Include(ap2.mss)
```

Figure 10-1: Sample Root File

"@Chapter" command is in the included file and not in the root file.

10.2 Separate Processing of Component Files

If you have used the @Include command to break your manuscript into small component files, you may process them independently as manuscripts in their own right. By judicious use of the @Part command, which we will shortly describe, you may separately process the components of a large document for draft copy purposes, yet be able to process the whole thing without having to change any of the components.

10.2.1 Component Files As Separate Documents

It frequently arises that some masterful piece of prose will be published as a chapter of a larger work as well as be published on its own. By carefully keeping its head in the sand at the right times, SCRIBE lets you set up files so that they will work both as included parts of a large document and as documents in their own right.

SCRIBE ignores certain commands if they are found inside an included file, but processes the same commands if they are found in a root file. This lets you put commands into a file that will be ignored if it is part of a larger document but will be processed if it is being produced as a document in its own right.

The commands **@Device**, **@Make**, **@Style**, and **@Font** will be ignored if they are found in an included file but processed if they are found at the beginning of the root file. The text inside a TitlePage environment (see section 5.4) will also be ignored inside an included file.

You may therefore put at the beginning of a file the commands necessary to process it as a stand-alone document, trusting that they will be ignored if it is included as part of a larger document.

10.2.2 Separate Processing of Component Parts

For very large documents, the processing time for the full document is large enough for it to be wasteful to re-process and re-print all of the document when only a piece of it has changed. A person working on a small section of a large document would like to be able to produce a draft copy of that section without coping with the whole thing. When each chapter is produced by a different author, the several authors would each like to be able to get proof copies of their chapter apart from the other text. For situations like these, SCRIBE provides a @Part command, which supports separate compilation of fragments of a larger document.

To take advantage of this separate compilation facility, place a @Part command at the front of each included file. It should be the first command in the file. The @Part command must provide two items of information: a name for the part and the file name of the root file of the document. Thus

```
@Part(Chapter3,ro[illegible] "primer.mss")
```

defines a part whose name is "Chapter3" and that is part of the document whose root file is PRIMER.MSS. The name that you choose for the part is just a name, and may be any combination of letters, digits, and "-" characters.

Having put @Part commands, each with a different part name but all specifying the correct root file, into the various subfiles, you must process the entire document once to record the part information. When Scribe finds a @Part command in an included file, it makes a note of the part name and the page and section numbers at that point. These notes are saved along with the cross-reference data in the AUX file for the document.

Once there exists an AUX file with part information stored in it, you may process any part by itself. If Scribe finds a @Part command at the beginning of a file it is asked to process, it assumes that you are separately compiling a piece of a larger document and enters a special sub-compilation mode. Scribe does three special things in sub-compilation mode:

1. Scribe finds the root file and processes the @Device, @Make, @Style, @Font, @Equate, @PageHeading, and @PageFooting commands that are in it. Some other commands, documented only in the *Experts' Manual*, will also be picked up from the root file.

2. Scribe restores the page and section numbers to the correct values for the part being processed.

3. Scribe reads in the cross-reference label definitions for the entire document, and not just for the part being processed.

The result of this complicated series of acts is that the separately-compiled part will be produced with the correct page and section numbers; if it contains any cross references to labels defined in some other part, the references will be correct.

At the end of processing a separately-compiled part, SCRIBE will update and rewrite the AUX file. If the number of pages in the part has changed since the last processing, SCRIBE will correct the stored page number information for parts that follow this one, in order that page numbers will be right if other parts are later processed.

The index and table of contents produced by processing a part will reflect only the contents of that part. If a new part is to be added, the entire root file must be processed to incorporate the new part correctly.

10.3 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.

10.3.1 The @String Command

@String defines names to equal a quoted string:

```
@String(CACM="@i[Communications of the ACM]")
@String(EQ4="@i[x]@+[2]+4*@i[x]*@i[y]-17*@i[y]@+[2]")
@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.*
The solution is $x^2+4*x*y-17*y^2$
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.

10.4 Environment Inquiry: Predefined Strings

SCRIBE has a mechanism that allows you to include in your document various characteristics of the processing environment, such as the date and the name of the file being processed. These things are called "internal strings" because they are defined inside SCRIBE.

10.4.1 Internal Strings and the @Value Command

The @Value command is used to access internal strings. For example, @Value(Date) will produce the current date at that point in the document, and @Value(Manuscript) will produce the name of the manuscript file. Thus, if you put in your manuscript file the text

```
    This document was produced on @Value(Date).
```

then your document will contain the text:

This document was produced on 3 August 1978.

@Value(Date) will produce the current date. The precise format in which the date will be printed is controlled by the "DATE" parameter to the @Style command. Normally the date will print this way: "3 August 1978". However, if you have put a @Style command at the front of your manuscript file that specifies a DATE style, it might come out in a different format.

@Style(Date=1) is the default. @Style(Date=2) gives you "August 3, 1978". @Style(Date=3) gives you "3 Aug 78". @Style(Date=4) gives you " 3 AUG 78". @Style(Date=5) gives you "08/03/78".

@Value(Month) will return the full name of the current month, e.g. "August". **@Value(Day)** will give you the day number in the current month, e.g. "3". **@value(Year)** will return the current year, e.g. "1978". **@Value(Weekday)** will give you the name of the current day of the week, e.g. "Thursday". **@Value(Time)** will tell you the time, in 24-hour notation, to the nearest minute, when the current SCRIBE run began; e.g. "03:54". To get a "time stamp" that records the date and time of processing, use **@Value(TimeStamp)**; it will give you a value like "3 August 1978 at 03:54".

You can also inquire about the files that SCRIBE is working on. **@Value(Manuscript)** tells you the name of the manuscript file that SCRIBE is processing, e.g. "USER.MSS". **@Value(SourceFile)** tells you the particular place in the manuscript file or the @Included subfile that SCRIBE is processing at that moment, e.g. "UMCH10.MSS, 05800/5. And if you really must know, **@value(ScribeVersion)** tells you the version of SCRIBE that is doing the processing, e.g. "CMU 0E(110)".

If you are producing a document with a table of contents, then you may inquire as to the number or title of the current section. **@Value(SectionTitle)** gives you the title of the chapter, section, subsection, or whatever is appropriate, e.g. "Internal Strings and the @Value Command". **@Value(SectionNumber)** gives you its number, e.g. "10.4.1". **@Value(Page)** tells you the current page number, e.g. "70".

10.4.2 Current Values of Counters

If the @Ref command is applied to a counter, e.g. PAGE or CHAPTER, it will give the current numeric reference value of that counter. If the @Title command is applied to a counter, it will give the current title associated with that counter. Thus, at this point in the SCRIBE manual, "@Title(Chapter)" returns "Producing Large Documents" and "@ref(Chapter)" returns "10".

There is a bit of ambiguity between @Value(Page) and @Ref(Page): they both do exactly the same thing. Early versions of SCRIBE did not have the @Value command, so @Ref(Page) is left in the system for compatibility.

If you ask for the title of a counter that has no title, a null string will be returned: "@Title(Page)" returns "".

10.5 File Names and the @Use Command

The "@Use" command directs SCRIBE to use other database, bibliography, and auxiliary files than it normally would. For example, SCRIBE will normally look for an AUX file with the same name as the manuscript file: if your manuscript file is "PRIMER.MSS", the AUX file would be "PRIMER.AUX". If you put the command

```
@Use(AuxFile "OTHER.AUX")
```

at the front of your manuscript, then SCRIBE will use "OTHER.AUX" instead of "PRIMER.AUX". *The extension ".AUX" will always be used.*

To request that SCRIBE use some particular bibliography file, use the @Use command with the parameter "Bibliography":

```
@Use(Bibliography "PROSYS.BIB[C410HB00]")
```

The extension ".BIB" is not required, any extension is allowed.

The @Use command may also be used to direct SCRIBE to use a nonstandard version of the database. If you put the command

@Use(Database "*somewhere*")

at the very beginning of your root file, then SCRIBE will interpret "somewhere" as the location of a database and will look there for everything that it needs. The precise nature of the "somewhere" string depends on the details of the computer system in use; sample strings for various popular computer systems are "<Jones>", "[M100MDOC]", and "[51,1062]".

Warning

If you develop your own version of the SCRIBE database, the software maintainer disavows any responsibility to maintain compatibility between your private database and future versions of the program.

11. Character Codes and Type Fonts

One of the least satisfactory aspects of computer document production is the difficulty of specifying and printing special characters. The ASCII character code, the official U. S. standard, has 95 printing characters in it; there are several thousand different characters used in the printing industry. 95 of those many thousand are standardized and available; all of the rest must somehow be fudged.

Because the availability of special characters is so dependent on the printing device being used, we discuss separately the availability of special characters for each kind of printing device.

11.1 Keyboards and Characters

There are 128 different characters that may be stored in a computer file. Some of those characters are "carriage control" characters: the tab, carriage return, line feed, back space, and such. Your keyboard has 95 printing keys on it. That leaves 26 characters that it is possible to store in a computer file but that it is not possible to type on your keyboard.

The SOS editor, which is the one you most likely use at CMU, has a scheme that allows you to type in those 26 extra characters. When you type a "?" to SOS, it combines that "?" with the character that follows it and takes the two characters together to mean one. If you really mean "?", you have to type two of them: "??". Table 11-1 shows the 26 special-character codes that you can get by combining the question mark with various other characters.

11.2 Font and Character Considerations for the CMU XGP

The Carnegie-Mellon Computer Science Department's XGP prints characters from special "character set" files. A character set file is a file with 121 printing characters in it. If it is a "standard" character set, then those 121 characters are the 95 ASCII and the 26 Stanford characters. If it is a "non-standard" character set, then it may have some other characters in it instead of the standard ones, say

Type This	To Get This	
?!	↓	Down-arrow character
?6	→	Right-arrow character
?2	↔	Left-right two-way arrow
?"	α	Lower-case Greek alpha
?#	β	Lower-case Greek beta
?&	ϵ	Lower-case Greek epsilon
?'	π	Lower-case Greek pi
?(	λ	Lower-case Greek lambda
?$	$\wedge$	Logical "and" symbol
?8	$\vee$	Logical "or" character
?%	$\neg$	Logical "not" symbol
?7	$\equiv$	Logical "equivalence" character
?/	$\forall$	Logical "for any" symbol
?0	$\exists$	Logical "there exists" symbol
?+	$\subset$	Set theory "subset of" symbol
?,	$\supset$	Set theory "superset of" symbol
?-	$\cap$	Set theory "intersection" symbol
?.	$\cup$	Set theory "union" symbol
?1	$\otimes$	Cross inside circle
?=	$\neq$	Not-equal character
?<	$\leq$	Less-than-or-equal character
?>	$\geq$	Greater-than-or-equal character
?)	∞	Mathematical "infinity" symbol
?*	∂	Mathematical "partial derivative" symbol
?9	_	Underline character for XGP
?4	~	Tilde character[1]
??	?	Question mark

Figure 11-1: "Question-Mark Code" Character Correspondence

[1]This character is the same as the tilde character on your keyboard. They are duplicated for reasons that are lost to history.

Greek letters or mathematical symbols.

11.2.1 Standard Fonts

For use with SCRIBE, the various character sets have been organized into families, which we call "Fonts". A SCRIBE font is a set of a dozen or so character sets, which have been chosen to look attractive when used together. All of the information that defines a font is stored away in the SCRIBE database by name. If you ask for the News Gothic font by putting the command @Font(NewsGothic10) at the front of your manuscript file, then SCRIBE will load the News Gothic 10 font definition from its data base. This font definition contains a set of instructions of the form "to get italic, use character set X" or "to get boldface, use character set Y".

When you type @i[x], what SCRIBE actually does is to switch to a character set known to contain italics, and print a lower-case "x". When you type @g[x], SCRIBE switches to a character set known to contain Greek, and prints a lower-case "x".

When you type "?7", which is the SOS code for the "≡" character, SCRIBE does not perform any character-set switching; it simply puts out the Stanford-ASCII code for the "≡" character. But some character sets don't contain that character. If you ask for @g[?7], for example, SCRIBE will print an error message telling you that the Greek character set does not include the "?7" character.

11.2.2 Special Fonts

If you are going to use a non-standard character set, you need to know what characters are in what positions; this information is called the correspondence. The Greek fonts, for example, have a correspondence wherein "a" corresponds to "α", "b" corresponds to "β", and so forth. Knowing that "a" corresponds to "α", then you would type "@g[a]" to get an "α" character, and so on.

Of the many special character sets that are available for the XGP, the two that are most likely to be useful are the Greek and mathematical character sets. The character correspondence for printing Greek characters is shown in Table 11-2; the character correspondence for printing mathematical characters is shown in Table 11-3.

We should probably explain why the mathematical character set contains the characters that it does. CMU's XGP is limited by design to only two different fonts at a time. It would be nice to be able to print an equation that contained Greek letters, a large $\prod$ character, various letters and numbers, and small-sized letters and numbers for super- and sub-scripts. Alas, since we only get two fonts to work with, the characters in the mathematical font were chosen to be representative of what everybody would need most. There are some mathematical symbols, some Greek letters, the digits 0 through 9 in a small size, and some lower-case letters in a small size. You will surely find that some character that you need is missing from the mathematical font, but it's better than nothing.

To get characters in the Greek set, use @g. To get characters in the mathematical set, use @z.

11.2.3 Non-Standard and User-Defined Fonts

One of the joys of working with the XGP is the ability to make your own character sets. If you absolutely must have an "$\mathbb{R}$" character in your text, you can use the various support programs for the XGP to make your own character set that contains an "$\mathbb{R}$". Before you do this, however, you ought to check to make sure that it hasn't already been done; at CMU we have about 200 different character set files for the XGP that have been created through the years.

In any event, let's suppose that you have managed to make, find, or steal a character set that contains the characters that you need. This character set will have a file name, and the exact nature of the file name will depend on the particular computer that you are using. Let's suppose that your character set file is named "CYR30.KST[C410BR10]", and that it contains Russian characters.

Scribe has ten "Special Fonts" that are set aside to be defined by the user.

ASCII	GREEK	Ascii	Greek	Name
A	Α	a	α	Alpha
B	Β	b	β	Beta
G	Γ	g	γ	Gamma
D	Δ	d	δ	Delta
E	Ε	e	ϵ	Epsilon
Z	Ζ	z	ζ	Zeta
H	Η	h	η	Eta
Q	Θ	q	θ	Theta
		j	ϑ	Theta (script)
I	Ι	i	ι	Iota
K	Κ	k	κ	Kappa
L	Λ	l	λ	Lambda
M	Μ	m	μ	Mu
N	Ν	n	ν	Nu
X	Ξ	x	ξ	Xi
O	Ο	o	ο	Omicron
P	Π	p	π	Pi
R	Ρ	r	ρ	Rho
S	Σ	s	σ	Sigma
		v	ς	Sigma (alternate)
T	Τ	t	τ	Tau
U	ϒ	u	υ	Upsilon
F	Φ	f	ϕ	Phi
		J	φ	Phi (script)
C	Χ	c	χ	Chi
Y	Ψ	y	ψ	Psi
W	Ω	w	ω	Omega

Figure 11-2: Greek Characters Available with @G

Code	Character		Code	Character	
?!	≪	Much less than	[	[	Left bracket
?"	≫	Much greater than	]	]	Right bracket
?<	≤	Less or equal	{	{	Left brace
?>	≥	Greater or equal	}	}	Right brace
?=	≠	Not equal	(	(	Left parenthesis
'	∝	Proportional to	)	)	Right parenthesis
B	≜	Defined to equal	/	/	Small solidus
?&	∈	Is member of	:	/	Large solidus
?#	∉	Is not a member	N	ℕ	Naturals
?.	∪	Union	Z	ℤ	Integers
?-	∩	Intersection	Q	ℚ	Rationals
?,	⊕	Exclusive or	R	ℝ	Reals
?8	∨	Conjunction	C	ℂ	Complex
?$	∧	Disjunction	#	×	Multiply
?%	¬	Negation	"	X	Large multiply
?7	≡	Equivalence	D	∆	Del
@	⇒	Implication	?*	∂	Partial
?	⇐	Definition	P	∏	Product
_	←	Assignment	S	∑	Sum
?0	∃	Existential quantifier	$	⌠	Top of Integral
?/	∀	Universal quantifier	&	⌡	Bottom of Integral
E	⊢	Derives	↑	↑	Diverges; Loops
%	⊨	Theorem	\	↓	Converges; Halts
!	‖	Spectral norm	?1	⇀	Vector
|	|	Euclidian norm	?2	−	Fraction bar
T	⌈	Left ceiling	?(	□	Box
U	⌉	Right ceiling	?'	§	Article/item
V	⌊	Left floor	`	′	Prime
X	⌋	Right floor	;	▯	Box

Figure 11-3: Mathematical Characters Available With @Z

Code	Character		Code	Character	
a	α	alpha	L	Λ	Lambda
b	β	beta	F	Φ	Phi
g	γ	gamma	Y	Ψ	Psi
d	δ	delta	W	Ω	Omega
e	ϵ	epsilon	A	ℵ	Aleph
z	ζ	zeta	O	𝔄	German U
H	η	eta	u	𝔑	German N
q	θ	theta	0	0	small 0
K	κ	kappa	1	1	small 1
l	λ	lambda	2	2	small 2
M	μ	mu	3	3	small 3
v	ν	nu	4	4	small 4
z	ζ	xi	5	5	small 5
p	π	pi	6	6	small 6
r	ρ	rho	7	7	small 7
s	σ	sigma	8	8	small 8
t	τ	tau	9	9	small 9
f	ϕ	phi	h	h	small h
J	φ	script phi	l	l	small l
c	χ	chi	j	j	small j
y	ψ	psi	k	k	small k
w	ω	omega	m	m	small m
G	Γ	Gamma	n	n	small n
I	I	Iota	o	o	small o

Figure 11-4: Other Characters Available in the @Z Font

They are numbered 0 through 9, and you get at them by using the codes @F0 through @F9. The @SpecialFont command is used to declare your intent to use a special font and to give it a number, and the @F0 through @F9 codes are used to print it.

The command

```
    @SpecialFont(F3="CYR30.KST[C410BR10]")
```

defines special font 3 to be your Russian font. If you would also like to define, say, a Hebrew font at the same time, you can put

```
    @specialfont(f3="CYR30.KST[C410BR10]",f4="HBRW30.KST[C410BR10]")
```

and so forth. You must choose the numbers yourself. Having put this command at the front of your manuscript file, you may now use the code **@f3** for Russian and the code **@f4** for Hebrew; the input

```
    This line prints @f3(Russian Letters)
```

will produce

This line prints Руссиан Леттерс

and the line

```
    This line prints @f4(SRETTEL WERBEH)
```

will produce

This line prints העברעת לעטטערש

Because of limitations of the CMU XGP, you cannot use more than two fonts on a line, or for that matter you cannot use more than 3 fonts in any two lines, so we can't demonstrate a line that has all 3 of Roman, Russian, and Hebrew on it.

11.3 Considerations for Other Printing Devices

If you use one of the 26 Stanford ASCII characters in a manuscript file, Scribe will make a valiant effort to print it, even on printing devices that are limited to the normal 95 characters. On an LA36 DECwriter, for example, Scribe will overstrike a

"V" with a "-" to get the "∀" character. This same trick works on a Texas *Instruments terminal*, but it won't work on a Diablo, because the "V" is shaped differently. But the Diablo terminal has a graphics mode that allows the construction of special characters, so SCRIBE will try something else there. If you are producing output for device "FILE" or "EDITOR" or "SOS", then SCRIBE will just leave the special character in the file and trust that your editor or file-printing program knows what to do with the special character.

There is a general mechanism in SCRIBE to allow the fabrication of *special* characters by overprinting and the like, but you'll have to *read the Experts' Manual* to find out how to use it. If you use *one of the 26 Stanford special characters*, we'll print it for you without *your having to learn anything*; if you want some other character *besides the ones* in that set, you need to learn how to use the special-character *mechanism.*

11.4 *A Note About The ASCII Character Set*

A *file* stored on a computer is usually a sequence of characters in a code called ASCII. The ASCII character set (American Standard Code for Information Interchange, pronounced ass'-key) was approved in 1968 as the official code for the <u>interchange</u> of information among computers. Our DEC computers also use it to <u>store</u> information in files. This dual use of the code for purposes that are very *similar causes some confusing situations* at times; it will help you be a better SCRIBE user if you understand them.

The ASCII character code has 128 distinct characters in it. These are divided *into* 94 printing characters, 1 space, and 33 "control" characters. When ASCII was first invented, people had grand visions of what those 33 "control" characters would be used for, but it has never come to pass. They have exotic names like "End of Transmission" and "Unit Separator"; on modern computers they lie essentially unused.

Through the years, different people have tried to use the ASCII control characters for different purposes. On DEC computers, the "End of Text" character, or control-C, has been used to mean "stop the execution of this program". On IBM computers, the "End of Transmission" character, or control-D, has been used to mean "kill this job and hang up the telephone"; on some Control Data computers the "End of Transmission" character is used to mean "I am done with this line of input"; on DEC computers we use a carriage return to mean that.

A number of years ago, some people at Stanford selected 26 special characters and assigned them to slots in the ASCII code that are supposed to be slots for control functions. The particular set of 26 that they chose was motivated by their need to print mathematical and logical expressions. That set of 26 special characters has come to be known as "Stanford ASCII". We at CMU use the Stanford ASCII character code because our XGP uses it; the XGP uses the Stanford code because the people who built the XGP system were largely Stanford graduates. In any event, we are stuck with Stanford ASCII, so we might as well learn how to use it effectively.

Meanwhile, the people at DEC who build the computers and write the operating systems for them had taken that same set of ASCII control characters and assigned various control meanings to them. For example, the "Device Control 4" code, which you can generate by typing control-T, is used by DEC as a "probe" command to find out what the computer is doing with your program.

Both the Stanford and DEC uses of the ASCII control characters are in violation of the USA Standard Code, but no Federal Marshal is likely to come running out and arrest people who type control-T to their computers. These misuses are going to stay with us, so you may as well learn how to use them. As you might expect, when a standard is violated there are problems with compatibility among the various groups who have violated it. The people who violate the standards like to tell you that they have "extended" the standards, just as terrorists will often tell you that they are "freedom fighters". It's all a matter of vocabulary.

12. Epilogue and Sermon

The guiding principle that shaped every aspect of the design of SCRIBE is that most people who produce documents don't know or care about the details of the formatting involved. To this end, those details are determined by information in SCRIBE's database and not by commands from the user.

The benefits of this somewhat arrogant approach are legion. For those people who really *don't* care about details, it is possible to produce attractive formatted output with very little work and even less decision-making. To produce a document for a different printing device, the simple change of one @Device command will yield a completely different set of formats, designed to look attractive on the new device.

The cost of this approach is that it is not always possible to get every character to land on the page in precisely the spot that you think it ought to. Trying to force SCRIBE to format one way while it is trying to format in another way is like pulling teeth: possible but painful.

Please don't use pliers on SCRIBE. If you find yourself fighting with SCRIBE, you are probably not using it right. If you catch yourself constantly thumbing through the table of commands and style parameters, trying to find the command or parameter that will force some particular character to a particular column, then you are not using SCRIBE correctly. And if you ever catch yourself, even once, wanting to use the @Blankspace command outside a figure, then you are definitely using SCRIBE wrong.

If you find yourself fighting SCRIBE, think carefully about what that means: SCRIBE is trying to produce a document in one format, while you want it to be in another. You have several recourses:

1. Learn to like the standard SCRIBE formats. This is by far the simplest approach. Not everyone needs to be a typographer.

2. Look at all of the variant styles that are in the SCRIBE database. Perhaps one of them will please you. You can get these variant styles

by using the "Form" parameter to the @Make command; see section 4.3 for details.

3. If none of the standard or variant styles pleases you, then you are a very finicky person. To indulge your finickiness you are going to have to work hard. If you are not willing to work hard, go back to step 1; it might be more appealing the second time around. If you *are* willing to work hard, then go get a copy of the *Experts' Manual*, curl up in a comfortable place, and read it from cover to cover. If that doesn't stop you, then roll up your sleeves and go make your own document style. Please tell me about it when it's done, so that I can add it to the SCRIBE database for other people to use.

One final behest: please don't complain to me about the formats that SCRIBE produces. SCRIBE is capable of producing almost any format at all, but it comes from the factory equipped to produce formats that I happen to like. If you don't like my formats, don't complain to me about it. Either learn to like them or make your own.

I. A Few 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.

I.1 How to Make a Title Page

Document Types REPORT, MANUAL, and ARTICLE contain a title page facility; you can also say "@Make(TitlePage)" 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
@value(Date)

@end(TitleBox)
@begin(Heading)
Carnegie-Mellon University
Computer Science Department
@end(Heading)
This manual corresponds to SCRIBE version @value(ScribeVersion).
@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-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.
@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

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

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.

I.2 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[h] + 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 $n \geq 10$, 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

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

   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.

I.3 Printing on Model Paper

Academic journals frequently request that papers submitted for publication be typed on "model paper", which they provide. Model paper is usually larger than the normal 8.5-by-11 inch size, and has various blue lines on it that demarcate the area in which your paper is supposed to go.

It is usually convenient to proof your document on normal-sized paper, then convert it to the format required by the journal. There are dozens of different kinds of model paper in use. Ideally, SCRIBE would have a document type for each, so you could just say "@Make(NorthHolland)", for example, and generate model paper for North-Holland journals. In time these document types will become available; right now they are not.

There are two approaches to the preparation of model paper. You may use the Diablo, typing directly on the paper, or you may cut and paste. In either case, you will need to be able to control page size, margins, and line spacing.[1]

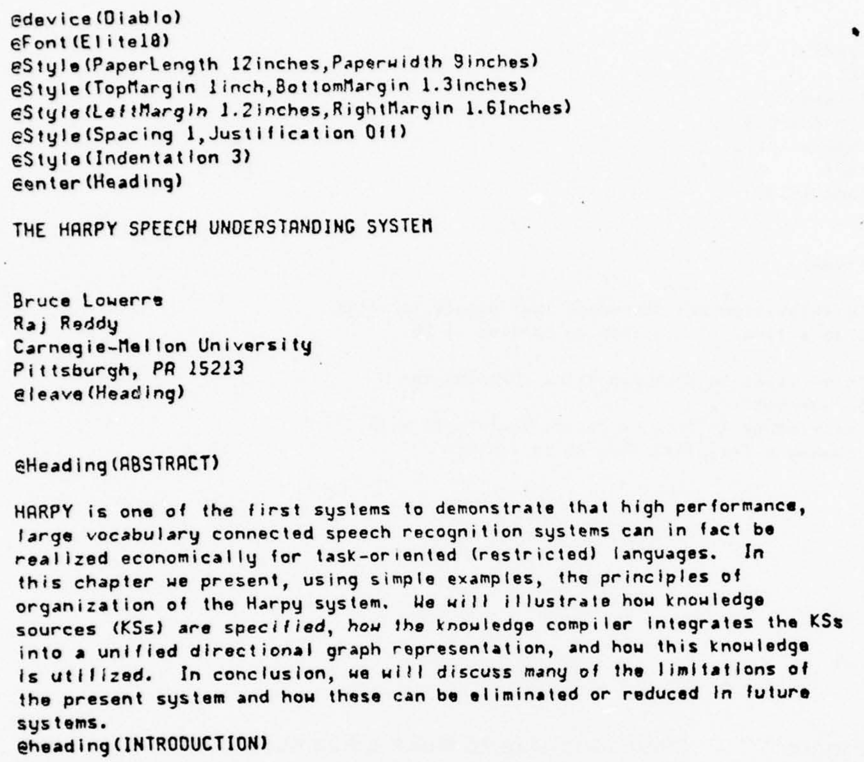
All of the information that you need to produce model paper is in chapter 9, but it may not have been obvious to you how to put it all together. Figure 12-1 shows an excerpt of a sample manuscript file used to prepare a document to be typed directly on the model paper with the Diablo typewriter. To print with the XGP to cut and paste, you would use a similar approach.

I.4 Business Letters

SCRIBE will format a business letter for you and print it either on the XGP, simulating stationery, or on the Diablo, typed on real stationery. Your manuscript file will be in the same format either way. Figure 12-2 shows a sample manuscript file that will produce a business letter, albeit a whimsical one. Figure 12-3 shows how that letter would look when printed on the XGP.

[1]If you choose to cut and paste, I recommend Carter's Glue Sticks for the pasting part.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC



```
@device(Diablo)
@Font(Elite10)
@Style(PaperLength 12inches,Paperwidth 9inches)
@Style(TopMargin 1inch,BottomMargin 1.3inches)
@Style(LeftMargin 1.2inches,RightMargin 1.6Inches)
@Style(Spacing 1,Justification Off)
@Style(Indentation 3)
@enter(Heading)

THE HARPY SPEECH UNDERSTANDING SYSTEM


Bruce Lowerre
Raj Reddy
Carnegie-Mellon University
Pittsburgh, PA 15213
@leave(Heading)


@Heading(ABSTRACT)

HARPY is one of the first systems to demonstrate that high performance,
large vocabulary connected speech recognition systems can in fact be
realized economically for task-oriented (restricted) languages.  In
this chapter we present, using simple examples, the principles of
organization of the Harpy system.  We will illustrate how knowledge
sources (KSs) are specified, how the knowledge compiler integrates the KSs
into a unified directional graph representation, and how this knowledge
is utilized.  In conclusion, we will discuss many of the limitations of
the present system and how these can be eliminated or reduced in future
systems.
@heading(INTRODUCTION)
```

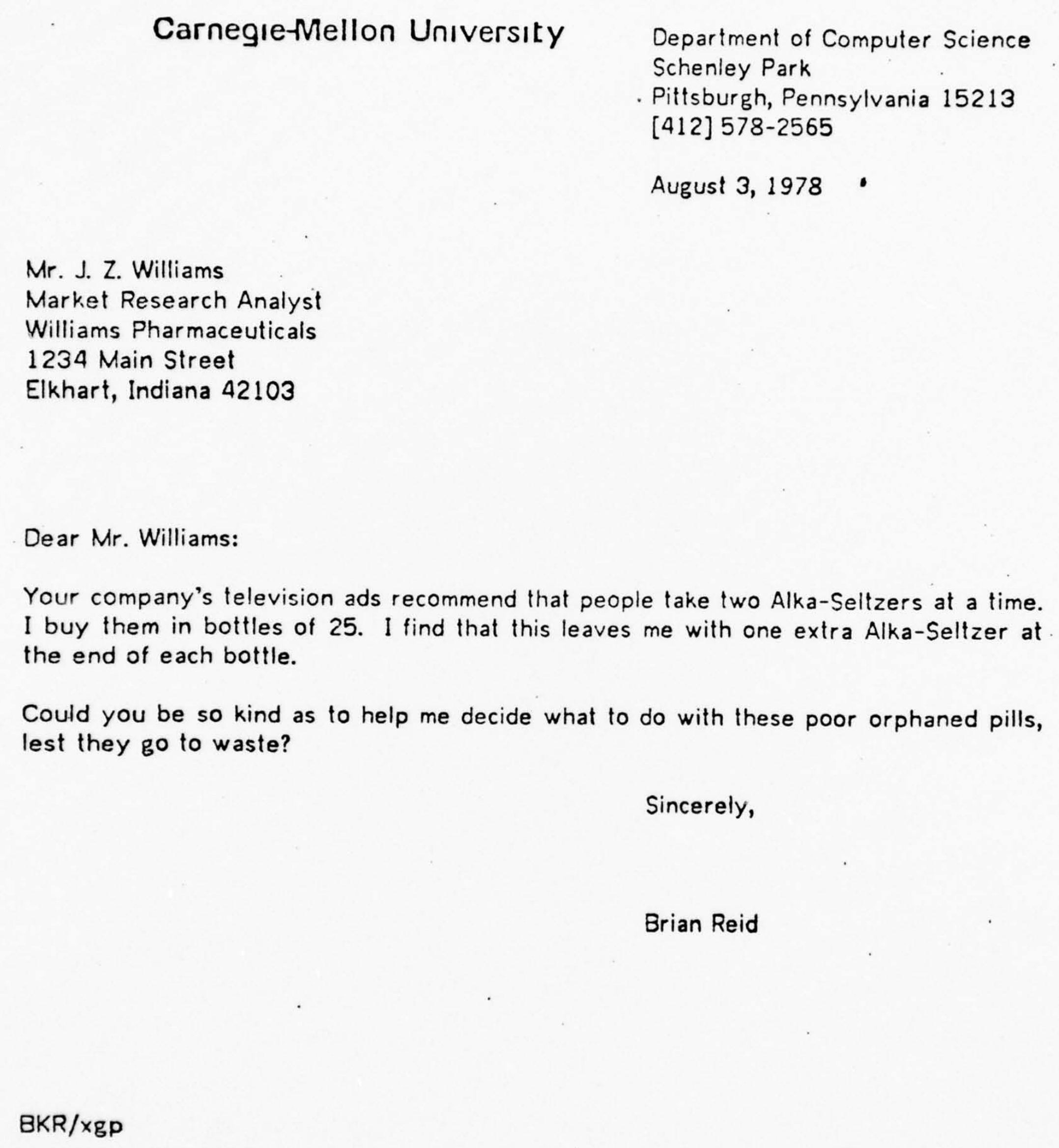
Figure 12-1: Sample text for paper typed onto model paper

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

```
@device(XGP)
@make(Letterhead)
@begin(Address)
Mr. J. Z. Williams
Market Research Analyst
Williams Pharmaceuticals
1234 Main Street
Elkhart, Indiana 42183
@end(Address)
@begin(Body)
Dear Mr. Williams:

Your company's television ads recommend that people take two
Alka-Seltzers at a time.  I buy them in bottles of 25.

I find that this leaves me with one extra Alka-Seltzer
at the end of each bottle.
Could you be so kind as to help me decide what to do with
these poor orphaned pills, lest they go to waste?
@end(body)
Sincerely,



Brian Reid
@begin(Notations)
BKR/xgp
@end(Notations)
```

Figure 12-2: Manuscript file to make a business letter

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC



Carnegie-Mellon University

Department of Computer Science
Schenley Park
Pittsburgh, Pennsylvania 15213
[412] 578-2565

August 3, 1978

Mr. J. Z. Williams
Market Research Analyst
Williams Pharmaceuticals
1234 Main Street
Elkhart, Indiana 42103

Dear Mr. Williams:

Your company's television ads recommend that people take two Alka-Seltzers at a time. I buy them in bottles of 25. I find that this leaves me with one extra Alka-Seltzer at the end of each bottle.

Could you be so kind as to help me decide what to do with these poor orphaned pills, lest they go to waste?

Sincerely,

Brian Reid

BKR/xgp

Figure 12-3: XGP output produced by sample letter file

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

II. Font Samples

This appendix contains samples of the various fonts that are available on our XGP system. Remember that each 'font' is a family of character sets that are used together, and that by selecting one of these fonts, you determine the character set that will be used for italics, boldface, small caps, and so forth as well as the 'normal' font.

To print your document in one of these fonts, use the @Font command. Squeeze the spaces out of the name of the font, and then say "@Font(name)". Thus:

```
@Font(Nonie12)
@Font(NewsGothic10)
@font(Baskerville12)
```

and so forth.

The text is taken from various fables of these and other times.

PRECEDING PAGE BLANK-NOT FILMED

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

News Gothic 12

News Gothic 12 is an expanded version of the standard News Gothic 10 font. It is popular with people who like to photoreduce documents: when standard 70 percent reduction is performed on News Gothic 12, it looks quite nice and is quite readable.

Normal (@R)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :
;<=>?@ !"#$%&*+,-./
↓→↔∝β∈πλ∧∨¬≡∀∃⊂⊃∩∪⊗≠
≤≥∞∂_~
abcdefghijklmnopqrstuvwx
yz

Boldface (@B)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}'¬ 0123456789
:;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwx
yz

Italics I
ABCDEFGHIJKLMNOPQRS
TUVWXYZ

abcdefghijklmnopqrstuvw
xyz

SMALL CAPS (@C)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>
?@ !"#$%&*+,-./

The declivity was so small, that I walked near a mile before I got to the shore, which I conjectured was about eight o'clock in the evening. I then advanced forward near half a mile, but could not discover any sign of houses or inhabitants; at least I was in so weak a condition, that I did not observe them.

I was extremely tired, and with that, and the heat of the weather, and about half a pint of brandy that I drank as I left the ship, I found myself much inclined to sleep.

I lay down on the grass, which was very short and soft, where I slept sounder than I ever remember to have done in my life, and, as I reckoned, above nine hours; for when I awaked, it was just daylight. I attempted to rise, but was not able to stir: for as I happened to lie on my back, I found my arms and legs were strongly fastened on each side to the ground; and my hair, which was long and thick, tied down in the same manner.

AD-A064 808 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 5/2
SCRIBE INTRODUCTORY USER'S MANUAL.(U)
AUG 78 B K REID F44620-73-C-0074
UNCLASSIFIED AFOSR-TR-79-0045 NL

2 OF 2
AD
A064808

END
DATE
FILMED
4-79
DDC

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

News Gothic 10

News Gothic 10 is the font that you get when you don't include an explicit @Font command. It is simple and readable and small enough to fit a lot of letters onto the page.

Normal (@R)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>
?@ !"#$%&*+,-./
↓→↔∝β⊂πλ∧∨¬≡∀∃⊂⊃∩∪⊗≠≤≥∞
∂_~
abcdefghijklmnopqrstuvwxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>
?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Italics (@I)
ABCDEFGHIJKLMNOPQRSTUV
WXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?
@ !"#$%&+,-./*
abcdefghijklmnopqrstuvwxyz

SMALL CAPS (@C)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?@ !"#$%
&*+,-./

R: No, I don't.
P: You think this is the right step to go?
R: I do.
P: It can occur -- it's going to be -- it's going to be bloody.
R: I think that.
P: Believe me.
R: That the top people in government deserve the same consideration as anybody else.
P: Damn right.
R: The idea that a top person in government is, you know -- it isn't the question beyond reproach, you know. A person could be beyond reproach. Take me -- I should have been fired many times because I've been so heavily criticized in the press, many of those were things I didn't do. You remember?
R: Well, as a matter of fact, it's a little bit the same attitude that Lucius Clay had about you and the fund.
P: Right.
R: That there's Mr. Eisenhower and you should get off. Well, that wasn't really what he said.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Nonie 12

Nonie12 is a nice sans-serif text font. It is very nicely proportioned and very readable, although it is somewhat larger than usual body fonts.

Charles I of England had an illegitimate daughter named Nonie, and the name has been passed down maternally to the firstborn daughter through all the intervening generations. We aren't sure who created this font, so we can't check to see if it was named after one of those Nonies.

Normal (@R)
ABCDEFGHIJKLMNOPQRSTU
VWXYZ
[\]←(↑)'{|}'¬ 0123456789
:;<=>?@ !"#$%&*+,-./
↓→↔∝β∈πλ∧∨¬≡∀∃⊂⊃∩∪⊗≠
≤≥∞∂ _ ~
abcdefghijklmnopqrstuvwx
yz

Boldface (@B)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}' 0123456789
:;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvw
xyz

Italics (@I)
ABCDEFGHIJKLMNOPQRSTU
VWXYZ
[\]←(↑)'{|}' 0123456789
:;<=>?@ !"#$%&+,-./*
abcdefghijklmnopqrstuvwx
yz

SMALL CAPS (@C)
ABCDEFGHIJKLMNOPQRSTUVWX
YZ
[\]←(↑)'{|}' 0123456789 :;<=>?
@ !"#$%&*+,-./

Bold italics (@P)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}' 012345678
9 :;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvw
xyz

Once upon a time the Hares found themselves mightily unsatisfied with the miserable condition they lived in. Here we live, says one of them, at the mercy of men, dogs, eagles, and I know not how many other creatures, which prey upon us at pleasure; perpetually in frights, perpetually in danger; and therefore I am absolutely of opinion, that we had better die once for all, than live at this rate in a continual dread that's worse than death itself. The motion was seconded and debated, and a resolution immediately taken, by one and all, to drown themselves. The vote was no sooner passed, but away they scudded with that determination to the next lake. Upon this hurry there leapt a whole shoal of Frogs from the bank into the water, for fear of the Hares. Nay then, my masters, says one of the gravest of the company, pray let's have a little patience. Our condition, I find, is not altogether so bad as we fancied it; for there are those, you see, that are as much afraid of us as we are of others.

There is no contending with the Orders and Decrees of Providence. He that makes us, knows what is fittest for us; and every man's own lot (well understood and managed) is undoubtedly the best.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Nonie 9

Nonie 9 is a small but readable *sans-serif font*, suitable for footnotes, inset text, and in some circumstances body text.

Normal (@R)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]←(↑)'{|}' 0123456789 :;<=>?@ !"
#$%&*+,-./
_~
abcdefghijklmnopqrstuvwxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQRSTUVWX
YZ
[\]←(↑)'{|}' 0123456789 :;<=>?
@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Italics (@I)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]←(↑)'{|}' 0123456789 :;<=>?
@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Bold Italics (@P)
ABCDEFGHIJKLMNOPQRSTUVWXY
Z
[\]←(↑)'{|}' 0123456789 :;<=>
?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

I was determined that we should get to the bottom of the matter, and that the truth should be fully brought out -- no matter who was involved. At the same time, I was determined not to take precipitate action, and to avoid, if at all possible, any action that would appear to reflect on innocent people. I wanted to be fair. But I knew that in the *final* analysis, the integrity of this office -- public faith in the integrity of this office -- would have to take priority over all personal considerations.

Today, in one of the most difficult decisions of my Presidency, I accepted the resignatins of two of my closest associates in the White House -- Bob Haldeman, John Ehrlichman -- two of the finest public sevants it has been my privilege to know.

I want to stress that in accepting these resignations, I mean to leave no implication whatever of personal wrongdoing on their part, and I leave no implication tonight of *implication on the part of others* who have been charged in this matter. But in matters as sensitive as guarding the integrity of our democratic process, it is essential not only that rigorous legal and ethical standards be observed, but also that the public, you, have the total confidence that they are both being observed and enforced by those in authority and particularly by the President of the United States. They agreed with me that this *move was necessary in order to restore that* confidence.

Because Attorney General Kleindienst -- though a distinguished public servant, my personal friend for 20 years, with no personal involvement whatever in this matter -- has been a close personal and professional associate of some of those who are involved in this case, he and I both felt that it was also necessary to name a new Attorney General.

The Counsel to the President, John Dean, has also resigned.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Meteor 12

Meteor 12 is a nice text body font. It has broad serifs, good spacing, and is very readable. It is a bit larger than the normal 10-point fonts.

Normal (@R)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}' 0123456789 :;
<=>?@ !"#$%&*+,-./
∈∧∨≠≤≥__~
abcdefghijklmnopqrstuv
wxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQR
STUVWXYZ
[\]←(↑)'{|}' 0123456789
:;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstu
vwxyz

Italics (@I)
ABCDEFGHIJKLMNOPQRS
TUVWXYZ
[\]←(↑)'{|}' 0123456789
:;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuv
wxyz

SMALL CAPS (@C)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}' 0123456789 :;<=>
?@ !"#$%&*+,-./

Bold italics (@P)
ABCDEFGHIJKLMNOPQR
STUVWXYZ
[\]←(↑)'{|}' 012345678
9 :;<=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstu
vwxyz

I want to talk to you tonight from my heart on a subject of deep concern to every American.

In recent months, members of my Administration and officials of the Committee for the Re-election of the President -- including some of my closest friends and most trusted aides -- have been charged with involvement in what has come to be known as the Watergate affair. These include charges of illegal activity during and preceding the 1972 Presidential election and charges that responsible officials participated in efforts to cover up that illegal activity.

The inevitable result of these charges has been to raise serious questions about the integrity of the White House itself. Tonight I wish to address those questions.

Last June 17, while I was in Florida trying to get a few days' rest after my visit to Moscow, I first learned from news reports of the Watergate break-in. I was appalled at this senseless, illegal action, and I was shocked to learn that employees of the Re-election committee were apparently among those guilty. I immediately ordered an investigation by appropriate government authorities. On September 15, as you will recall, indictments were brought against seven defendants in the case.

As the investigations went forward, I repeatedly asked those conducting the investigation whether there was any reason to believe that members of my Administration were in any way involved. I received repeated assurances that there were not. Because of these continuing reassurances -- because I believed the reports I was getting, because I had faith in the persons from whom I was getting them -- I discounted the stories in the press that appeared to implicate members of my Administration or other officials of the campaign committee.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Meteor 9

Meteor 9 is a font series suitable for small text, footnote, quotations, etc. It is slightly too small to be valuable as a main text font.

Normal (@R)
ABCDEFGHIJKLMNOPQRSTUVWX
YZ
[\]←(↑)'{|}' 0123456789 :;<=>?@
!"#$%&*+,-./
≠≤≥_~
abcdefghijklmnopqrstuvwxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}' 0123456789 :;<=>
?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Italics (@I)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}' 0123456789 :;<=>
?@ !"#$%&+,-./*
abcdefghijklmnopqrstuvwxyz

Bold Italics (@P)
ABCDEFGHIJKLMNOPQRSTUVW
XYZ
[\]←(↑)'{|}' 0123456789 :;<=>
?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Until March of this year, I remained convinced that the denials were true and that the charges of involvement by members of the White House staff were false. The comments I made during this period, and the comments made by my Press Secretary on my behalf, were based on the information provided to us at the time we made those comments. However, new information then came to me which persuaded me that there was a real possibility that some of these charges were true, and suggesting further that there had been an effort to conceal the facts both from the public, from you, and from me.

As a result, on March 21, I personally assumed the responsibility for coordinating intensive new inquiries into the matter, and I personally ordered those conducting the investigations to get all the facts and to report them directly to me, right here in this office.

I again ordered that all persons in the Government or at the Re-election Committee should cooperate fully with the FBI, the prosecutors and the Grand Jury. I also ordered that anyone who refused to cooperate in telling the truth would be asked to resign from government service. And, with ground rules adopted that would preserve the basic constitutional separation of powers between the Congress and the Presidency, I directed that members of the White House staff should appear and testify voluntarily under oath before the Senate Committee investigating Watergate.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Baskerville 12

Baskerville was an English typographer of the first half of the eighteenth century; he was considered a dangerous revolutionary by most of his typographer associates. He invented the style of type that has come to be associated with American colonial days. The XGP Baskerville fonts are pleasant and a bit old-fashioned, but are not quite so readable as the simpler fonts. The Baskerville fonts contain ligature characters, i.e. single glyphs that print multiple letters. Ligatures are necessary when hot type is used, in order to get correct spacing with letter pairs like 'ff' and 'fl' and 'fi'. Although the XGP does not need them, they add a quaint touch to what might otherwise be a dull document.

Normal (@R)
ABCDEFGHIJKLMNOPQ
RSTUVWXYZ
[\]←(↑)'{|}'– 0123456789 :;<=>?
@ !"#$%&'*+,-./
↓→↔αβ^πλ∧fl–fi . ∃⊂⊃∩∪⊗ffi
≤ffffl∂_~
abcdefghijklmnopqrstuvwxyz
fiflffffiffl–

Boldface (@B)
ABCDEFGHIJKLMNOPQR
STUVWXYZ
[\]←(↑)'{|}'– 0123456789 :;<=>?@
!"#$%&'*+,-./
abcdefghijklmnopqrstuvwx
yz fiflffffiffl–

Italics (@I)
ABCDEFGHIJKLMNOPQ
RSTUVWXYZ
[\]←(↑)'{|~'– 0123456789 :;
<=>?@ !"#$%&'+,-./*
abcdefghijklmnopqrstuvwxyz
fiflffffiffl–

The terms "Turkestan" and "Central Asia" are often used indiscriminately to describe the whole of the immense territory to the east of the Caspian, comprised between Siberia on the north and Khorasan (Persia), Afghanistan, and Tibet on the south, or to designate separate, sometimes arbitrarily determined, parts of the same region.

In the beginning of the 19'th century the whole of the territory just named, with its great variety of altitudes, climate, inhabitants -- these last differing as much in their history as in their present characteristics -- was comprised under the vague denomination of High Tartary, or High or Interior Asia. After the appearance of Humboldt's first draft of Asie Centrale in 1831, the term "Central Asia" came into favour. But Humboldt's limits of Central Asia were too mathematical (based on degrees of latitude), and were further unsatisfactory because influenced by his erroneous conception of the mountains of Central Asia, which he supposed to run either along parallels or along meridians.

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Bodoni 10

Bodoni was an Italian printer of the latter half of the 18'th century. The fonts that he carved and used were the predecessors of today's 'modern' faces: thin round lines and short thin serifs.

Unfortunately, the very quality that made Bodoni fonts so revolutionary in their time is the same one that makes them so unreadable on our XGP: the thin lines. Since the XGP often blurs thin lines, and since the Bodoni fonts use ragged lines anyhow, they are not very satisfactory.

Bodoni 10 is terribly hard to read. It is on the system mostly for historical reasons (it was the first Roman font available here).

Normal (@R)
ABCDEFGHIJKLMNOPQRSTU
VWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?@ !
"#$%&*+,-./
↓→↔α∧β∈πλ∧∨¬≡∀∃⊂⊃∩∪⊗≠≤≥∞∂
_~
abcdefghijklmnopqrstuvwxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQRSTU
VWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?@ !
"#$%&*+,-./
abcdefghijklmnopqrstuvwxyz

Italics (@I)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?
@ !"#$%&+,-./*
abcdefghijklmnopqrstuvwxyz

Having taken leave of Liu Pei and Liu Chi, Lu Su and Chuko Liang embarked on a boat for Chaisang. The two men were discussing affairs together in the boat. Lu Su said to Chuko Liang, "When you see General Sun, don't tell him frankly that Tsao Tsao has a great many soldiers and generals."

"You need not bid me do so," replied Chuko Liang. "I know what to say and reply."

When the boat arrived, Lu Su asked Chuko Liang to take a rest in the guests' quarters while he went alone to see Sun Chuan.

Sun Chuan, assembling his civil and military officers in his audience chamber, was discussing affairs with them. Hearing that Lu Su was back, he quickly called him in and asked, "What did you discover on your trip to Chianghsia?"

"I have known the general situation. Allow me to report it to you later," replied Lu Su.

Then Sun Chuan showed a dispatch from Tsao Tsau to Lu Su and said, "Yesterday Tsao Tsao sent a messenger to bring the dispatch here. I have sent the messenger back and this gathering is considering the reply."

THIS PAGE IS BEST QUALITY PRACTICABLE
FROM COPY FURNISHED TO DDC

Bodoni 12

Bodoni 12 is a bit more readable than Bodoni 10, but it is still more showy than readable. To a certain extent, its larger size emphasizes the problems of the Bodoni fonts, rather than smoothing them as one might expect.

Normal (@R)
ABCDEFGHIJKLMNOPQRST
UVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=
>?@ !"#$%&*+,-./
↓→↔α/β∈πλ∧∨¬≡∀∃⊂⊃∩∪⊗≠
<>∞∂_~
abcdefghijklmnopqrstuvwxyz

Boldface (@B)
ABCDEFGHIJKLMNOPQR
STUVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=
>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvwxy
z

Italics (@I)
ABCDEFGHIJKLMNOP
QRSTUVWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<
=>?@ !"#$%&*+,-./
abcdefghijklmnopqrstuvw
xyz

SMALL CAPS (@C)
ABCDEFGHIJKLMNOPQRSTU
VWXYZ
[\]←(↑)'{|}'¬ 0123456789 :;<=>?@ !
"#$%&*+,-./

From back of the transmission case, install parking toggle rod and link into the case. Install parking pawl link spacer onto the case retaining pin. Dimpled side of the spacer should be facing the link.

Install the parking pawl link onto the case retaining pin. Install flat washer and retaining ring.

Position inner manual lever behind the manual lever link, with the cam on the lever contacting the lower link pin.

Install the upper end of the manual lever link onto the case retaining pin. Install flat washer and retaining ring.

Operate the manual lever and check for correct linkage operation. For each of the lever actuating arms, ensure that the correct clearance exists between the driving cam and the stop. Check this clearance while rotating the assembly by hand, or while an assistant is rotating it.

Install the top cover, using a new gasket and hardening gasket cement. Tighten all of the bolts to 10 foot-pounds. Wipe excess gasket cement from the endge of the gasket. Tighten the bolts again to 45 foot-pounds.

III. The Format of Bibliography Files

As we mentioned in chapter 8, a bibliography file is a sequence of bibliography entries. This appendix describes the details of the 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
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 must 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.

Index

[1]@Foot command 21

PRECEDING PAGE BLANK-NOT FILMED

Acknowledgements

I got by with a little help from my friends.

Bill Wulf dreamed up the SCRIBE project and got me started on it. David Lamb and Mary Shaw participated in the initial design studies, and helped formulate the language-design issues.

Craig Everhart taught me BLISS and helped me fight the vile TOPS-10 monitor. Bruce Leverett and Paul Knueven showed me how to make BLISS code portable. Andy Hisgen and Steve Hobbs helped with some sticky bugs that were too big to tackle alone.

I spent about a year programming SCRIBE, during which time I was frequently at the brink after too many hours of debugging. Elaine Rich, Richard Swan, Loretta Guarino, and Pat Metzger, my housemates, kept me fed and properly supplied with Stroh's, and pretended not to mind that I didn't do my fair share of the cooking and dishwashing. The Oakland Original Hot Dog Store provided periodic escape and wonderful french fries.

Ivor Durham was the first user; his extraordinary patience with a developmental system deserves special thanks.

Loretta Guarino, Mary Shaw, and David Phillips helped me get the bugs out of the manual. Philip Wadler did the cover drawing in India ink on parchment.

Anita Jones gave me periodic kicks or pep talks, and just generally kept me going.