SCRIBE

Document Production Software

USER MANUAL

UNILOGIC

SCRIBE

Document Production System

User Manual

UNILOGIC, Ltd.

June, 1985

Published by UNILOGIC, LTD., Suite 240, Commerce Court, Four Station Square,
Pittsburgh, PA 15219-1119

Fourth Edition.

First Printing: April, 1984
Second Printing: June, 1985

Previous Editions: August 1978, August 1979, May 1980

Preface

Scribe is a system that makes producing attractive documents as easy as creating a text file. No special typographical knowledge is required, even if you want to use color, special characters, multiple fonts, foreign alphabets, and complex mathematics. All you have to know is how to log in to a computer and use a simple editor. Scribe is designed to take the pain out of document formatting. If you leave something out or make inconsistent requests, Scribe straightens everything out for you.

It does this straightening by dividing up the work of document preparation. You, the user, must provide the *content*, which consists of text and a small number of commands that tell Scribe where various pieces of the document begin and end. The *appearance* of the document — how big the margins ought to be, what type sizes and styles are appropriate, where footnotes should be place, how headings should be numbered — all are provided for and standardized by Scribe under your control. This typographical information resides in a large Database that is supplied with Scribe. The Database is so extensive that Scribe even has its own Database management system so that it can find what it needs at the right time. The Database can be modified or expanded locally to include all sorts of customized document styles.

Scribe was originally designed and implemented by Brian K. Reid while he was a graduate student in the Computer Science Department at Carnegie-Mellon University in Pittsburgh. UNILOGIC was formed in 1979 to continue the development and popularization of Scribe. In 1982, Dr. Reid was honored by the Association for Computing Machinery with its Grace Murray Hopper Award, given annually to that individual who, while under thirty years of age, has made the most significant contribution to the computer industry.

Earlier editions of this manual were written by Brian Reid and Janet H. Walker. Portions have been revised extensively by the staff of UNILOGIC, Ltd.

This manual was produced entirely with Scribe and the POSTSCRIPT Driver, except for the outside cover and the line drawing on page 171. It was compiled on a VAX/VMS system and printed on an Apple LaserWriter with a resolution of 300 dots per inch. This printer deposits a very black, smooth character on the paper and is excellent for reproduction.

The fonts used in this manual were supplied courtesy of The Allied Corporation. The basic body size is 11-point. Their selection of mathematical and special characters is extensive, as you will see by leafing through the manual.

The Scribe Mathematics Facility is derived from a macro package known as MATHLM,

built by Louis Monier of Carnegie-Mellon University. Bibliography formats for different journal styles have been contributed by many different Scribe users and sites. We hope you will let us know how you extend Scribe to fit your needs.

Table of Contents

List of Figures

List of Tables

Chapter One
Introduction

Scribe is a computer program that helps writers practice their art. Used in conjunction with a text editor, it takes care of all of the tedious aspects of producing printed text.

To use Scribe, you prepare a *manuscript file* using a text editor. You process this manuscript file through Scribe to generate a *document file*, which you then print on some convenient printing machine to get paper copy.

Scribe controls the words, lines, pages, spacing, headings, footings, footnotes, numbering, hyphenation, tables of contents, indexes, and more. It has a Database full of *document format definitions*, which tell it the rules for formatting a document in a particular style. Under normal circumstances, writers need not concern themselves with the details of formatting, because Scribe does it for them.

For example, when you ask for the *Thesis* format, Scribe knows to leave a one-and-a-quarter-inch left margin; it knows where to put the page number; it knows that footnotes are numbered sequentially throughout the document and are placed in a section at the end called *Notes*, and so forth. On the other hand, if you ask for the *Report* format, Scribe knows to produce a document with numbered chapters, sections, subsections, and paragraphs; it knows to produce a title page; it knows to leave 1 inch margins, and so on.

Scribe also smooths over the differences among printing devices. You ask for what you want and Scribe does its best with the printing device it has. For example, if you request that a word be printed in italics, but the printing machine cannot print italics, then Scribe underlines it instead. When printing a heading, Scribe uses larger letters if the printing machine can change letter size; if not, then it uses some other style, such as capitalizing and centering the heading.

Scribe tries to assist you with several aspects of the document assembly task. It provides you with an outline of the manuscript under construction, generates an Index and Table of Contents, and can provide you with a sorted list of words used in the document, as well as generating a Bibliography, List of Figures, and List of Tables.

You will find that Scribe appears to know a great deal about how documents should be formatted, what type sizes and alphabets are available on different printers, where footnotes should go, and other typographical details. Most of this knowledge is stored in a separate set of files known as the Scribe Database. The Database can be extended or modified as needed

at individual sites. For example, there is no limit on the number of customized document styles that may be added. When your computer center purchases new fonts for its laser printer, for example, there is a way of telling Scribe about them by updating the Database.

Maintaining the Scribe Database is difficult and time-consuming work. For this reason, it is kept out of the hands of users and is assigned to a Scribe Database Administrator, or ***DBA***, at each site. Scribe provides a large set of facilities to permit the ***DBA*** to create new document types, add support for additional output devices, and perform other chores. Most of these tools are not even hinted at in this manual. The ***DBA*** has to read another whole manual that is even thicker than this one. Should you have any difficulty in using Scribe, the ***DBA*** should be your principal source of help.

This manual is organized primarily as a tutorial and designed for sequential reading. We suggest that you read Chapters 1 through 5 before you try to use Scribe. You can safely ignore material in the later chapters until you have gained experience with the program.

1.1 Some Explanation for Non-Programmers

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

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

1.2 Some Explanation for Programmers

The Scribe system was designed to make document production easy for nonexperts, allowing them 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 of achieving it. For example, the usual convention for the appearance of a quotation in running text requires that you switch to single spacing, indent the left and right margins, place the quotation, and then change back again to normal margins and spacing. In Scribe, you place @Begin(Quotation) before the text of the quotation and @End(Quotation) after its text. The specifications for changing spacing and margins (and anything else necessary to print a quotation) are stored in Scribe's Database.

The nonprocedural nature of Scribe typically gives fits to people who have grown accus-

tomed 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. The design of Scribe was predicated on the belief that portability and procedurality don't mix, and therefore discourages procedurality.

Scribe has been in use since January 1978 and in general distribution since summer of 1979. It is now used regularly by tens of thousands of people at hundreds of sites. We have found that even experienced programmers can learn how to use Scribe effectively, but that they must learn to think differently before they are comfortable with it. People who aren't programmers or experts at older document systems have had no difficulty learning to use Scribe.

Chapter Two
Getting Started

This chapter is for beginners who have never used Scribe before (or have never used any text formatting program) and who would like to learn how to use it for producing simple documents.

2.1 The General Picture

You are starting with some material either in your head or handwritten, and you want a printed version of it. You know the purpose of your material — a letter, a memo, an article for a scientific journal, a chapter of a book, a draft of a thesis, a ransom note, or whatever — and that means you have preconceptions about what appearance and form the material must take.

Scribe has been designed also to have preconceptions about the components and appearance of difference kinds of documents. These preconceptions are contained in Scribe's *Database* of specifications for the appearance of different kinds of documents. (Chapter 5 explains the document types.) It would be nice if your needs matched Scribe's assumptions, but even if they don't, you aren't stuck. Later on in this manual, we explain how to change things (Chapter 15).

Scribe is a formatting *program* and, like all useful programs, it accepts *input* and produces *output*. The input to Scribe is a file containing both the text for your document and the instructions telling Scribe what kind of document it is and what components it has. Using the information in your file and the formats in its Database, Scribe creates an output file tailored for your particular printer, which you then print to get paper output.

You follow the same general sequence of actions for any document. First, decide what you want to do, and choose the appropriate Scribe document type. Then, create the input file containing your manuscript (called a *manuscript file* in this manual).[1] Whenever you want to see how the document will look, run Scribe to produce a *document file* that you can print. Then make any necessary revisions to the manuscript file, process it through Scribe again, print it again, and so on.

---

[1] Create the file with any convenient text editor.

That's what it is all about. The rest of this manual explains everything you need to know to use Scribe, starting with the simplest case. If you are a beginning Scribe user, you will find it helpful to read this manual from the beginning rather than hopping around. As you become more experienced, you will find the reference appendixes and the index more useful than the tutorial, but please read the tutorial at least once to become familiar with our terminology.

2.2 Preparing Input for Scribe

A manuscript file that you prepare for Scribe consists primarily of text, but it also has whatever instructions are necessary to get it formatted properly. Both you and Scribe recognize instructions by the fact that they are prefixed by an ''@'' character. Since the @-sign has this special meaning to Scribe, to get a literal @-sign to appear in your output, you must type twice as many as you really want. For example, type two @-signs to get one printed; type four @-signs to get two printed. The next few sections concentrate on simple text; later chapters describe the ''@'' commands that control the format of the text.

Normal text is composed of words, sentences, paragraphs, and, sometimes, larger units like chapters or sections. For Scribe to format your text properly, it must be able to recognize these units. For example, when it recognizes a sentence, it puts two spaces after the final period. It puts one space after a period that does not end a sentence. It normally indents the first line of each paragraph. Remember these simple rules that Scribe uses:

1. *Words* are separated by spaces or end-of-line.
2. *Sentences* are ended by a full stop character (''.'' or ''?'' or ''!'') having at least two spaces after it. The end of a line counts as two spaces.
3. A *paragraph* is ended by one or more blank lines. By ''blank line'' we mean a line that looks blank when printed on your terminal.

You now know enough about Scribe to produce a simple document. Just don't use any ''@'' signs in your text.

2.3 Processing Your File with Scribe

Suppose that you have created a manuscript file, and now you want to process it into a document file for printing. For example, suppose that you have built a manuscript file named TRIAL.MSS, with the contents as shown below. Notice the label **''Manuscript Form''** in front of it. In this manual, there are many examples showing pieces of a manuscript file and the document form that results when you process that manuscript file with Scribe; we use the labels **''Manuscript Form''** and **''Document Result''** to identify them.

Manuscript Form:

```
M & M Enterprises verged on collapse.  Milo cursed
himself hourly for his monumental greed and
stupidity in purchasing the entire Egyptian
cotton crop, but a contract was a contract and
had to be honored, and one night,
after a sumptuous evening meal, all Milo's fighters
and bombers took off, joined in formation directly
```

```
overhead, and began dropping bombs on the
group.

He had landed another contract with the
Germans, this time to bomb his own outfit.
Milo's planes separated in a well-coordinated
attack and bombed the fuel stocks and the
ordinance dump, the repair hangars and
the B-25 bombers resting on the lollipop-shaped
hardstands at the field.  His crews spared
the landing strip and the mess halls so that they
could enjoy a hot snack before retiring.
```

The blank line indicates the end of one paragraph and the beginning of another. To process this file, we run Scribe. The particular way of running Scribe varies from one computer to another. Appendix A shows you how to run Scribe on many different computers. On most systems, you type either R SCRIBE or just SCRIBE. Scribe responds by typing its sign-on line and then printing an asterisk to wait for you to type something:

```
@r scribe
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
*
```

In response to the prompt, type the name of the manuscript file that is to be processed. We've built a sample file named TRIAL.MSS, so we will type its name in response to the asterisk:

```
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
*trial.mss
```

Scribe begins processing the file when you press the RETURN key. It prints on your terminal a running account of what it is doing. You should see something like this example:

```
@r scribe
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
*trial.mss
[Processing TRIAL.MSS
    [Device "LPT"]

    [Document type "TEXT"]
]
 1.

**TRIAL.LPT for device LPT has 1 page.

@
```

We'll talk later on about what device and document types are; right now, notice what it told you about the formatted document file:

```
**TRIAL.LPT for device LPT has 1 page.
```

LPT is computerese for Line PrinTer, and TRIAL.LPT is the name of the file that Scribe has

produced. When you print this file on your line printer, it looks somewhat like this result:[2]

Document Result:

```
M & M Enterprises verged on collapse.  Milo cursed himself
hourly for his monumental greed and stupidity in purchasing
the entire Egyptian cotton crop, but a contract was a
contract and had to be honored, and one night, after a
sumptuous evening meal, all Milo's fighters and bombers took
off, joined in formation directly overhead, and began
dropping bombs on the group.

He had landed another contract with the Germans, this time
to bomb his own outfit.  Milo's planes separated in a
well-coordinated attack and bombed the fuel stocks and the
ordnance dump, the repair hangars and the B-25 bombers
resting on the lollipop-shaped hardstands at the field.  His
crews spared the landing strip and the mess halls so that
they could enjoy a hot snack before retiring.
```

Compare this output with the manuscript file. Notice that Scribe has made two normal-looking paragraphs by filling up the lines with words and adding some blanks to line up the right margin.

You're invited to try this example on your own computer right now. Use your favorite text editor to produce a file named TRIAL.MSS, and then process it with Scribe to produce the formatted file TRIAL.LPT. Take a look at it either by printing it on your line printer or by typing it on your terminal.

2.4 Printing Devices

The manuscript file that you prepare for Scribe does not contain anything that constrains it to any particular printing device. However, when Scribe processes a manuscript file into a document file, it does so with a particular printing device in mind. It must have some particular printing device in mind in order for it to know how many characters to fit on a line, how many lines to fit on a page, how to print headings or italics, and so forth.

When you don't tell Scribe anything one way or another about printing devices, it assumes that you are preparing a file for the standard printer at your site. Often the default device is the LPT printer, which is the simplest and least attractive printing device. On the other hand, if you have some better printing device and want to use it, you have to tell Scribe to produce output for that device. You can specify the device by including a *command-line option* when you give Scribe the name of the file to process. The term *command-line option* is a general piece of computerese. What it means here is that you must follow the filename with some character and the code for the printing device. There is a table of the options recognized by Scribe for all operating systems on page 203 in Appendix E.1. (Appendix A, which discusses operating system dependencies, lists other options that are available for some operating systems and gives the appropriate command-line character for each operating system.) For exam-

---

[2] From *Catch-22*, by Joseph Heller (Simon and Schuster, 1961).

ple, on a VAX/VMS system, typing **trial.mss/device:lg1200** indicates that you would like output for a QMS Lasergrafix 1200 Laser Printer, and **trial.mss/x9700** means that you would like output for an Xerox 9700 Printer. The syntax of using command-line options differs with operating systems. If you used an LG1200 on a TOPS-20 system, the whole script would look like this sample:

```
@r scribe
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
*trial.mss/device:lg1200
[Processing TRIAL.MSS
    [Device "LG1200"]
    [Document type "TEXT"
        [FontFamily Roman10]
    ]
]
 1.

**TRIAL.LG1200 for device LG1200 has 1 page.

@
```

Compare this example with the example shown for the line printer. Notice that the formatted file has a different name this time, to remind you that it was built to be printed on a different printer.

Scribe doesn't have a different command-line option for every possible output device, just the most common ones. You can specify any device on Scribe's command line, though, by using the **Device** option. It is explained in detail in Appendix E.1.

You can put a command into your manuscript file that tells Scribe which device to use. It is the @Device command. Suppose your file starts with the following command:

```
@Device(Lg1200)
```

This command tells Scribe to process the file for "Lg1200" instead of for "LPT" as the default device when it processes that file. If your file says one thing and you say another when you run Scribe, Scribe listens to you. That is, your option *overrides* the @Device command in the file. Thus, if your manuscript file has the aforementioned @Device(Lg1200) command in it, but you include the option **x9700** when you run the program, it uses "x9700" and not "Lg1200" for the device. The devices currently known to Scribe are shown in Table 2-1 on page 10. More detailed explanations of the various printing devices are in Appendix B. As you look at the list of devices that Scribe supports, remember that it is always growing and that the list reflects the devices available at the time that this manual was written.

Not every feature provided by Scribe can be realized on every output device. The line printer, for example, cannot print Italics or Greek. It can, however, generate boldface by overstriking. The Diablo can go into its so-called graphics mode and generate special characters, but it cannot print letters of varying size.

The manuscript file that you prepare for Scribe is supposed to describe the text, not describe some particular way of printing it. Suppose you have a foreign word; just ask for italics. If your document is being printed on a device that cannot italicize (a Diablo or a line

Device Name	*File Type*[3]	*Page Length*	*Page Width*	*Description*
Agile	.POD	11 inches	8.5 inches	Agile A1 Series Data Terminal
AJ832	.POD	11 inches	8.5 inches	Anderson-Jacobsen 832
AJ833	.POD	11 inches	8.5 inches	Anderson-Jacobsen 833
CAT4	.GSI	11 inches	7.75 inches	GSI CAT-4 Photocomposer
CG8600	.CG	11 inches	7.5 inches	Compugraphic 8600
CRT	.DOC	24 lines	79 cols	CRT Display
Diablo	.POD	11 inches	8.5 inches	Diablo HyTyper Printing Terminal
Dover[4]	.PRESS	11 inches	8.5 inches	Xerox Dover Printer
File	.DOC	*no limit*	79 cols	Simple on-line documentation file
Gigi	.GG	479 rasters	767 rasters	DEC VK100 (GIGI) Terminal
GP300	.PGP	11 inches	8.5 inches	Philips GP 300
GSI	.GSI	11 inches	7.75 inches	GSI CAT-8 Photocomposer
Imagen300	.IMP	11 inches	8.5 inches	Imagen 8/300
Imagen480	.IMP	11 inches	8.5 inches	Imagen 5/480
Imprint10	.IMP	11 inches	8.5 inches	Imagen Imprint-10
LA36	.TXT	66 lines	132 cols	DECwriter II
LG1200	.LG1200	11 inches	8.5 inches	QMS Lasergrafix 800, 1200 & 2400
LGP1	.LGP	11 inches	8.5 inches	Symbolics LGP-1
LPT	.LPT	57 lines	132 cols	Line Printer
Omnitech	.OMNI	11 inches	8.5 inches	Mergenthaler Omnitech/2000
PagedFile	.DOC	57 lines	132 cols	"File" but with page marks
PostScript	.PS	11 inches	8.5 inches	Apple LaserWriter (POSTSCRIPT)
Printronix	.LPT	57 lines	132 cols	Printronix Printer
Santec	.VFX	11 inches	8.5 inches	Santec S700 Veriflex
SpinWriter	.POD	11 inches	8.5 inches	NEC SpinWriter Terminal
Talaris	.LG1200	11 inches	8.5 inches	Talaris 800, 1200 & 2400
TI700	.TXT	66 lines	80 cols	TI Silent 700
TI725	.TXT	66 lines	80 cols	TI Silent 725
VIP	.VIP	11 inches	7.5 inches	Mergenthaler Linotype VIP
X2700	.X27	11 inches	8.5 inches	Xerox 2700
X2700II	.X27	11 inches	8.5 inches	Xerox 2700 II
X9700	.X9700	11 inches	8.5 inches	Xerox 8700 & 9700 Printer

Table 2-1: Devices Known to Scribe

printer, for example), the word comes out underlined. But if someday your text is being printed on a machine that can print italics, then you won't need to make any changes if you've asked for italics in the first place.

In Appendix B, we discuss low-level details of various printing devices that you can use with Scribe.

---

[3] The "file type" of a file is sometimes referred to as its "extension". In this manual, we will use the terminology "file type".

[4] Scribe also drives the Penguin printer. Since it is very much like a Dover printer, they are both in this category and both specified by the @Device(Dover) command.

Chapter Three
Simple Formatting Environments

You now know how to build a manuscript file that contains plain text and how to run it through Scribe to produce a document. In this section, we describe how to tell Scribe more about the appearance of the text, using italics, footnotes, and so forth.

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

Manuscript Form:

```
The inscription read, ``@i[De minimis non curat lex].''
```

This line produces the following output:

Document Result:

The inscription read, ''*De minimis non curat lex.*''

You can probably guess what happened here. The instruction @i means ''use italics'', and the square brackets just tell Scribe how much of the text to italicize. The name *i* is an *environment*. Scribe text is formatted according to the rules of the environment that contain it. The environment *i* requests italics. Most formatting effects in Scribe are achieved by placing the text in an appropriate environment.

In all Scribe command and environment names, letter case is completely unimportant. As a matter of consistency, we will show all of them with initial capitals for easy readability, except for the one-character environments, which will be shown in lower case.

Before going on with more kinds of formatting environments, we ought to talk about some nuts and bolts definitions.

3.1 Delimiting Characters

The square brackets [ ] used in the example above are called *delimiters*. They delimit (surround) the characters being italicized. Computer keyboards contain several pairs of matching left and right delimiters. You can use any of the pairs equally well. For example, you could have put @i(De minimis...) or @i<De minimis...>. 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 minimis...], Scribe would just keep italicizing until it came to a ''>'' character.

Scribe recognizes the following delimiter pairs.

(...), [...], {...}, <...>, '...', `...', and "..."

(In the line above, the fifth pair of delimiters is just a pair of normal apostrophes; in the sixth pair, the opening delimiter is the left quote character, or backwards apostrophe, and the closing delimiter is the normal apostrophe. On some printing devices, you can't tell them apart.)

In this manual, we usually use (parentheses) as delimiters, for no particularly good reason. (It probably helps to develop consistent habits about delimiters.)

Scribe allows you to combine requests for text characteristics. Suppose that you wanted an italic superscript. Normally for italics, you use @i(...) around the text to be italicized. For a superscript, you put @+(...) around the text to be superscripted. You can combine the two without worrying about their interaction:

Manuscript Form:

```
Why would anybody want @i(@+(italic superscripts))?
```

Document Result:

Why would anybody want $^{\textit{italic superscripts}}$?

Putting one environment completely inside another is called *nesting* of the environments.

The concept of nesting should be familiar to everyone in the notion of quoting a source inside a quotation; the inner quotation is delimited with single quotes, and the outer quotation is delimited with double quotes:

> Macomber said, ''But the sign said 'No Hunting'.''

Scribe is clever enough to let you nest delimiters that look the same without getting itself confused, but you can use different delimiters if you want:

```
Why would anybody want @i<@+(italic superscripts)>?
```

If you want to include a delimiter character (like a parenthesis) inside a pair of delimiters, you have to be careful. Suppose, for example, that you wanted to put an underlined ''('' character in your text. An attempt to produce this output the wrong way might look like the following example:

Manuscript Form:

```
there is an @u(incorrect ``)'' here)
```

In this case, the result that you really intended was the following:

Document Result:

there is an <u>incorrect</u> <u>''(''</u> <u>here</u>

However, Scribe would treat the first ''('' it found as the end of the request for underlining, and it produces something looking like this line:

Document Result:

there is an incorrect ''('' here)

You might think that our heeding is 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]` works, but `@i(2*(4+y)-b)` does not. It prints as ''*2*(4+y*-b)'', which is very subtly wrong.

3.2 Environments for Changing Typeface

Standard practice in publication calls for certain words to be set in italic or boldface. Sometimes it is useful to use small capitals. We sometimes want to put Greek letters in text. All of these effects, and more, amount to changing the typeface in which the letters are printed, without changing the letters themselves.

Typefaces and fonts are confusing, and the ambiguous terminology makes them more so. The English word *font*, for example, is mightily ambiguous: it can mean something like ''italic'' or ''boldface''; it can mean something like ''Helvetica''; it can mean ''Helvetica italic''; it can also mean the bowl in a church where people are baptized.

Scribe uses a three-level scheme for naming type faces:

- a *FontFamily*, which is a group of fonts that have been chosen by a document designer to look harmonious when used together. Examples of FontFamilies are ''Times Roman'' or ''News Gothic''. You can specify the FontFamily for your document by using the @Style command, which is explained on page 39.
- a *Font*, which is some particular member of a FontFamily, for example, ''BodyFont'' or ''SmallBodyFont''. Scribe users seldom need to be concerned with this information.
- a *FaceCode*, which is a particular component within a Font. Kinds of FaceCodes include italics, boldface, and Greek. When you ask for the italic FaceCode in the heading Font while printing a document in Times Roman, you get whatever typeface the document designer has chosen for heading italics in Times Roman. In this section of the manual, we explain the various Scribe environments that select new FaceCodes.

Scribe recognizes all of the FaceCodes listed in Table 3-1. Whether the requested FaceCode actually appears in the document depends on the nature of the final printing device being used. If Scribe cannot produce a particular code on a particular printing device, it attempts a reasonable compromise. For example, if you ask for Greek letters and your printing device is a line printer, Scribe just echoes the @g command, printing ''@g(a)'' instead of ''α''.

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

Table 3-1: FaceCodes

3.3 Environments to Change Format

You now know how to produce paragraphs of text with various fancy effects for the appearance of the text, like italics and underlining. In this section, we describe alternate appearance of text, things other than ordinary paragraphs. Anything put into running text is called an *insert*. Inserts can be quotations, examples, equations, tables, or what have you. In Scribe, inserts are created by putting the inserted text in a named *formatting environment*. For the sake of brevity, we will, in general, just call them environments rather than formatting environments.

You must mark (delimit) the beginning and end of each piece of text that you want to have formatted in some special environment. There are two ways to do this marking — a short form and a long form. You have already seen the short form; we used it in our discussion on font-changing environments (Section 3.2, page 13). Both forms yield the same result; one is easier to type, but more prone to errors while the other is tedious, but generally easier to get correct.

Consider the Quotation environment. The long form requires you to mark the beginning and the end of the environment with two commands, @Begin and @End:

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

The short form requires you to mark the beginning and end of the environment with opening and closing delimiters:

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

If your quotation is short, you can specify it all on one line:

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

As long as the "@" character for the environment is the first character on a manuscript file line, it doesn't matter whether or not you put the delimiters on the same line as the text of the environment. That is, the following two cases would have exactly the same effect:

```
@Quotation[
Delimiters on different lines.
]
@Quotation[Delimiters on the
same line.]
```

The cases become different when the "@" character is on some position other than the first one in the line. In the following case, Scribe treats the carriage return following the command as if it were two leading spaces in the body of the environment.

```
The French have a word for it.@Quotation[
Chacun a son gout.
]
```

Since many filled environments (like Quotation) discard leading spaces, this difference often doesn't matter. However, you don't want to have to remember which environments drop leading spaces and which ones don't, so just remember to put the @-sign at the beginning of the line, and it won't matter.

The short form of an environment might be more convenient or easier to read when the text is short, but the long form has the advantage of being immune to problems with delimiters. Obviously, the short form requires that the body of the quotation not contain the right delimiter that closes the quotation. For example, the following long form specification works just fine.

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

A similar short form quotation with parentheses in the body would *not* work because the parenthesis after the word "short" would act as the closing delimiter for the quotation.

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

Table 3-2 on page 17 lists the set of simple environments in Scribe that are available in all document types. (Some document types might have specialized environments that are not

generally applicable. Those environments are not included in Table 3-2.) They can be used in either the long form, using the @Begin and @End commands:

```
@Begin[Center]
Text of the environment.
@End[Center]
```

or the short form, using delimiters:

```
@Center[Text of the environment.]
```

3.4 Unfilled Environments

The next few sections describe *unfilled* environments. In an unfilled environment, Scribe does not fill the lines by adding words until the line is full, meaning that each line in the manuscript file produces exactly one line in the document file. However, when a line in the manuscript file is too wide to become a line in the document, Scribe prints a warning message and discards the end of that line. (See Chapter 14 on Scribe's messages.)

3.4.1 Verbatim and Format

Verbatim and *Format* are similar unfilled environments. They produce text exactly as you type it, without moving margins or justifying text or any other kind of formatting. Verbatim and Format differ only in the typeface they specify. Verbatim specifies a fixed-width font; Format specifies a variable-width font (if the designated printing device has one).

With Verbatim, text appears exactly the same as you type it. Therefore, Verbatim is good for short tables. If the columns line up on your terminal, they will line up in the final document. This assured alignment is not true for Format where you must use Scribe's tabbing commands (see Chapter 9) to line up columns. On the other hand, fixed-width fonts look peculiar in a document that is set primarily in a variable-width font, so it is generally better to use Format when you can.

The "@" signs in a Verbatim environment are not taken literally; that is, @-signs still indicate Scribe commands. You can, therefore, nest any other Scribe commands or environments within either Verbatim or Format.

See Figure 3-1 on page 18 for a vivid example of the difference between Verbatim and Format.

3.4.2 Center, FlushLeft, and FlushRight

The *Center*, *FlushLeft*, and *FlushRight* environments cause each line within them to be centered, set flush left, and set flush right, respectively. Within each of these environments each line in the manuscript file produces one line in the finished document.

Environment	Description
Center	Unfilled environment. Each manuscript line centered.
Description	Filled environment. Outdented paragraphs. Single spacing with wider margins.
Display	Unfilled environment. Normal body typeface. Widens both margins.
Enumerate	Filled environment. Numbers each paragraph. Widens both margins.
Example	Unfilled environment. Uses fixed-width typeface for examples of computer type-in or type-out. Widens both margins.
FlushLeft	Unfilled environment. Prints the manuscript lines, in the normal body font, flush against the left margin.
FlushRight	Unfilled environment. Prints the manuscript lines, in the normal body font, flush against the right margin.
Format	Unfilled environment. Normal body typeface. No changes to margins. Any horizontal alignment that is needed should be done with tabbing commands.
Itemize	Filled environment. Marks each paragraph with a tick-mark or bullet. Widens both margins.
ProgramExample	Unfilled environment. Appropriate for formatting sample pieces of computer programs. Similar to Example, but different typeface.
Quotation	Filled environment. Single-spaced. Widens both margins. Indents each paragraph.
Text	Filled environment. Indented paragraphs. Double-spaced. Most often seen as a document's running text. This environment is in effect when no other environment has been specifically mentioned.
Verbatim	Unfilled environment. Fixed-width typeface. No changes to margins.
Verse	Semi-filled environment. Fills lines, but starts a new line for each line break in the manuscript. Widens both margins.

Table 3-2: Basic Environment Types

Manuscript Form:

```
@begin(Verbatim)
        Directory listing        5-May-83        2:22:41
Name Extension Len  Prot    Creation       Version

LESCAL  SAV    34  <255>    6-Aug-80
TTYCHR  SAV     7  <155>    1-Sep-81    22-1
PASCAL  SAV     2  <155>    2-Feb-83
IMPCOM  SHR    25  <155>    2-Feb-83    10D(72)-3
WATCH   SHR    11  <155>   30-Jan-79
CLOCK   LNK    48  <155>    3-Nov-79
  Total of 127 blocks in 6 files on DSKB: [1,4]
@end(Verbatim)
```

Document Result:

```
        Directory listing        5-May-83        2:22:41
 Name Extension Len  Prot    Creation       Version

 LESCAL  SAV    34  <255>    6-Aug-80
 TTYCHR  SAV     7  <155>    1-Sep-81    22-1
 PASCAL  SAV     2  <155>    2-Feb-83
 IMPCOM  SHR    25  <155>    2-Feb-83    10D(72)-3
 WATCH   SHR    11  <155>   30-Jan-79
 CLOCK   LNK    48  <155>    3-Nov-79
   Total of 127 blocks in 6 files on DSKB: [1,4]
```

Document Result Using FORMAT instead of VERBATIM:

Directory listing 5-May-83 2:22:41
Name Extension Len Prot Creation Version

LESCAL SAV 34 <255> 6-Aug-80
TTYCHR SAV 7 <155> 1-Sep-81 22-1
PASCAL SAV 2 <155> 2-Feb-83
IMPCOM SHR 25 <155> 2-Feb-83 10D(72)-3
WATCH SHR 11 <155> 30-Jan-79
CLOCK LNK 48 <155> 3-Nov-79
Total of 127 blocks in 6 files on DSKB: [1,4]

Figure 3-1: A Comparison of Fixed-Width and Variable-Width Fonts

Manuscript Form:

```
@Begin(Center)
This is a CENTER environment.
It contains
several lines.
@End(Center)
```

Document Result:

This is a CENTER environment.
It contains
several lines.

The FlushRight and FlushLeft environments work analogously; each line inside them is moved so that its right (left) end is aligned with the right (left) margin of the document. (You can align text with something other than the document margins. See Chapter 9.)

Manuscript Form:

```
@FlushRight(This line is flushed right.)
@Begin(FlushLeft)
These lines are
flushed left.
@End(FlushLeft)
```

Document Result:

This line is flushed right.

These lines are
flushed left.

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 should use @Heading(Mynah Birds) instead; see Chapter 6. (In any case, the order doesn't matter; refer to Section 3.1 regarding *nesting*.)

3.4.3 Display, Example, and ProgramExample

Display, *Example*, and *ProgramExample* are three very similar unfilled environments.

The *Display* environment is used more frequently than either of the other two similar environments. Text inside a Display has wider left and right margins, and each line in the manuscript file produces one line in the document. Display environments appear in the normal body typeface.

Here is an example using the Display environment within the context of ordinary filled text. Notice what happens to the margins and to the line breaks from the manuscript.

Manuscript Form:

```
For the cultured, these hints are sufficient;
but some elaboration of the matter seems
worth while, in the interest of the partly
cultured and the ignorant.  Now observe, the
points noted as concerns the card -- and they
are exceedingly important -- are as follows:
@Begin(Display)
Its texture.
Style of engraving.
Hour of leaving it.
@End(Display)
If these fall short of the standard established by social
law, the visitor is placed in a ``disagreeable attitude''.
```

Document Result:

For the cultured, these hints are sufficient; but some elaboration of the matter seems worth while, in the interest of the partly cultured and the ignorant. Now observe, the points noted as concerns the card -- and they are exceedingly important -- are as follows:

> Its texture.
> Style of engraving.
> Hour of leaving it.

If these fall short of the standard established by social law, the visitor is placed in a ``disagreeable attitude''.[1]

Example is just like Display, except that it uses a different typeface. It is designed for showing examples of computer type-in and type-out (useful in user's manuals). Its text appears in a typeface that is designed to look like computer output.

ProgramExample is just like Example except for its typeface. Its text appears in a small fixed-width typeface that is appropriate for the printing of some computer programs.

Here is an example using the Example environment in the context of ordinary filled text. Notice what happens with the margins, line breaks, and spaces at the beginnings of lines.

---

[1] From *Letters from the Earth*, "From an Unfinished Burlesque of Books on Etiquette", by Mark Twain.

Manuscript Form:

```
If the array A contains N integers, this
set of two nested loops will sort that
array into descending sequence, albeit
somewhat slowly:
@Begin(Example)
for I := 1 step 1 to N-1 do
  for J := I+1 step 1 until N do
    if A[I] < A[J] then A[I] swap A[J];
@End(Example)
This is the simplest case of the algorithm
known as a ``bubble sort''; it will execute
the inner statement N@+[2]/2 times.
```

Document Result:

If the array A contains N integers, this set of two nested loops will sort that array into descending sequence, albeit somewhat slowly:

```
for I := 1 step 1 to N-1 do
  for J := I+1 step 1 until N do
    if A[I] < A[J] then A[I] swap A[J];
```

This is the simplest case of the algorithm known as a ``bubble sort''; it will execute the inner statement $N^2/2$ times.

Here is a similar case, using ProgramExample instead of Example:

Manuscript Form:

```
If the array A contains N integers, this
set of two nested loops will sort that
array into descending sequence, albeit
somewhat slowly:
@Begin(ProgramExample)
for I := 1 step 1 to N-1 do
  for J := I+1 step 1 until N do
    if A[I] < A[J] then A[I] swap A[J];
@End(ProgramExample)
This is the simplest case of the algorithm
known as a ``bubble sort''; it will execute
the inner statement N@+[2]/2 times.
```

Document Result:

If the array A contains N integers, this set of two nested loops will sort that array into descending sequence, albeit somewhat slowly:

```
for I := 1 step 1 to N-1 do
  for J := I+1 step 1 until N do
    if A[I] < A[J] then A[I] swap A[J];
```

This is the simplest case of the algorithm known as a ''bubble sort''; it will execute the inner statement $N^2/2$ times.

If your only Scribe printing devices are those that cannot change fonts (such as the Diablo printer), then the typeface styles of the Example, Display, and ProgramExample environments all look identical, and you might be tempted to ignore the distinction among them. It is wise to plan ahead, though, and use the environments appropriate to the material if there is any chance your document might be saved and printed elsewhere.

3.5 Filled Environments

The next few sections describe *filled* environments. In a filled environment, Scribe doesn't pay any attention to where the lines end in the manuscript file. It fills each line in the document with words from the manuscript file until the line is full. The example in Section 2.3 showed how Scribe fills lines.

Lines in filled environments can also be *justified. Justifying* means arranging the words on the line so that the right ends of the lines are all aligned. Filled lines are not automatically justified; you can have filled lines that are not justified. You can control overall justification in your document with the @Style command (see page 39).

3.5.1 Text

Text is a filled and justified environment. It is the environment which produces the running text of the document (even though you do not need to say @Begin[Text] ... @End[Text].) Text is the simplest environment, producing filled and justified text, with indented paragraphs. This description of the Text environment has in fact been formatted using the Text environment.

3.5.2 Quotation and Verse

Quotation is a filled environment. It formats prose quotations in running text:[2]

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

Manuscript Form:

```
@Begin(Quotation)
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.
@end(Quotation)
```

Document Result:

> 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.

Verse is an environment very similar to the Quotation environment. It too is single-spaced, with wider margins. It differs from Quotation in that Verse starts a new line in the final document for each end-of-line in the manuscript file, regardless of whether the document 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 are indented a bit from the left. (In our example here, we are not using the standard "Manuscript Form" typeface because we want to show what happens with very long lines.)[3]

Manuscript Form:

```
@Begin(Verse)
Along the roads the telephone poles stand alone like words in a broken conversation
     The wire between them hums a soliloquy in an endless encore with no ovation
Like all the poetry that has poured from my lips to fall on fields empty of ears
     Good land is prose, but what grows there is the poetry of years and years.
@End(Verse)
```

Document Result:

Along the roads the telephone poles stand alone like words in a broken conversation
The wire between them hums a soliloquy in an endless encore with no ovation
Like all the poetry that has poured from my lips to fall on fields empty of ears
Good land is prose, but what grows there is the poetry of years and years.

[3] From *Awaiting the Illinois Winter* by Rich Warren, Chicago, The Unrequited Press, 1975.

3.5.3 Itemize and Enumerate

Itemize and *Enumerate* are another pair of very similar environments. They produce nicely formatted lists. The only difference between them is that Enumerate numbers the items in the list and Itemize marks the items in the margin. Writers and editors argue endlessly over when numbered lists are appropriate and when tick-mark lists (often called bulleted lists in honor of the largish dots in the margin) are appropriate.

Itemize and Enumerate each justify the paragraphs and widen the margins slightly. They both expect a sequence of paragraphs, where each paragraph is a separate item. There are instances, however, when a single item must be more than one paragraph. For that situation, use the Multiple environment. Text in @Multiple environment delimiters is considered a single item, regardless of paragraph breaks. Use of this environment is illustrated in Figure 3-2 with the Itemize environment, and Figure 3-3 shows the Multiple environment used within the Enumerate environment.

3.5.4 Description

A *Description* environment is designed for the "command description" style that is so common in reference manuals. The first line of a paragraph in a Description environment begins at the left margin of the page. The second and remaining lines are indented substantially, so that the word or phrase at the head of the description stands out.

You often need a means for separating the heading word from the rest of the text in a description item. Use the Scribe tab command @\, which in this case means "tab to the indented margin". (The @\ command is described in detail in Chapter 9.) If the heading phrase was long enough to reach the margin, then the text following the @\ command starts on the next line at the indented margin. See Figure 3-4, on page 27 for an example of input and output using the Description environment.

3.6 Simple Environments for Mathematical Text

Scribe has two features to help in producing mathematical output. The first is a set of basic environments that are available in all document types: *Theorem*, *Lemma*, *Proposition*, *Definition*, *Proof*, and *Equation*. The second and much more powerful feature is the Scribe mathematics facility, described in Chapter 11. The basic math environments all work in roughly the same way. Scribe prints the environment's text in the format defined for the environment and automatically provides whatever prefix and number is appropriate for the environment. Consider the following examples:

Manuscript Form:

```
@Begin(Itemize)
The bullets in an Itemize environment
are filled in by Scribe.  Some styles use
hyphens or asterisks instead of bullets.
@Begin(Itemize)
When you nest one Itemize inside another, the margins are
adjusted appropriately.

The switch from one kind of bullet to another is specified
in the Scribe Database file for the output device being used.
@End(Itemize)

@Begin(Multiple)
Normally, each blank line starts a new item
because each paragraph is a new item.

When you need more than one paragraph in a single item, use
@@Multiple or the @@Begin(Multiple) and
@@End(Multiple) commands.
@End(Multiple)

When you close the Multiple environment, the next paragraph
is a new item.
@End(Itemize)
```

Document Result:

- The bullets in an Itemize environment are filled in by Scribe. Some styles use hyphens or asterisks instead of bullets.
    - When you nest one Itemize inside another, the margins are adjusted appropriately.
    - The switch from one kind of bullet to another is specified in the Scribe Database file for the output device being used.
- Normally, each blank line starts a new item because each paragraph is a new item.

  When you need more than one paragraph in a single item, use @Multiple or the @Begin(Multiple) and @End(Multiple) commands.
- When you close the Multiple environment, the next paragraph is a new item.

Figure 3-2: An Exampled of the Itemize Environment

Manuscript Form:

```
@Begin(Enumerate)
The numbers in an Enumerate environment are filled in
by Scribe. Some styles use roman numerals instead of numbers.
@Begin(Enumerate)
When you nest one Enumerate inside another, the numbers and
margins are adjusted appropriately.

The switch from numbers to letters is specified in the
Scribe Database file for the output device being used.
Deeper nesting will produce other kinds of
numbering.
@End(Enumerate)

@Begin(Multiple)
Normally, each blank line starts a
new item because each paragraph is an item.

When you need more than one paragraph in a single item,
use @@Multiple or the @@Begin(Multiple) and
@@End(Multiple) commands.
@End(Multiple)

When you close the Multiple environment, the next paragraph
is a new item.
@End(Enumerate)
```

Document Result:

1. The numbers in an Enumerate environment are filled in by Scribe. Some styles use roman numerals instead of numbers.
    a. When you nest one Enumerate inside another, the numbers and margins are adjusted appropriately.
    b. The switch from numbers to letters is specified in the Scribe Database file for the output device being used. Deeper nesting will produce other kinds of numbering.
2. Normally, each blank line starts a new item because each paragraph is a new item.

    When you need more than one paragraph in a single item, use @Multiple or the @Begin(Multiple) and @End(Multiple) commands.
3. When you close the Multiple environment, the next paragraph is a new item.

Figure 3-3: An Example of the Enumerate Environment

Manuscript Form:

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

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

The act of subdividing some object into its distinct parts.]
@End(Description)
```

Document Result:

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.

The act of subdividing some object into its distinct parts.

Figure 3-4: Sample Input and Output for the Description Environment

Manuscript Form:

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

Document Result:

Theorem 1: The closed interval (a,b) is compact.

Text of the document.

Theorem 2: A closed bounded subset of R^n is compact.

Manuscript Form:

```
@Begin(Equation)
@i[x]@-[n+1] = @i[x]@-[n] - F(@i[x]@-[n])/F'(@i[x]@-[n])

  or

@Tag(Newton)@i[x]@-[n+1]=(@i[x]@-[n]+@i[c]/@i[x]@-[n])
@End(Equation)
```

Document Result:

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

or

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

In the normal document types, Theorems, Lemmas, Propositions, and Definitions are all numbered together: There will not be a Theorem 5-1 and also a Definition 5-1. Equations are numbered separately. Proofs are not numbered at all. Every Theorem, Lemma, Proposition, and Definition has a number. For equations, only those Equations that contain a cross-reference tag (@Tag command) actually receive a number. To make proper use of this numbering facility, you should use the Scribe cross-referencing mechanism, which is described in Chapter 7.

You might also find the @Blankspace command (described in Chapter 8), to be useful. @Blankspace allows you to leave space to either write in the equation by hand or paste in an equation produced by some other means.

3.7 Color Output

The use of color in ordinary text is an unfamiliar notion even today. If you look around at magazines and display advertising, you will see color used to great advantage, but look at a textbook and you will see black and white. In rare cases, color will be used in the background to highlight an example, but as a general matter color is not used in text. With the advent of affordable color graphics terminals and hardcopy devices, authors of documents now have the ability to use color in a meaningful way to prepare manuscripts and presentation slides.

Scribe provides color support for devices capable of printing in color. In other words, Scribe makes no attempt to change color on devices for which manual intervention is required; Scribe only supports changes that can be accomplished via commands included in files shipped to the device. Even though a Line Printer, for instance, will print in red if you replace the printer ribbon and an ordinary CRT terminal will display a different color if you replace the CRT itself with one having a different phosphor.

3.7.1 Using Color

Within Scribe, "Color" refers to the color of the *text* that is being printed, while "BackgroundColor" refers to the color of the *surface* on which printing occurs. A device may be able to vary either, both, or neither of these areas. You have the ability to specify color in any of those areas without being concerned about what your device can and can not do. As with everything else in Scribe, a reasonable default will be chosen to compensate for differences in device characteristics in a reasonable way. For example, on Line Printer and Robot Typewriter devices, you will get black ink on white paper. On the GIGI, you'll get white text on a blue screen. On a Concept-108 terminal, you will get white characters on a dark screen.

To change the color of the text to red, use the command @Style(Color Red). To make the background green, you would say @Style(BackgroundColor Green). See Section 4.5 for further details.

3.7.2 Coloring Pieces of Text

Frequently, you will want to have a single symbol, word, or sentence appear in a color different from that of the surrounding text. To make this change as easy as possible, Scribe provides a number of standard color environments. They are used in exactly the same way as other standard environments such as @i, @b, and @Center. The standard color environments are listed below:

Dark	**Black**	**White**
Red	**Green**	**Blue**
Yellow	**Magenta**	**Cyan**

"Dark" and "Black" are the same color, and it is the default color for most devices and document types. Be warned that Scribe's interpretation of the colors is not necessarily literal. On all CRT's, for example, Scribe considers the color of the writing to be Black, even though the screen may have a white or green phosphor.

As with all environments, the color environments can be used in either the short form

@Red[*Text to appear in red.*]

or the long form

@Begin(Red)
Text to appear in red.
@End(Red)

As was already stated, if the requested color is not available on your output device, a suitable alternative will be supplied. In the case of color output on devices which simply cannot change color, that alternative is to do nothing. You will receive no error messages, and there will be nothing special done to the text.

Different portions of the text can be colored consistently throughout the document using single-line commands. It is possible to make all page numbers come out in blue, footnotes in green, quotations in red, and so forth by using the environment attribute "Color". See the discussion of the @Modify command on page 162 to learn how to produce these changes.

Chapter Four
Simple Commands

Environments, such as those described in Chapter 3, are defined in Scribe's Database and can differ from one document type to another. Scribe also has a set of *commands*, which are imperative instructions to the Scribe program. Commands have the same meaning for all document types and all devices. In this chapter, we describe a few of the simpler commands.

4.1 Footnotes and Endnotes

Scribe places and numbers footnotes for you automatically. To get a footnote[1] into running text, simply insert an @Foot command into your text at the point where the superscripted footnote marker should appear:

```
@Foot(body of footnote)
```

Always place the @Foot command at the point where you want the superscripted footnote number to appear. The footnote in the previous sentence was generated by the following line. Notice the lack of space between the word "footnote" and the @Foot command.

```
To get a footnote@Foot(Like
this one.) into running text,
```

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

When you are generating output for devices such as File, which do not have separate pages, footnotes are converted into parenthetical notes and left in the text. This alteration is done because there is no page bottom to which the notes can be moved.

An *endnote* is a note that appears not at the bottom of the page or within the text as a parenthetical phrase, but at the end of the document. You generate endnotes with the @Note command:

```
@Note(This is the text of an endnote.)
```

Scribe assigns a number to the endnote, places the number in your text at the point where the

---

[1] Like this one.

@Note command occurs, and moves the text of the note to the end of the document, where it appears in a ''Notes'' section.

4.2 Storing and Re-Using Text: The @String and @Value Commands

The @String command provides a way to define abbreviations or the text of phrases that you might need to change. You take a string of characters, and give it a code name by using the @String command. Choosing the name is up to you; Appendix F gives the full set of rules for what is permitted in the @String command (and other commands like it). If you will limit your names to letters and use delimiters the same way we do, you don't have to read the appendix.

In the following example, the code word is on the left, and the string of characters is on the right. The string of characters must be surrounded by a pair of Scribe delimiters. Be sure to choose a delimiter pair that does not appear as part of the string!

```
@String(USPS="the United States Postal Service")
@String(CACM="@i[Communications of the ACM]")
@String(OurName="[We'll think of something to put here.]")
```

The @Value command retrieves the text string associated with a code name. It puts the string into the document at the point where the @Value command occurs in the manuscript, exactly as if you had typed the string in your manuscript. For example, suppose the following manuscript lines appear somewhere after the definitions above.

Manuscript Form:

```
In the rate increase request, @Value(USPS) alleges ...

Prior work was published in @Value(CACM).

Our new company name, @Value(Ourname),
     was carefully chosen to...
```

The resulting document text looks like these lines:

Document Result:

In the rate increase request, the United States Postal Service alleges ...

Prior work was published in *Communications of the ACM*.

Our new company name, [We'll think of something to put here later],
was carefully chosen to...

You can use the @String command to define a new string whose contents are the same as an old string by providing the name of an existing string as the value for the new string instead of providing delimited text.

Manuscript Form:

```
@String(BillingAddress="1234 Main St, Peoria")
@String(ShippingAddress=BillingAddress)

Bill to: @Value(BillingAddress)
Ship to: @Value(ShippingAddress)
```

Document Result:

Bill to: 1234 Main St, Peoria
Ship to: 1234 Main St, Peoria

You can also copy the value of a counter into a string; see Section 7.1.2 on page 66.

4.3 Using Predefined Internal Strings

The ''text string'' mechanism just described allows you to define names for text strings and retrieve their values with the @Value command. Scribe predefines some text strings for you, containing various characteristics of the Scribe run and of the document being processed.

For example, using @Value, you can obtain the name of the file being processed, when that file was created, what time the run began, and other bookkeeping information. A complete table of predefined internal strings appears in Appendix E.6. The next few pages contain some examples of how to use them.

As an example, the predefined string named Date holds the current date, so @Value(Date) produces the current date at the point in the document where that command is found. Suppose your manuscript file contains this sentence:

Manuscript Form:

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

The document would contain a sentence like the following:

Document Result:

This document was produced on 15 July 1985.

The precise format in which the date is printed is controlled by the Date parameter for the @Style command. The normal format for the date is like the one above: ''15 July 1985''. However, you can change the date style (at any point in the file) so that @Value(Date) produces almost any format at all. To make this change, you give the Date parameter a value that represents the date March 8, 1952, in any format that you want. (The date March 8, 1952 *must* be used in the @Style command.) Scribe analyzes your string, figures out the format that you have used to specify the date March 8, 1952, and uses that format for the current date. You can use nearly any notation; month and day names can be in English, Spanish, German, or French. Scribe does not, however, recognize numerical dates (for example, eighth, ninth, tenth) in other languages. Some examples are shown below:

Manuscript Form:

```
@Style(Date="03/08/52")
Today is @Value(Date).

@Style(Date="8th of March, 1952")
Today is @Value(Date).

@Style(Date="Saturday, 8th of March")
Today is @Value(Date).

@Style(Date="08-MAR-52")
Today is @Value(Date).

@Style(Date="the eighth day of March, nineteen fifty-two")
Today is @Value(Date).

@Style(Date="Samedi le 8@+[e] Mars, 1952")
Aujourd'hui c'est @Value(Date)

@Style(Date="8 Maerz 1952")
Heute ist der @Value(Date)
```

Document Result:

Today is 07/15/85.

Today is 15th of July, 1985.

Today is Monday, 15th of July.

Today is 15-JUL-85.

Today is the fifteenth day of July, nineteen eighty-five.

Aujourd'hui c'est Lundi le 15^{e} Juillet, 1985

Heute ist der 15 Juli 1985

@Value(Weekday) gives you the name of the current day of the week, for example, "Monday".

@Value(Time) tells you the time, to the nearest minute, when the current Scribe run began; for example, "15:07". This time is normally a 24-hour time, but you can specify a printing style for the time. The method is the same as for the date style. Use the Time parameter for the @Style command to provide a sample format for the time 4:30 p.m. (Again, as with the Date specification, a certain time *must* be used: 4:30 p.m.) Scribe then analyzes the format you used to represent 4:30 p.m., figures out the format that you have used to specify the time 4:30 p.m., and applies that format to printing the current time string.

@Value(TimeStamp) produces both the date and the time when Scribe began processing your .MSS file. Its default style is "15 July 85 15:07". You can change the style in which timestamps are printed by specifying a value for the TimeStamp parameter in an @Style command. Use the same scheme as for Date and Time, with the same standard date and time (March 8, 1952 and 4:30 p.m.).

Scribe provides three ways for you to inquire about the files that it is working on. @Value(Manuscript) tells you the name of the manuscript file that Scribe is processing, for example, "USER1.MSS". @Value(FullManuscript) tells you the name of the manuscript file that Scribe is processing, complete with device and directory specification, for example, "DSKC:USER1.MSS[SMITH]". @Value(SourceFile) tells you the particular place in the manuscript file or the included subfile that Scribe is processing at that moment, for example, "COMAND.UM1, 28400/1". @Value(FileDate) tells you the date and time that the current manuscript file was last updated, for example, "7 June 1985 at 14:38". If you are using multiple files to hold your manuscript (see Chapter 13), then @Value(RootFileDate) tells you the date and time that the root file was last updated, for example, "19 June 1985 at 12:02". The format in which both @Value(FileDate) and @Value(RootFileDate) are printed can be specified by using the FileDate parameter for an @Style command. Use the same scheme (and the same standard date and time) as described above for Date and Time.

@Value(ScribeVersion) tells you the version of Scribe that is doing the processing, for example, "4(1400)", and @Value(UserName), for example "Smith", gives you the name of the current user according to the local operating system.

Several predefined strings contain information about components of the document. These are useful in @PageHeading commands. @Value(Page) tells you the current page number, for example, 35. If you are producing a document with a Table of Contents, then you can retrieve the number or title of the current section. @Value(SectionTitle) gives you the title of the most recent section command (chapter, section, subsection, or whatever is appropriate), for example, "Using Predefined Internal Strings". (Go back and look at the heading at the beginning of this section (4.3) on page 33.) @Value(SectionNumber) gives you its number, for example, 4.3.

4.4 Simple 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 it. Scribe provides two indexing commands that will produce a single-level index entry, which has the Index term and the reference number. There are two other commands that produce a multiple-level Index, but they are not explained in this chapter. Refer to page 172 in Section 15.6.1 for details on those commands.

4.4.1 The @Index Command

The @Index command is the simplest indexing command. All that you need to specify when using it is the Index term. Scribe fills in the correct page number and puts the entry in alphabetical order in the Index according to the collating sequence used on your computer. The syntax of the command is

@Index (*index-entry*)

where *index-entry* is the text you want to appear in the Index. You can put the command at any point in the .MSS file. @Index works like @Foot in that it doesn't interrupt the sequence of the text. Nothing appears in the running text of the output at that point, but the entry appears in the Index with the correct page number. This manuscript form would produce the document result shown here, although the result would be in the Index, of course, and not in the running text:

Manuscript Form:

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

Document Result:

Text to be indexed 36

The @Index command can be on the same line as text or on a line by itself in the manuscript file. If it is on its own line, Scribe ignores the carriage return that follows it. The @Index command should be tied as closely as possible to the term in the manuscript file that it is referencing to ensure correct reference numbers in the Index. In a filled environment, you cannot be certain where the page breaks in the output will be, and the further the @Index command is from the appropriate term in the manuscript file, the better the chances that there will be a page break between them and that the Index number will be one page off. Consider the following example, which shows the true document result — the entries are actually in the Index.

Manuscript Form:

```
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.
```

Document Result:

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 can consult the Index of this manual (under "Apples", "Peaches", and "Wine") to see how those three entries look.

Only text should be put in the @Index command. If you include any Scribe command or environment in the text to be indexed, you will get unexpected and unsatisfactory results. For specialized formatting in Index entries, use the @IndexEntry form.

4.4.2 The @IndexEntry Form

The @Index command allows you to put simple text strings into the Index. As you just learned, Scribe will add a page number to each entry, sort them into alphabetical order, and print them. One drawback to this command, however, is that any Scribe commands or environments that are contained in it are processed *after* the entry is sorted alphabetically; therefore, not all entries made via the @Index command are alphabetized as you would expect. Consider this example. The following command is included in a .MSS file:

```
@Index[@i(Scribe User Manual)]
```

The result would be an entry alphabetized by "@". The @IndexEntry form allows you to get around this problem and have that entry listed under "S".

A form is an advanced Scribe construct roughly analogous to a command in its invocation. Environments contain text; forms contain delimited strings as parameters. While a complete description of forms is contained in the *Scribe Database Administrator's Guide*, you don't need to understand that material to use the @IndexEntry form. Just follow the instructions in this section.

The format of the @IndexEntry form is

```
@IndexEntry[Key="text-to-be-alphabetized-as",
            Entry="text-to-be-indexed",
            Number="reference-number"]
```

Each of the parameters is described below. Two of the three are required.

Key The sort key. Required. This text will be used to determine the alphabetic position of the Index entry.

Entry The text of the entry. Required. This text will not participate in the sorting, but it will be used to test for the merging of adjacent entries in the Index.

Number The reference number indicator. Optional. This parameter determines, by its absence or presence, whether a reference number will be assigned to the entry and, if one will be assigned, what that number will be. If it is missing, then the entry will not be given a number. If it is present but has no value, then the current page number will be used. If it is present and has an argument, then that argument will be used to number the entry.

A typical @IndexEntry form and its resulting output is

Manuscript Form:

```
@IndexEntry(Key="Joy of Cooking",Entry="@i[Joy of Cooking]",
            Number)
```

Document Result:

Joy of Cooking 37

@IndexEntry also gives you the freedom to alphabetize random text at any point in the Index. Since you supply the sort key and the text to be indexed separately, they need not be related in any particular way. "Fancy Indexing" can be alphabetized as though the entry was "Misleading Entries" by this line:

```
@IndexEntry[Key=<Misleading Entries>,
            Entry=<Fancy Indexing>, Number]
```

If you look in this manual's Index under ''M'', you'll see that entry.

All of the parameters for the @IndexEntry form can have either delimited strings or @String names as a value. If, for example, the **Key** parameter is given the value [**ABC**] (a delimited string), then the sort key will be the literal text ''**ABC**''. If it is given the value **ABC** (no delimiters), then the value of the string named **ABC** and defined with the @String command will be used. In other words, the sort key will not be ''**ABC**'', but rather the contents of a defined string named **ABC**.

Consider this situation. A string **dbag** is defined this way:

```
@String[dbag="@i(Scribe Database Administrator's Guide)"]
```

and the following @IndexEntry form is included in the manuscript file:

Manuscript Form:

```
@IndexEntry[Key="Scribe", Entry=dbag, Number]
```

The resulting Index entry would be this line:

Document Result:

Scribe Database Administrator's Guide 38

Let's take that same example and extend it a bit to include several examples of what the @IndexEntry form can do.

Manuscript Form:

```
@IndexEntry[Key="Scribe User Manual",
        Entry="@i(Scribe User Manual)"]
@IndexEntry[Key="Documentation",
        Entry="@i(Pocket Reference)", Number=17]
@IndexEntry[Key="Scribe",
        Entry=dbag, Number]
```

The resulting Index entries would look like these entries:

Document Result:

Pocket Reference 17
Scribe Database Administrator's Guide 38
Scribe User Manual

4.5 Adjusting Document Formats: The @Style Command

The @Style command gives you the ability to change some aspects of the appearance of a document. For example, suppose you want to use the Text document type, which is normally single-spaced, and, for some reason, you need a double-spaced document. All is not lost. You need not spend days trying to understand how to design your own document type. Use an @Style command with the parameter and value you need to make the change. In this case, you would use @Style(Spacing 2).

The @Style command provides parameters for many aspects of document appearance, for example, indenting, spacing, fonts, justifying, footnote numbering, etc. These aspects all have values predefined in the Database for the document type you are using. However, by placing an @Style command at the beginning of your manuscript, you can specify values that override the ones in the Database. Any @Style command that changes the overall appearance of the document has to come at the beginning of the .MSS file. You cannot change overall styles in mid-document.

Some changes are too complicated to be effected with simple @Style commands. If you really must have changes beyond those that the @Style mechanism provides, the @Modify and @Define commands to do these things are documented in Chapter 15 and in the *Scribe Database Administrator's Guide.* But don't even *think* about trying modifications of this magnitude until you have become familiar with the basic system.

Let's consider some examples. Suppose you want your document to have the properties listed below.

Printed in red:[2]

```
@Style(Color Red)
```

A five character indent instead of none:

```
@Style(Indent 5 Characters)
```

A ragged right margin instead of a justified one:

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

A specific text body width:

```
@Style(LineWidth 6.7inches)
```

Footnotes flagged with superscripted asterisks instead of numbers:

```
@Style(Footnotes "@@+[@*]")
```

Pages numbered with the word "Page" included:

```
@Style(PageNumber "Page @1")
```

Appendix E.7 and the *Scribe Pocket Reference* contain a complete list of @Style parameters. We include a list of the more common ones here with a brief description for each.

---

[2] Remember that true color output is only available on some output devices and that the color produced, since it depends on the capabilities of the printer, may not be the color you request. See Section 3.7 for more details.

BackgroundColor Specifies what color the background of the document should be. Takes a value from the set {Dark, Red, Yellow, Black, Green, Magenta, White, Blue, Cyan}. Other values may be available for your site. Check with your ***DBA*** for details.

BottomMargin Specifies the bottom margin of the page — the distance between the last line of text and the bottom edge of the paper. Takes a vertical distance as a value. PaperLength, TopMargin, and BottomMargin all affect the length of the printing area on the page.

Color Specifies what color the text of the document should be. Takes a value from the set {Dark, Red, Yellow, Black, Green, Magenta, White, Blue, Cyan}. Other values may be available for your site. Check with your ***DBA*** for details.

FontFamily Specifies the FontFamily for the document. Takes a FontFamily name as an example. For example, @Style(FontFamily <Helvetica10>) or @Style(FontFamily <TimesRoman8>). Not every output device is capable of changing fonts; not every font is available even on those output devices that can change fonts. Consult with your ***DBA*** to find out what fonts are available on your printing device.

Footnotes Specifies the numbering style to use for marking footnotes in the text and in the footnote itself. Takes a numbering template as a value. (See Section 15.8 on numbering templates.)

Indent Specifies how much space to indent the first line of each paragraph in the text. Takes a horizontal distance as a value. Negative indent values are valid for *hanging indents* (''outdented paragraphs'').

Justification Specifies whether the right margin is printed ''ragged'' or aligned. Takes a Boolean value. Most document types have justified right margins. The following line creates a ragged right margin: @Style(Justification Off).

LeftMargin Specifies the width of the overall left margin on the page. Takes a horizontal distance as a value. The margin is really the white border along the edge, so a bigger number means more white space and a shorter print line.

LineWidth Specifies the width of a print line. Takes a horizontal distance as a value. LeftMargin, RightMargin, and LineWidth all provide means for controlling the line layout. You can specify any two of the three, but not all three of them.

PageNumber Specifies the style of the page number. Takes a numbering template as a value. (See Section 15.8 on numbering templates.) Most documents are defined to be numbered sequentially, beginning at 1, but you may change that number style with this @Style parameter.

Singlesided Specifies that the printing of the document is to be on one side only. Takes a Boolean value. Some document types are set up for double-sided reproduction, which means that chapters are set to begin on a right-hand (odd) page, and the left and right margins are varied from even to odd page to make room for the binding. If you don't want these effects, specify @Style(Singlesided).

RightMargin Specifies the width of the overall right margin on the page. Takes a horizontal distance as a value. The margin is the white border along the edge, so a larger right margin means a shorter print line.

TopMargin Specifies the amount of white space between the top of the paper and the first print line on the page. Takes a vertical distance as a value.

4.6 Page Headings, Page Footings, and Page Numbers

The information printed at the top or bottom of each page is called a *page heading* or *page footing*, respectively. One of the most common items in a heading or footing is the page number. Normally, Scribe numbers the pages for you, putting a page number centered at the top of all pages after the first. This convention is the default page heading supplied by most document types. You can 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 that set the page heading and page footing for this page are shown here:

```
@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 delimiters (the delimiters are quotation marks in this example) can contain any manuscript-file text, including Scribe commands. Several Scribe commands provide information useful for including in running heads. Use @Value(Page) to get the current page number. Use @Value(Date) for the current date. The following command would put the words ''Reference Manual'' in the top left corner, the current date in the top center, and the page number in the top right corner:

```
@Pageheading(Left "Reference Manual",
        Center <@Value(Date)>,
        Right <@Value(Page)>)
```

(Section 4.3 tells you more about the @Value command.)

Not all of the three parameters ''Left, Center, Right'' need to be specified in an @Pageheading or @Pagefooting command. However, Scribe assumes that if you do not specify a parameter, you do not want any text in that position. For example, let's say that the first 10 chapters of a manual were to have the page number in the left corner of the page. The Scribe command for that output is

```
@Pageheading(Left "@Value[Page]")
```

If you wanted to change the page heading beginning with Chapter 11 so that the title of the manual was in the right corner of the page, then the command necessary is this one:

```
@Pageheading(Left "@Value[Page]", Right "Manual Title")
```

If you had not repeated the ''Left'' setting, you would have only received text in the right corner of your pages.

You can define page headings and footings at the beginning of the manuscript and at any

point in the manuscript. Normally, when Scribe see an @Pageheading or @Pagefooting command, it continues processing the .MSS file to complete the output page that it is currently creating and then produces the specified heading or footing on the next output page. However, the @Pageheading and @Pagefooting commands can include an argument, Immediate. If the Immediate argument is present, then the heading or footing changes on the current page. In that case, Scribe stops processing text and redraws the page heading or footing for the current output page.

```
@Pageheading(Immediate,Left "Left heading",...)
```

In the absence of the Immediate argument, the newly-declared heading or footing takes effect on the following page.

You can 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 might want odd and even headings to be the mirror image of one another. Use two different @Pageheading commands, one containing the parameter Odd and the other containing the parameter Even. The commands that generated the page headings for this manual are:

```
@Pageheading(Even,Right "@c[Scribe User Manual]",
                    Left "@c[@Value(Page)]")
@Pageheading(Odd,Left "@c[@Title(Chapter)]",
                   Right "@c[@Value(Page)]")
```

(The @Title command is described in Section 7.1.1).

For even and odd page headings to work properly, the document must be defined as doublesided. Some document types are defined to be doublesided by default for certain output devices. If you want a document that is normally singlesided to be doublesided so that the even and odd page headings print properly, then included the @Style(DoubleSided On) command in your .MSS file. If an even and odd page heading is defined for a singlesided document, Scribe will not produce any error message, but the odd page heading will be used throughout the document.

Since not all of the three @Pageheading command parameters (Left, Center, Right) need to be mentioned and if one is missing, no text is inserted in that position, erasing the page heading for a document is simple. Just include this command at the top of your manuscript file:

```
@Pageheading()
```

No page headings will appear in the document beyond the first page. ''Immediate'' can be included in that command and then no page heading will be printed on any page, including page one:

```
@Pageheading(Immediate)
```

4.7 Comments in Manuscript Files

You can put text in the manuscript file that you do not want to come out in the finished document by using the Comment command. Scribe totally ignores any text that you specify with the Comment command, no matter what it contains.

Comment does not let you nest delimiters. This restriction is necessary because Scribe does not process any commands or environments inside Comment, so it cannot know about their delimiters. In the following example, the word "Typewriter" will appear in the document even though it was not intended to because the first right-curly-bracket character acts as the closing delimiter for the comment.

Manuscript Form:

```
@Comment{He bought a @i{Royal} typewriter.}
```

Document Result:

typewriter.}

It is legal to have Scribe commands within the body of a comment. However, they will not be processed. Just be sure that none of them uses the same delimiters as the Comment command:

```
@Comment{He bought a @i[Royal] typewriter.}
```

The @Begin/@End construction may be used with comments, allowing you to avoid delimiter problems completely.

```
@Begin{Comment}
He bought a @i{Royal} typewriter.
@End{Comment}
```

4.8 Special Characters

A computer terminal keyboard normally has only 95 characters on it. People frequently want to include special characters, whether a Greek letter like Π or ζ or a mathematical symbol like $\equiv$ or $\leq$.

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 ask Scribe to leave space for you to write in the characters by hand.

Do not try to get special characters by putting control codes into your .MSS file. Scribe will have no idea what the widths of the characters are supposed to be and will be unable to format your text properly.

4.8.1 Printing Predefined Special Characters

Scribe lets you specify special characters for printing by pretending that they are a font change; for example, the Greek character Alpha is simply the "Greek Font" representation of the letter "a". Each printing device provides some means of printing the special characters, though the means may be different on different devices. The method for representing Greek characters is as follows:

Manuscript Form:

```
if A < B then @g(l) must be 0
```

Document Result:

if A < B then λ must be 0

Font-change codes were described in Section 3.2. A full table of Greek letters available with @g appears in Appendix C on page 190.

Printing mathematical characters is accomplished via the Scribe mathematical facility described in detail in Chapter 11.

On some computer systems, the non-printing (control) characters on your keyboard can be used to specify special print characters. This use of control characters is not recommended, as there is no standardization from one printing device to the next and from one site to the next. Nevertheless, Scribe does process otherwise unassigned control codes as text characters when it finds them in your manuscript file.

4.8.2 Faking Special Characters

Scribe's internal information about printing devices includes knowledge about what characters and fonts it can print. Scribe automatically leaves a blank space when it encounters a character that it can't print on a device, allowing you to write the character in by hand.

Scribe also has a command that allows you to request blank space for a special character. You can use it if you don't want to learn the special characters or if you need Chinese characters or something else that really isn't available.

The @# command leaves a blank space large enough to write one character. (In printer's terminology, the space it leaves is called an *em-space* or a *quad space*.)

Suppose your manuscript file contains the following words:

Manuscript Form:

```
Kaiser Wilhelmstra@#e
```

Scribe replaces @# with an em-space (or the closest equivalent on your device):

Document Result:

Kaiser Wilhelmstra e

In addition, Scribe makes a note of the page and line number in the error log file (see page 147). This record means that when you take pen in hand to write in the missing character, you can find the right spot in the document.

Document Result:

Kaiser Wilhelmstraβe

You can request blank space either with the @# or the @␣ (an @-sign followed by a space; see page 102) command. The @# mechanism makes an entry to a list of special characters for each @#, whereas the @␣ command does not. Choose whatever mechanism is appropriate.

Chapter Five
Organizing Manuscript Files

We've told you about a lot of commands but haven't said much about how and when to use them. In this chapter, we discuss the overall structure and sequence of commands in your manuscript file.

5.1 Document Types

Each time Scribe produces a document, it must know two things:

1. What kind of document are you making?
2. What is the printing device?

These two items of information are called the *Document type* and the *Device type*. We already showed you how to tell Scribe about printing devices (Section 2.4, page 8). To tell it about document types, you need to put a command at the beginning of your file, before any of the text (see Section 5.2). The command name is @Make, and you use it by putting the name of the kind of document to make inside delimiters:

```
@Make(Letter)
@Make "Manual"
```

The name Letter tells Scribe that you are producing a document whose name is Letter. Scribe turns to its Database, finds the definition of Letter, and discovers that letters consist of a return address, an inside address, a greeting, a body, a closing, a signature, and possibly some notations. It expects you to provide the text for some or all of those various pieces.

Had you begun your manuscript file with the command @Make(Manual), then Scribe would have consulted its Database to discover that a Manual consists of a title page, a Table of Contents, possibly a Preface and/or an Introduction, then a series of Chapters, then possibly some Appendices, and finally an Index. You are responsible for providing those pieces, in the right order and properly labelled.

So the details of document organization depend on what you are doing. Any general rules are necessarily vague. We can, however, give you some guidelines and some specific examples.

5.2 Which Commands Go Where?

It is customary, but by no means required, to have the first two commands in the manuscript file be the @Device and @Make command. The only firm rule for the sequence of commands is that all commands that change Scribe's overall processing must come before the first text. For example, @Style commands defining the overall appearance of a document must appear in the manuscript file before any text in the document.

We often say things like, "Command thus-and-so must be at the beginning of the file". By "beginning", we mean the group of Scribe commands that comes before the first text of the document and not literally the first line of the file. Please note that the @Include command pulls text into the document and so is itself considered text. Therefore, no Scribe command that alters the document as a whole may be inserted after either text or an @Include command. (See Section 13.1 for more information about this command. If you've never heard of it, don't worry; it won't affect you.)

If you make a mistake and put some commands to change things after the first text in the file, Scribe normally prints a warning message reminding you of your mistake, ignores the misplaced command, and does what was initially defined in the .MAK file (the document definition file), the .DEV file (the device definition file), or the beginning of the document. (See Chapter 14 for an explanation of Scribe messages.)

5.3 Details of Particular Document Types

Many installations using Scribe have their own customized document types, so it's not possible to discuss all of the document types on your system in this manual. In this section, we tell you how to use the basic document types and formats that are part of Scribe when it is distributed.

Some installations support additional variant forms of the regular document types. A variant form is one that has the same set of pieces and uses the same commands, but produces a different output format. You request a variant form by adding a parameter to the @Make command:

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

If your installation has a Form-1 Article definition in its Database, then Scribe uses that alternate form for your document. (Find out what is available by contacting your ***DBA***.)

Some document types have optional parameters; you specify these parameters in exactly the same way that you specify a variant form. For example,

```
@Make(LetterHead,Phone "(412) 281-5959")
```

(Again, consult your ***DBA*** or documentation for details.)

5.3.1 The Text Document Type

The default document type (the one you get when you don't specify any document type in your .MSS file) is called Text. If you don't put an @Make command in your file, Scribe acts as if you had used @Make(Text). This default format is very simple; it has no Table of

Contents, no numbered sections, no figures or tables, no title page. It is intended to be used for simple or small documents with no fancy characteristics.

All of the environment types described in Chapter 3 can be used in the Text document type. Since Text has no chapters, the Theorem and Equation environments are numbered sequentially rather than within chapters.

The standard Text document type does not indent its paragraphs. If you want indented paragraphs, use an @Style command to set a value for indenting (see @Style in Section 4.5, beginning on page 39). For example, the following command requests that the first line of every paragraph be indented 3 spaces.

```
@Style(Indent 3)
```

Figure 5-1 is a short example of a complete manuscript file that uses document type Text. Figures 5-2 through 5-4 show the Scribe output of this file on different output devices. The .MSS file uses some commands that we haven't discussed yet — namely the @> and @\ (flush right and tab, respectively) commands (they are described in Section 9.1.2, beginning on page 82, but the format of document type Text is still clearly shown in the figures.

```
@Heading(15-211: Fundamental Structures of Computer Science
Homework Assignment 5)
@Begin(Format)
@TabDivide(3)
Assigned:@>15 October@\@\Instructor:@>Bill Wulf
Due:@>1 November@\@\T.A.:@>Brian Reid
@End(Format)

We have been studying the data structures known as
@i[graphs] and @i[trees].  To a genealogist, a family tree
is a chart showing a group of people and how they are
related to one another.  There are two kinds of links:
marriage links and offspring links.
@Begin(Enumerate)
Design a data structure that can be used to represent a
simple family tree.  By "simple", we mean that each person
marries only once, and that no person in the tree marries
incestuously.  Hint: Represent each person with a record;
each record would have a @i[spouse] field, a @i[parent]
field, etc.

Modern families are not so simple.  Family trees are often
family DAGs.  People divorce and remarry.  Revise your data
structure to be able to handle multiple marriages.  Suppose
that a person marries his first cousin.  Will your data
structure be able to handle this?

Write a Pascal program that will build this data structure.
Its input should be a series of lines identifying events:
"John marries Susan" or "Bill is born to John and
Susan".  Feel free to devise your own syntax to make the
input processing as easy as possible.

Write a relationship-finding program.  This program will
tell you the relationship, in English, between any two
people in the graph.  For example, if you give it names
"John" and "Mary", it might print out "John is Mary's
father" or "John is Mary's 3rd cousin 2 times removed."
@Foot<Hint: To find the relationship between nodes @i[A] and
@i[B], find the nearest common ancestor, say @i[C].  Let X
be the number of generations between @i[A] and @i[C], and Y
be the number of generations between @i[B] and @i[C].  If X
is equal to Y, then A and B are (X-1)@+[th] cousins.  If X
is not equal to Y, then without loss of generality assume X
less than Y; @i[A] and @i[B] are then (X-1)@+[th] cousins
@w[Y-X] times removed.>  Some relationships have special
names: a zero'th cousin is a brother or sister, a first
cousin once removed is an aunt, uncle, nephew, or niece.
@End(Enumerate)
```

Figure 5-1: An Example of a Manuscript File in the Text Document Type

15-211: FUNDAMENTAL STRUCTURES OF COMPUTER SCIENCE
HOMEWORK ASSIGNMENT 5

ssigned: 15 October Instructor: Bill Wulf
ue: 1 November T.A.: Brian Reid

e have been studying the data structures known as graphs and trees. To genealogist, a family tree is a chart showing a group of people and ow they are related to one another. There are two kinds of links: arriage links and offspring links.

1. Design a data structure that can be used to represent a simple family tree. By ``simple'', we mean that each person marries only once, and that no person in the tree marries incestuously. Hint: Represent each person with a record; each record would have a spouse field, a parent field, etc.

2. Modern families are not so simple. Family trees are often family DAGs. People divorce and remarry. Revise your data structure to be able to handle multiple marriages. Suppose that a person marries his first cousin. Will your data structure be able to handle this?

3. Write a Pascal program that will build this data structure. Its input should be a series of lines identifying events: ``John marries Susan'' or ``Bill is born to John and Susan''. Feel free to devise your own syntax to make the input processing as easy as possible.

4. Write a relationship-finding program. This program will tell you the relationship, in English, between any two people in the graph. For example, if you give it names ``John'' and ``Mary'', it might print out ``John is Mary's father'' or ``John is Mary's 3rd cousin 2 times removed.''[1] Some relationships have special names: a zero'th cousin is a

---

[1] Hint: To find the relationship between nodes A and B, find the earest common ancestor, say C. Let X be the number of generations etween A and C, and Y be the number of generations between B and C. If is equal to Y, then A and B are (X-1)th cousins. If X is not equal to , then without loss of generality assume X less than Y; A and B are hen (X-1)th cousins Y-X times removed.

Figure 5-2: Sample Output on a Line Printer (Device LPT)

15-211: FUNDAMENTAL STRUCTURES OF COMPUTER SCIENCE
HOMEWORK ASSIGNMENT 5

Assigned:	15 October	Instructor:	Bill Wulf
Due:	1 November	T.A.:	Brian Reid

We have been studying the data structures known as graphs and trees. To a genealogist, a family tree is a chart showing a group of people and how they are related to one another. There are two kinds of links: marriage links and offspring links.

1. Design a data structure that can be used to represent a simple family tree. By "simple", we mean that each person marries only once, and that no person in the tree marries incestuously. Hint: Represent each person with a record; each record would have a spouse field, a parent field, etc.

2. Modern families are not so simple. Family trees are often family DAGs. People divorce and remarry. Revise your data structure to be able to handle multiple marriages. Suppose that a person marries his first cousin. Will your data structure be able to handle this?

3. Write a Pascal program that will build this data structure. Its input should be a series of lines identifying events: "John marries Susan" or "Bill is born to John and Susan". Feel free to devise your own syntax to make the input processing as easy as possible.

4. Write a relationship-finding program. This program will tell you the relationship, in English, between any two people in the graph. For example, if you give it names "John" and "Mary", it might print out "John is Mary's father" or "John is Mary's 3rd cousin 2 times removed."[1] Some relationships have special names: a zero'th cousin is a brother or sister, a first cousin once removed is an aunt, uncle, nephew, or niece.

---

[1]Hint: To find the relationship between nodes A and B, find the nearest common ancestor, say C. Let X be the number of generations between A and C, and Y be the number of generations between B and C. If X is equal to Y, then A and B are $(X-1)^{th}$ cousins. If X is not equal to Y, then without loss of generality assume X less than Y; A and B are then $(X-1)^{th}$ cousins Y-X times removed.

Figure 5-3: Sample Output on a Diablo HyType II (Device DIABLO)

15-211: Fundamental Structures of Computer Science
Homework Assignment 5

Assigned:	15 October	Instructor:	Bill Wulf
Due:	1 November	T.A.:	Brian Reid

We have been studying the data structures known as graphs and trees. To a genealogist, a family tree is a chart showing a group of people and how they are related to one another. There are two kinds of links: marriage links and offspring links.

1. Design a data structure that can be used to represent a simple family tree. By "simple", we mean that each person marries only once, and that no person in the tree marries incestuously. Hint: Represent each person with a record; each record would have a spouse field, a parent field, etc.

2. Modern families are not so simple. Family trees are often family DAGs. People divorce and remarry. Revise your data structure to be able to handle multiple marriages. Suppose that a person marries his first cousin. Will your data structure be able to handle this?

3. Write a Pascal program that will build this data structure. Its input should be a series of lines identifying events: "John marries Susan" or "Bill is born to John and Susan". Feel free to devise your own syntax to make the input processing as easy as possible.

4. Write a relationship-finding program. This program will tell you the relationship, in English, between any two people in the graph. For example, if you give it names "John" and "Mary", it might print out "John is Mary's father" or "John is Mary's 3rd cousin 2 times removed."[1] Some relationships have special names: a zero'th cousin is a brother or sister, a first cousin once removed is an aunt, uncle, nephew, or niece.

---

[1] Hint: To find the relationship between nodes A and B, find the nearest common ancestor, say C. Let X be the number of generations between A and C, and Y be the number of generations between B and C. If X is equal to Y, then A and B are $(X-1)^{th}$ cousins. If X is not equal to Y, then without loss of generality assume X less than Y; A and B are then $(X-1)^{th}$ cousins Y-X times removed.

Figure 5-4: Sample Output on a Xerox 2700 Laser Printer (Device X2700)

5.3.2 The Letter and LetterHead Document Types

The document type Letter is designed for making standard personal letters. The document type LetterHead is designed for making business letters on company stationery. Since every installation using Scribe has its own format for company stationery, we suggest that you contact your ***DBA*** for the details of LetterHead at your site.

Most letters are composed of several components. Some personal letters might have just a greeting, a body, and a closing; most business letters contain all of the following components:

- A return address
- An inside address (of the person receiving the letter)
- A greeting
- The body of the letter
- A closing
- A signature
- Notations, usually in the lower left corner

To make a letter with Scribe, you put the text for each component in an appropriate environment, in the right sequence. See Figure 5-5 for a sample personal letter. (Consult your ***DBA*** for the names of the environments defined for Letter at your site.)

5.3.3 The Sectioned Document Types: Thesis, Report, Article, and Manual

Sectioned documents are those consisting of major logical sections. The real difference between Text and a sectioned document is that sectioned documents can produce title pages, numbered sections, and automatic tables of contents.

Figure 5-6 shows the general organization of commands in a sectioned document. In this case, the document type is Thesis and the device type is Diablo. This thesis has a title page, two prefatory sections, two chapters, and an appendix.

The details of title page environments are in Section 6.3, beginning on page 62. The sectioning commands @Chapter, @Section, and so forth are described in Section 6.2, beginning on page 60.

Manuscript Form:

```
@Make(Letter)
6937 Penn Avenue
Pittsburgh, PA  15208

@Value(Date)

@Begin(Address)
Mr. John Jones
Subscription Manager
@i[Pittsburgh] Magazine
6023 Fifth Ave
Pittsburgh, PA  15213
@End(Address)

@Begin(Body)
@Greeting(Dear Mr. Jones:)
Several months ago, in response to a WQED
fund-raising drive, I sent in a check for $25.  It
was my understanding that this contribution would
also entitle me to a subscription to
@i[Pittsburgh] magazine.  I have not yet received
my first copy.  Could you please try to find out
why?
@End(Body)
Sincerely,


Brian K. Reid
```

Figure 5-5: Sample Manuscript File for a Personal Letter

Manuscript Form:

```
@Make(Thesis)
@Device(Diablo)

@Begin(TitlePage)
    title page text goes here
@End(TitlePage)

@PrefaceSection(Foreword)
    text of Foreword

@Unnumbered(Preface)
    text of Preface

@Chapter(First Chapter Title)
    text of the first Chapter

@Section(First Section)
    text of the first Section

@Section(Second Section)
    text of the second Section

@Section(Third Section)
    text of the third Section

@Chapter(Another Chapter)
    text of another Chapter

@Appendix(The Appendix)

@Section(First Section)
    text of the first section of the Appendix

@Section(Second Section)
    text of the second section of the Appendix
```

Figure 5-6: The Skeleton of Manuscript File for a Thesis

Note

You have now completed the introduction to "straightforward Scribe". With what you have read so far, you now know enough to produce documents with reasonably complex requirements.

We suggest that at this point you review these chapters and try out some of the examples to help establish the basic concepts in your mind; we further suggest that you not try to use any of the advanced commands until you are comfortable with the basic ones.

Chapter Six
Titles, 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. This chapter explains how to get various kinds of titles and headings.

As usual, we'll start with some terminology. A document can have many *headings*. Sometimes you need to put a few *subheadings* inside a heading. When you give numbers to headings or subheadings, then they become *sections* or *subsections*. At the top and bottom of every page are the *page headings* and *page footings*, respectively, 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 need 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 document you are producing.

6.1 When There is No Table of Contents

A simple set of *heading environments* produces headings for simple document types (ones 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.

MajorHeading prints large-size letters, boldfaced and centered.

Heading prints medium-size letters, boldfaced and centered.

SubHeading prints normal-size letters, boldfaced and flush to the left margin.

(Of course, on different printing devices, these headings might be quite different in appearance from the ones you see here.)

A simple example follows.

Manuscript Form:

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

Document Result:

This is a Major Heading

This is a Heading

This is a SubHeading

That's all there is to it. Since these particular heading commands are really just environment names, you can use the "long form" specification for them also. The following manuscript form produces the same results as the previous example.

```
@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)
```

6.2 When There is a Table of Contents

A set of *sectioning commands* produces headings for documents that need Tables of Contents. The sectioning commands are different from the heading commands of the previous section because they must both print the headings in the text and make an entry in the Table of Contents. Usually these commands automatically number the chapters and sections, although in some document types they do not.

For the purposes of this discussion, we are going to describe the general case, in which parts of the document are numbered. Almost everything we describe applies to the unnumbered documents, except for the presence of numbers.

When you use any of these sectioning commands, Scribe prints the appropriate section number followed by the title you specify in the sectioning command. The appearance of the title (placement on the page, capitalization, font, and so on) depends on how the document type that you are using has defined that kind of heading.

The sectioning commands available for any document type form a *hierarchy*. Scribe assumes that one particular kind of heading is the most general and that the others are numbered within that one. You are probably already familiar with this kind of numbering scheme. (This manual uses it.)

The standard sectioning commands in Scribe have names chosen to be mnemonic for the position of that section within the hierarchy and the level of number that appears for the section title. The names of the levels in the hierarchy are Chapter, Section, Subsection, and Paragraph (or Subsubsection). Remember that these names are just reminders for the kind of numbering you get; it doesn't mean that you have to think of something as being a chapter just because you use an @Chapter command to number it. (Although the @Chapter and @Appendix commands print out the words "Chapter" and "Appendix" for a few document types.)

Different document types have different depths of numbered section hierarchies. Report and Manual have a four-level hierarchy; Article (usually with less complex heading requirements) has a three-level hierarchy. Table 6-1 shows what level of section numbering you get using the sectioning commands for the various document types. There are two other section commands available to all document types that do not produce numbers. They are shown in Table 6-2.

		Commands	
Level	Numbering	Article	Report and Manual
1	1.	@Section	@Chapter
2	1.1.	@Subsection	@Section
3	1.1.1.	@Paragraph	@Subsection
4	1.1.1.1.		@Paragraph
1	A.	@Appendix	@Appendix
2	A.1.	@Section	@Section

Table 6-1: Results of Sectioning Commands

Command	*Purpose*
@PrefaceSection(Title)	Marks the beginning of a document segment that does not have a number and whose title does not appear in the table of contents.
@UnNumbered(Title)	Marks the beginning of a major section for which you do not want a number. Its title, which usually looks just like a chapter title, does appear in the table of contents.

Table 6-2: Other Sectioning Commands

A sectioning command marks where a new section starts in a document. In addition, the sectioning command specifies a title for the new section. For example, the following command appeared in the manuscript file to start this chapter:

```
@Chapter(Titles, Sections, and the Table of Contents)
```

The following command appeared to mark the beginning of the current section:

```
@Section<When There is a Table of Contents>
```

You might have noticed that we are calling these statements sectioning *commands* unlike the heading *environments* in Section 6.1. This distinction between commands and environments is very important here. You cannot use @Begin and @End with command names; you cannot say, for example, @Begin(Chapter). (The Comment command is the only exception to this rule.)

Most document types define other sectioning commands that are not part of the hierarchy for the main body of the document. They are designed for other purposes, and their names are mostly self-explanatory.

The best way to understand all this information is to create a small manuscript containing a number of sectioning commands, run it through Scribe, and print the result. (If you create several .MSS files using different document types and variant forms, you obtain a menu of all the heading styles available at your site.)

If you run a set of sectioning commands through Scribe, look closely at the listing of the results. See how the different kinds of headings differ in appearance. Scribe finds the definition for each of the sectioning commands in the document type Database and applies it to printing the section title. In some document types, certain commands start a new page and others do not. For example, in document type Manual, the @Chapter command always starts each chapter on a new page.

6.3 Title Pages

The sectioned document types (Report, Manual, and their variations) include a simple mechanism for generating title pages. The title page prints on a page by itself with centered text.

There are many standards for title pages. Most organizations that have their own letterhead also have their own format for a title page. The Modern Language Association suggests one format for the title pages of dissertations; many universities and research institutions use other formats. Different Scribe sites use different conventions for title pages, so we cannot document them all here. Instead, we try to give you the general flavor of title pages by showing you how one particular title page format works, and entreat you to consult your ***DBA*** for details of your local title page format.

A titlepage is defined in the Scribe Database to have four major component environments: a *TitleBox* (which contains the manual title), a *CopyrightNotice*, a *ResearchCredit* (a paragraph at the bottom), and the text of the *TitlePage* itself (lines containing author names, affiliation, and so on).

Assemble the information for your title page. Sort the information according to the compo-

nent to which it belongs. Then use the following guide to organize the manuscript for the title page. The title page appears in the document wherever you put the @TitlePage environment in the manuscript. The best place to put a titlepage is at the beginning. This position results in a title page as the first page of your document, with no interference from page headings or page numbers.

The TitleBox, CopyrightNotice, and ResearchCredit components must be nested within the Titlepage environment. Thus, the framework of commands for a standard title page looks like this.

@Begin(TitlePage)
@Begin(TitleBox)
text of the title
@End(TitleBox)
text of the titlepage
@CopyrightNotice(*name of the copyright holder***)**
@Begin(ResearchCredit)
paragraph describing the document
@End(ResearchCredit)
@End(TitlePage)

Technical report title pages have a "box" containing the title, author, and publication date. This box always has to be in a certain position on the page. The TitleBox environment automatically places its text in the correct page position.

When you specify @Begin(TitleBox), the next line of text is placed right at the top of the title box region. Use the @MajorHeading or @Heading commands (see Section 6.1, beginning on 59) to get large- or medium-sized letters for the title box text.

When you say @End(TitleBox), Scribe moves to a region of the page below the title box. There it puts any text that is within TitlePage but not inside any of the other environments of the TitlePage.

The CopyrightNotice environment puts a copyright notice with the current year on the title page. Suppose you typed this text into the manuscript file:

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

Scribe would put the following line on your title page, at the standard spot near the bottom of the page.

Copyright © 1985 L. Frank Baum

The ResearchCredit environment puts a research funding credit (or any other text you want) at the bottom of the title page. Any text that you put between @Begin(ResearchCredit) and @End(ResearchCredit) appears in an appropriate spot at the bottom of the page, justified and single-spaced.

Chapter Seven
Numbers, Labels, and Cross References

One of the most useful characteristics of Scribe is its automatic numbering of various things in your document. Each chapter and section has a number. Each page has a number. Theorems and footnotes and figures all have numbers. This numbering is automatic, which means that you never have to worry about fixing the numbers after adding or removing a chapter or footnote or figure. However, the automatic numbering means you need a special method for cross referencing. You cannot simply refer to the numbers, because you don't know in advance what number Scribe is going to assign to something. In this chapter, we explain how to use Scribe's numbering, counting, and cross reference mechanisms.

7.1 Numbering and Scribe Counters

Scribe has a general facility for counting and numbering. You have seen several instances of it already: page numbering, chapter and section numbering, footnote numbering, @Enumerate numbering, and so forth. Some counters are directly built into Scribe; others are defined as part of the document type.

Like other Scribe entities, each counter has a name. The name of the counter for page numbers, for example, is ''Page''. The name of the counter for figure numbers is ''FigureCounter''. Counters are divided into two groups conceptually — those that number *things* and those that number *places*. By our definition, Figures, Tables, Equations, and Enumerate items are *things*; Chapters, Sections, and Appendixes are *places*. (A Page is neither, or both, depending on how you look at it.)

7.1.1 Current Values of Counters

Scribe provides a means for you to print the value currently associated with a counter. Every counter has a current numeric reference value. The @Ref command retrieves and prints the counter value. For example, @Ref(Chapter) produces 7, the value of the current chapter; @Ref(Section) produces 7.1. Section 7.2 describes using @Ref with a codename to produce cross references.

Remember in Section 4.3 we described @Value(SectionNumber). Notice the distinction between it and the @Ref(Section) command. @Value(SectionNumber) gives you the number produced by the most recent sectioning command (whatever its level), and @Ref(Section)

gives you the value of a particular counter (the counter named Section). Thus, for this page, @Ref(Section) produces 7.1 while @Value(SectionNumber) produces 7.1.1.

Some counters have titles as well as numeric values, such as the ones for the sectioning commands. The @Title command retrieves and prints the current value of the title of the counter named as its argument. For example, @Title(Chapter) currently produces "Numbers, Labels, and Cross References". At this time, Scribe has no mechanism for doing cross references to titles of counters, although you could imitate that effect using @String. See page 32.

The difference between @Value(SectionTitle) and @Title(Section) is analogous to that just described for numeric values. @Title is most useful in running headers.

7.1.2 Changing Values of Counters

Normally, Scribe automatically takes care of changing values of counters. For example, it changes the page counter every time it starts a new page. However, you can change the value of any counter directly when necessary by using the @Set command. It sets counters to particular numeric values. For example, suppose the first page in your document has to be page 4 for some reason. The following example sets the current page number to 4, regardless of what it used to be.

```
@Set(Page=4)
```

You can also change the value of a counter relative to its old value. When the number has a plus or minus sign, Scribe will add or subtract the number from the previous counter value. The following example increments the chapter counter by one.

```
@Set(Chapter=+1)
```

The next example subtracts one from the current footnote number.

```
@Set(FootnoteCounter=-1)
```

It is possible, but more complicated, to change the way a counter's number is printed. This modification is done using templates. See Chapter 15 for details.

You can also use the @Set command to set one counter equal to another or to set a counter equal to the contents of a string:

```
@String(StartingPageNumber="7")
@Set(Page=StartingPageNumber)

@Set(Appendix=Chapter)
```

This example sets the page number to 7 and the appendix number to be the same as the current chapter number.

Similarly, you can use the @String command (see Section 4.2) to set a string to the current (numeric) value of a counter:

```
@String(CopyofPage=Page)
```

The string CopyofPage would be set to the sequence of digits that represents the current numeric value of that counter, regardless of the numbering or referencing templates.

7.2 Cross Referencing

If you put your finger down at random on some page in the middle of this manual and try to explain where your finger has landed to someone at the other end of a telephone, you might try several methods for describing its position. You could give the page number (67), or you could give the section number (7.2). If it happened to land in the middle of a figure or table or theorem, you could give the figure number or table number or theorem number. You could give the title of the section, ''Cross Referencing'', and expect your friend to look it up in the Table of Contents or Index.

A cross reference in a document is an attempt to help your reader find some particular place elsewhere in the document. Whether you refer to it by page number, by section, or by title is entirely up to you. It is usually a notation like ''see page 13'' or ''see Chapter 5''. If you are typing a document by hand or with a simple word processor, it is a major feat to get cross references right, especially page number references, 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. In a previous edition of this manual, we shuddered to think of the work that it took to get all 5000 to 6000 of them right in each new edition. Since then, we contacted the publisher to find out. The references are resolved manually using a huge number of 3-by-5 cards. If only they used Scribe!

Scribe does cross referencing automatically, by letting you define codewords to mark places or things in the text and then filling in the correct page or section number wherever you refer to the codeword.

When you are going to refer to something, you need to decide whether you are referring to a thing or a place. Are you going to say ''see Theorem 12'', which is a reference to a thing, or are you going to say ''see Section 12.2'', which is a reference to a place. We call these two kinds of references *object (thing) references* and *place references*, and there is a separate command to label each.

7.2.1 Marking a Place: The @Label Command

The mechanism for marking places is quite simple. You use a command called @Label with a codename for the place. For example, the codename labelling this section is @Label(LabelCommand)

The command @Label(*codeword*) causes Scribe to note both the number of the most recent sectioning counter and the page number where @Label occurs. @Label can appear anywhere in the document. You can refer to that location from anywhere else in the document. The command @Ref(*codeword*) finds and prints the section number; @PageRef(*codeword*) finds and prints the page number.

Codewords are just words made up of letters and digits. If you restrict your codewords to being just words, then you will never need to consult Appendix F, beginning on page 239, which gives the detailed rules for what characters are permitted in codeword names.

Here is a very short example of how to set up a manuscript to use place cross references.

```
@Make[Manual]
@Chapter[First Chapter Heading]
@Label[George]
This portion is Section @Ref(George) in the document.
There are only two parts to this document.
The other one is Section @Ref(Harry).
@Chapter[Second Chapter Heading]
@Label[Harry]
This portion is Section @Ref(Harry) in the document.
The other one is Section @Ref(George).
```

The hardest part of using this system involves thinking up the code names for the sections! Try to use a consistent scheme (perhaps a prefix plus a word from the title of the section being labelled). Scribe provides an easy way to keep track of the names you have used. Look in Section 13.4 for discussion of the .OTL file.

7.2.2 Marking a Thing: The @Tag Command

The @Tag command marks things. You use it in exactly the same way as you use @Label. Instead of noting the section number, Scribe remembers what kind of thing the @Tag command occurs in, for example, a table or a footnote. If @Tag appears within a figure, then the identity of the object is the figure number. If it appears in a footnote, then the identity of the object is the footnote number. If it appears in an @Enumerate command, then the identity of the object is the appropriate paragraph number in the Enumerate list.

As with the @Label command, @Ref(*codeword*) retrieves and prints the number associated with the labelled object; @PageRef(*codeword*) finds and prints the number of the page on which the @Tag command occurred.

7.2.3 Distinguishing Places and Things

The following example demonstrates the difference between the @Tag and @Label commands.

Manuscript Form:

```
There are three good reasons to take a job at a university:
@Begin(Enumerate)
June

July
@Tag(What)
@Label(Where)

August
@End(Enumerate)
Of these reasons, @Ref(What) is the best; see Section
@Ref(Where).
```

Document Result:

There are three good reasons to take a job at a university:

1. June
2. July
3. August

Of these reasons, 2 is the best; see Section 7.2.3.

Scribe always records the page number for both @Label and @Tag commands. This record means that you can always refer to a page number with @PageRef(*codeword*) regardless of whether the codeword marks a place or a thing.

For example, assume that some object has been tagged with @Tag(LetterFig).

Manuscript Form:

```
Look at Figure @Ref(Letterfig), page @PageRef(Letterfig).
```

Document Result:

Look at Figure 5-5, page 55.

Figures and Tables place a restriction on where the @Tag command must go. See Section 8.3 for details.

7.3 Forward References

What happens when you want to refer to something (with @Ref) before Scribe has seen the corresponding @Label or @Tag command? This situation is called a *forward reference.* Scribe processes your manuscript file from beginning to end. Therefore, when it sees @Ref, it has no way of knowing where the @Tag or @Label is going to be.

When Scribe has finished processing a manuscript file, it stores information about all of the @Label and @Tag commands in an auxiliary file. When you process the manuscript file a second time, Scribe uses this information to print your cross references.

The cross reference information stored from one run might be somewhat out of date for the next run because you normally change the manuscript file in between Scribe runs. At the end of a run, Scribe tells you if any of the cross references are wrong. Since normal evolutionary development of a document involves alternate editing and reprocessing, the references are correct almost all of the time. If you want perfection on a particular run, it is simple enough to run it through Scribe twice in succession.

Suppose you use @Ref with a codeword that Scribe has not seen so far in the manuscript. This reference could be a forward reference or it could be a mistake. In any case, Scribe puts the codeword, capitalized, into the document in place of the number it doesn't have. If the label is defined later in the document, then it is a forward reference and is resolved correctly the next time that the document is processed. However, if the label is not defined anywhere in the document, then it is an ***undefined label,*** and Scribe prints an error message to that effect at

the end of the processing. Figure 7-1 on page 71 shows the appearance of both a forward reference and an undefined reference after one and two times through Scribe.

The information that Scribe saves for itself from one run to the next is stored in an *auxiliary file*, with the file type .AUX. The name of the auxiliary file is the same as the name of the manuscript file, with the extension .AUX. For example, in processing a manuscript file named "TEST5.MSS", Scribe generates an auxiliary file named "TEST5.AUX".

As a user of Scribe, you will never need to read or understand the contents of an .AUX file. Never attempt to edit an .AUX file. Scribe assumes that the contents of an .AUX file are unchanged since it was last written by Scribe and does no error-checking of its contents. If you alter a .AUX file, you can cause random errors to occur inside Scribe when it is processing your file.

Scribe also stores its auxiliary information in an *outline file* in a form suitable for people to read. This file contains all the entries in the table of contents, all the @Label codewords, and all the @Tag codewords. For each entry, Scribe records its document page number and its manuscript file location. The outline file has the same name as the manuscript file with the extension .OTL. For example, Scribe generates "USER.OTL" while processing "USER.MSS". The outline file is an invaluable reference for any large document, particularly those written by more than one author (see Chapter 13).

Manuscript Form:

```
@Label(SimpleCommands)
```

The following sentences occur at some point later in the manuscript (Note the misspelling of the codeword as ''SimpleComands'' instead of ''SimpleCommands'').

```
In Section @Ref(AdvancedCommands), we will describe
some of the more advanced commands in the program.
The simple commands that we discussed in Section
@Ref(SimpleComands) may be used in any context.
```

Still later ...

```
@Label(AdvancedCommands)
```

Document Form After First Scribe Pass:

In Section ADVANCEDCOMMANDS, we will describe some of the more advanced commands in the program. The simple commands that we discussed in Section SIMPLECOMANDS may be used in any context.

Document Form After Second Scribe Pass

In Section 5.3, we will describe some of the more advanced commands in the program. The simple commands that we discussed in Section SIMPLECOMANDS may be used in any context.

Notice that the misspelled label is still unresolved after two passes.

Figure 7-1: An Example Showing a Forward Reference and an Undefined Label

Chapter Eight
Figures and Tables

Scribe provides a means for managing tables and figures in a document. It can place captions in the figures or tables for you and automatically keep track of their numbers. The figure and table mechanism is available in all document types that have a Table of Contents — Article, Report, Manual, Thesis, and so forth.

Note: Almost all of the discussion in this chapter applies to both figures and tables. Sometimes we say "figure (or table)" and sometimes, to avoid clumsiness, we say "figure". Unless we explicitly say, "This option works only for figures," you can assume that it also works for tables.

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

A figure or table has three parts:

1. A *body*. The figure or table body can contain either text produced with Scribe, graphics produced by a graphics package with adequate space left by Scribe via the @Picture command, or blank space where you later paste in whatever you want.
2. A *caption*. All figures and tables must have captions. You provide the text of the caption and decide (by putting it above or below the body) how the caption is to be placed with respect to the figure body.
3. A *number*. Scribe automatically assigns numbers to figures and tables. You use the standard Scribe cross-reference mechanism with the @Tag and @Ref commands to refer to them; see Chapter 7.

Figures and Tables have the property that, if they do not fit on the current page, Scribe is free to move them to a new page after first filling the current page. We call this property *Float* and refer to Figure and Table as *floated environments*.

8.1 Generating Figure and Table Bodies

You can generate the body of a figure or table by using any of the standard Scribe environments (such as Format, Verbatim, or Example), by using the @Blankspace command, or by using the @Picture command. The standard environments will not be discussed again in this chapter, but the two commands available for creating figures and tables are discussed below.

8.1.1 The @Blankspace Command

The @Blankspace command leaves blank space in the output document so that you can paste in some picture, table, or chart of a specified size. The syntax of the command is

@Blankspace (*vertical-distance*)

The *vertical-distance* may be expressed in many different units — for example, inches, millimeters, points, or lines. Appendix F.8 on page 242 lists all of the ways that you can specify distance to Scribe.

For example, @Blankspace(3 inches) leaves enough space to paste in a 3-inch high picture. @Blankspace(16 cm) leaves a blank space about 16 centimeters high. We say "about" because Scribe always leaves a small margin above and below the blank space that you request. This extra space means that you can measure the thing you want to paste in without worrying about how much space to allow around it.

8.1.2 The @Picture Command

You can think of the @Picture command as an automated way of leaving blank space and pasting a picture in the document; it leaves a specified amount of space and automatically includes an image that has been produced with a graphics package *not included with Scribe.* For example, Figure 8-1 on page 75 was placed automatically by the @Picture command.

The @Picture command only works for certain devices. In particular, the device must have the physical ability to print images. Contact your ***DBA*** for details. Even though the @Picture command does not work for the LPT, for instance, you can run a file which contains an @Picture command through Scribe for device LPT without getting error messages. Blankspace the size of the picture is left in the document. That way, you can still print an LPT proof copy of a file that contains an @Picture command and is intended for final printing on the Xerox 9700 without making any changes to the .MSS file.

The syntax of the command is:

@Picture (Size=*vertical-distance*, *GenericDevice=filename*)

The *vertical-distance* may be specified in any units listed in Appendix F.8.

GenericDevice is a name by which Scribe refers to a class of output devices. Here are some values of GenericDevice that can be used with the @Picture command.

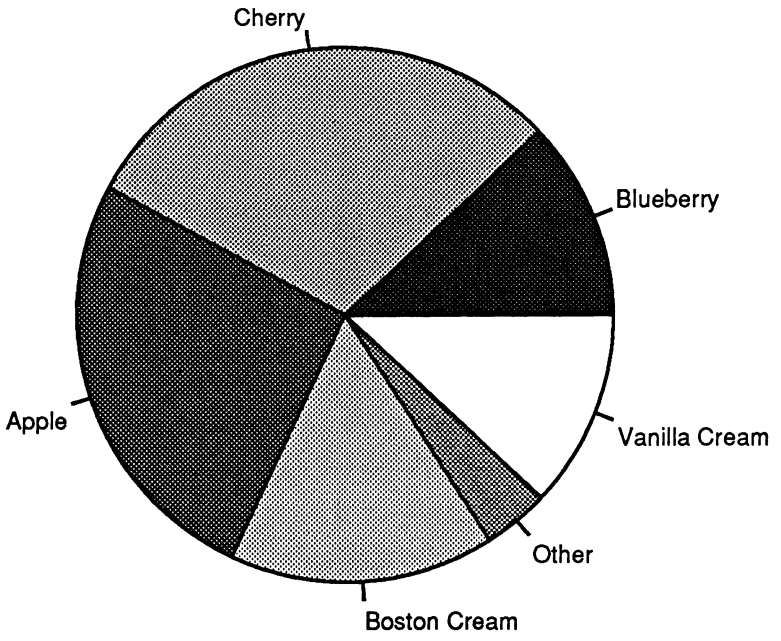

Figure 8-1: An Example of the @Picture Command

Device	GenericDevice
GIGI	ReGIS
Imprint-10	imPRESS
Lasergrafix 1200	NonScaleableLaser
Santec	Santec
X9700	X9700

The *filename* is the name of the file which contains the picture to be included. Multiple device specifications may be given in one command, but the size of the picture must be the same for all the devices specified. For example, this might be a command included in a .MSS file that is to be proofed on the Imprint-10 and printed on the Lasergrafix 1200:

```
@Picture(Size=2inch, Impress=[graph.imp],
        NonScaleableLaser=[pictur.lg1])
```

8.1.3 Drawing Lines: The @Bar Form

Horizontal lines are often used in figures and table to set them off from the running text. Scribe supplies a simple means to draw a horizontal line from the left margin to the right margin: The @Bar form. @Bar takes no argument, but the opening and closing delimiters are required. An example of its usage is shown here.

Manuscript Form:

```
@Bar()
```

Document Result:

8.2 Generating Captions

Figure and table captions are generated with the @Caption command. Scribe prefixes the text that you provide for the caption with the correct figure number and then places the whole thing in the document in the same position as the original command. That is, if you place the @Caption command above the figure body, then the caption appears above the figure body; if you place it below the figure body, then the caption appears below the figure body.

Consider the following example. It would leave 5 inches of blank space and then place the figure caption below the space left for the figure.

```
@Begin[Figure]
@BlankSpace(5in)
@Caption(EM Photograph for Sample 3)
@Tag(EM3)
@End[Figure]
```

The text of the @Caption command becomes the title of the figure or table. Scribe prefixes the title with either the word Figure or the word Table (as appropriate) and the correct number.

Manuscript Form:

```
@Caption(Chart Showing Number of Errors per Session)
```

The appearance of the caption depends on the document type. The environment for captions is usually unfilled, so you must separate long captions into several lines in the manuscript file. For this particular document type, a figure caption looks like this example:

Document Result:

Figure 1-7: Chart Showing Number of Errors per Session

The @Caption command is undefined outside the figure and table environments. If you use it outside a figure or a table, its only effect is to produce an error message.

8.3 Figure Numbers and References to Figures

@Begin(Figure) and @End(Figure) can delimit several distinct figures. That is, if you need several figures kept together in the document, you can put all of them in the same figure delimiters. Using one set of delimiters in that case could create a bit of a problem, however, when it comes to cross referencing. You have to mark for Scribe where one figure ends and the next one starts by using @Caption and @Tag in the right order.

Scribe uses the @Caption command to increment the figure number. It assumes that each figure has one caption; if there are two captions, then there must be two figures. This means that the @Tag command must come after the @Caption command in order to find the right number. (Using @Tag before @Caption ensures that your cross-references will be wrong.)

8.4 Lists of Figures and Lists of Tables

Whenever you include a figure or table in the manuscript, Scribe automatically generates a list of tables and figures in the document as part of the Table of Contents. When you use a Table environment, Scribe produces a list of the tables; when you use a Figure environment, Scribe produces a list of the figures.

8.5 Full-Page Figures

You can ensure that a figure or table occupies an entire page by using FullPageFigure or FullPageTable instead of Figure or Table. The figure then receives a full page to itself. Captions for fullpage figures will not be placed at the bottom of the page as you might expect, because there may be more than one caption in a single full-page figure. The caption to a figure or table will appear wherever you put the @Caption command.

```
@Begin(FullPageFigure)
@Blankspace(6 inches)
@Caption(Photograph of Specimen 3 after Omphaloskepsis)
@End(FullPageFigure)
```

You can have Scribe leave a page blank to add a figure later. Use the @Blankpage command to specify how many blank pages to leave. The following command leaves one page blank for a figure.

```
@Blankpage(1)
```

This figure page has page headings, page footings, and a page number, but nothing else is printed on it (not even a caption).

You might have seen that there is an @Newpage command (Section 10.5), 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 an @Newpage command, it skips immediately to the top of the next page. When Scribe encounters an @Blankpage command, it makes a note about the blank page request but continues on the current page. When it finally comes to the end of the current page, it puts in the extra blank page(s) at that point.

The following two sequences have exactly the same effect.

```
@Begin(FullPageFigure)
@End(FullPageFigure)
```

and

```
@Blankpage(1)
```

8.5.1 Examples of Figure and Table Command Usage

Here is a miscellaneous collection of examples of the use of figure and table commands.

8.5.1.1 An Ordinary Figure with a Cross Reference Tag

```
@Begin(Figure)
@Blankspace(4 inches)
@Caption(Cross-Sectional View of the Drive Shaft)
@Tag(Drive-Shaft-Figure)
@End(Figure)
Figure @Ref(Drive-Shaft-Figure) shows the cross-sectional
view from the left-hand side.
```

8.5.1.2 Two Figures Together on a Figure Page

```
@Begin(FullPageFigure)
@Blankspace(5 cm)
@Caption(Toxicity vs. Time, Sample 1)
@Tag(Sample-1-Toxicity)

@Blankspace(6 cm)
@Caption(Toxicity vs. Time, Samples 2 and 3)
@Tag(Sample-2-3-Toxicity)
@End(FullPageFigure)
```

8.6 Multiple-Page Figures

Many reports contain lengthy tables and figures that need to occupy more than one page in the document. Scribe limits the size of a figure to a single document page. When the material you provide for the body of the figure exceeds the page size, Scribe issues an error message and does what it can with the text.

If you must use run-on figures, they must be paginated manually. You might find the following strategy useful. Find the amount of material that fits in a page and place that much in a FullPageFigure environment containing an @Caption command and an @Tag command. Put the next pages of the figure into FullPageFigure environments with *no* caption or tag commands. (You don't want to use @Caption again because @Caption would give the page a new figure number where you want it to be all part of one figure.) You create headings for the continuation pages of the figure by using a cross reference to the figure number. For example,

```
@FullPageFigure[
@Blankspace(7inches)
@Caption(Ridiculously Large Figure)
@Tag(Ridic)
]
@FullPageFigure[
@Blankspace(7inches)
@Center(Figure @Ref(Ridic), continued)
]
@FullPageFigure[
@Blankspace(5inches)
@Center(Figure @Ref(Ridic), concluded)
]
```

People aren't likely ever to look at a 3-page figure, but at least you know how to make one if you are interested in putting something intimidating into your document.

Chapter Nine
Format Control: Tabs and Columns

9.1 Tabs and Tab Settings

Scribe has a tab mechanism that works very much like the way tabs on a typewriter work, rather than the way tabbing usually works with a computer terminal. You can tell Scribe where to set tabs in any horizontal position. When Scribe encounters a "tab" command, it moves right to the next tab stop and continues formatting there.

9.1.1 Setting Tabs

Clearing and setting Scribe tabs is very simple. The @TabClear command clears all tabs. It takes no arguments. The @TabSet command sets tabs. We suggest that you use the @TabClear command immediately before the @TabSet command so that you are sure that no tabs are already in effect. Both command's syntax are illustrated below:

```
@TabClear()
@TabSet[position, position, ...]
```

where *position* is any horizontal distance. If the distance is an absolute value, the tabs are set relative to the *prevailing* left margin, which is the margin of the environment you're in. (See Figure 15-1 on page 171 for a view of margins and the page layout.) If the distance is a signed value, then the tabs are measured from the previous tab setting (or the prevailing left margin in the case where the first tab setting is a signed distance).

The following sequence first clears any existing tabs and then sets new tabs at 1 and 2 inches from the left margin of the current environment and a third tab 1.5 inches from the second tab setting (making the third tab setting 3.5 inches from the prevailing left margin).

```
@TabClear
@TabSet(1inch,2inches,+1.5inches)
```

The @TabDivide command divides the formatting area into columns, making them however wide they must be in order to fit the requested number of columns on the page. For example, the following command creates five columns of equal width, by setting four tab stops, each 1/5th of the way across the page. (The right margin serves as the final tab stop.)

```
@TabDivide(5)
```

9.1.2 Tabbing to a Tab Stop: The @\ Command

The @\ command tells Scribe to move its formatting cursor to the right until it encounters the next tab setting. This command works exactly like the tab key on a typewriter (except that you don't see what happened until you get your output file printed). For example:

Manuscript Form:

```
@Begin(Format)@TabSet(1in,2in,3in)
Left@\One@\Two@\Three
@End(Format)
```

Document Result:

Left One Two Three

If there is no tab setting to the right of the current cursor position, then Scribe moves the cursor to the right margin.

The TAB character on your terminal's keyboard *does not* generate a Scribe tab command. Instead, it results in enough spaces to move the cursor to the next (hardware) tab position on your terminal, regardless of where the next Scribe tab stop is set. The terminal's tabbing is completely independent of Scribe's tab settings and tabbing commands. You can in fact use both tabbing methods at once, but it can be confusing. In the long run, it is better to take the time to learn the Scribe system.

Tabbing is only meaningful in unfilled environments like Table, Display, and Format. Using the @\ command in running text will produce erroneous and unexpected results.

9.1.3 Centering, Flushing Left, and Flushing Right: The @= and @> Commands

Scribe has commands that take pieces of text and center them or flush them right against some position. (Left flush, being the normal case for many environments, is no big trick.)

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

Manuscript Form:

```
@Begin(Display)
@>like this
@End(Display)
```

Document Result:

like this

However, text can be flushed right against any tab stop:

Manuscript Form:

```
@Begin(Format)@TabDivide(2)
@>text flushed@\
@>right@\
@>against a tab stop@\
@End(Format)
```

Document Result:

text flushed
right
against a tab stop

The exact placement of something flushed right depends both on the @> command, the location of tab stops, and whether the @\ command appears later in the line. The general rule is that the @\ command causes the text to be flushed against the next available tab stop (if there is one). Without the @\ command, Scribe always flushes the text against the right margin.

@>text — Flush the text to the right margin regardless of whether or not there are any tab stops on the rest of the line.

@>text@\ — Flush the text right against the next available tab stop (or the right margin if there are no more tab stops on the line).

Now for a more extensive example.

Manuscript Form:

```
@Begin(Format)
@TabDivide(3)
@>!@>!@>Show the tab stops
1.@>Flushed to right margin
2.@>to tab stop@\
3.@>longer line flushed to different tab stop@\
4.This line starts at the left@>and flushes from here
@End(Format)
```

Document Result:

```
                        !                       !     Show the tab stops
1.                                                    Flushed to right margin
2.              to tab stop
3.              longer line flushed to different tab stop
4.This line starts at the left                        and flushes from here
```

Line 1 does not contain a tab command (@\) after the flushright command (@>). Therefore, the text of the line is flushed against the right margin. Line 2 has a tab command (@\) at the end. Therefore, the text delimited by @> and @\ is flushed right against the next tab stop. Line 3 is like line 2, except that the delimited text is longer. Thus, the next available tab stop is the second one on the line, and the delimited text is flushed against that one. Line 4 is the same case as line 1 and shows a fragment of a line 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

this text is centered in the left half of the page

this text is centered in the right half of the page

Scribe uses tab settings and margins as the reference points for centering. It always uses the current left and right margins as centering guides when there are no other tabs set. If there are tab settings, then Scribe uses them too; the left margin is the first guide point, then the tab settings, and then the right margin.

To center something, you have to mark the left and right ends of the thing to be centered. Mark the left end always with the @= command. Mark the right end in one of the following four ways.

1. a tab command (@\)
2. another mark-left-end command (@=)
3. a flush-right command (@>)
4. the end of the line

When you mark the right end of the text with the end of the line, Scribe centers the remainder of the line using the right margin as the righthand centering guide.

In the middle of a line containing centering or flush-right commands, the same @= or @> command can function both to delimit the end of the previous text and the beginning of the next. Strictly speaking, you have to delimit the text on the left with either @= or @> and on the right with a tab command. However, in sequences of fields, the command marking the beginning of the next field can simulate the @\ command needed to mark the end of the last one. For example, the following two lines are the same:

Manuscript Form:

```
@Begin(Format)
@TabDivide(2)
@=text to be centered@\@=next centered column@\
@=text to be centered@=next centered column
@End(Format)
```

Document Result:

text to be centered next centered column
text to be centered next centered column

Look at the following example. The stars in line 1 show where the tabs are being set. The letters in line 2 are centered in the tab zones, as are the ones in line 3. See how lines 2 and 3 have the same effect, although their manuscript forms look different.

Manuscript Form:

```
@Begin(Format)
@TabDivide(4)
1.@\*@\*@\*
2.@=a@\@=b@\@=c@\@=d
3.@=e@=f@=g@=h
4.Left@=Center@>right
5.Left@=Center@>right@\
@TabClear
6.Left@=Center@>right
@End(Format)
```

Document Result:

```
1.                  *                   *                   *
2.        a                   b                   c                   d
3.        e                   f                   g                   h
4.Left  Center                                                            right
5.Left  Center    right
6.Left                                Center                              right
```

Line 1 positions three asterisks so you can see where the @Tabdivide command set the tab stops. In line 1, the @\ command is being used exactly like the tab key on a typewriter. 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 @= and on the right with @\. Line 3 produces the same effect without using the @\ commands; the tab command is implicit in all @= commands after the first one on the line.

Compare lines 4, 5, and 6. In line 4, @= begins a centering zone; the next command on the line is @>. There is no tab command to mark the end of the centered zone, but @> serves both to close the centered zone and to mark the beginning of the flush-right zone. Therefore, Scribe tabs to the first tab stop and centers the word "Center" between the left margin and the first tab stop. The end of the flush-right zone is delimited by the end of the line rather than by a tab command. Therefore, 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 @\ marks the end of the flush-right zone. Therefore, the word "right" is flushed to the appropriate tab stop, not to the right margin.

Line 6 has the same commands as line 4, but the tab stops have been cleared by an @TabClear command. This command makes the "Center" right marker be the right margin, since there is not a tab stop for it to settle on.

9.1.4 The Lifetime of Tab Settings

In most cases, tab settings last as long as the environment in which you define them. At the end of that environment, the tab settings revert to those that were set before the beginning of the environment. Any tab settings you define in an environment also apply to any environments nested in that one. This carryover means that any new tabs you define are simply added

to the ones already defined; tab clearing does not happen automatically when you enter the environment.

Consider the following example of how tab settings are restored when an environment ends. (The vertical-bar character | at the left end of each text line shows you the location of the prevailing left margin.)

Manuscript Form:

```
@Begin(Format)
@TabSet(1 inch, 3 inches)
|Tabs now set 1 and 3 inches from left margin
@Begin(Display)
@TabSet(1.5 inches)
|The left margin is widened inside Display
|A new tab is set 1.5 inches from the new Display margin
|@\First@\Second@\Third
@End(Display)
|@\Fourth@\Fifth
@End(Format)
```

Document Result:

|Tabs now set 1 and 3 inches from left margin

|The left margin is widened inside Display
|A new tab is set 1.5 inches from the new Display margin
| First Second Third

| Fourth Fifth

Notice that the word "Fifth" is three inches from the original left margin. The command @End(Display) caused the tabs set inside the display to be undone, leaving the original tab settings at 1 inch and 3 inches.

Some environments allow tabs set inside them to remain defined when the environment is exited. This characteristic is most useful with the @^ command (see Section 9.2.1). The following example works because it preserves the tabs set inside it.

Manuscript Form:

```
@b[Provided by the @^National Institutes of Health]
@\(Bethesda, Maryland  20034)
```

Document Result:

Provided by the National Institutes of Health
(Bethesda, Maryland 20034)

The environments that automatically preserve tabs set inside them are the FaceCode environments (listed in Table 3-1, page 14) and the Subscript and Superscript environments, @+ and @-. Tab preservation is under user control. The TabExport environment attribute discussed in Chapter 15 controls what happens to tabs in various environments.

9.2 Fancy Cursor Control

This section tells you how to achieve more elaborate cursor control. If you don't see how these features would be useful to you, then they probably aren't.

9.2.1 Dynamic Tab Settings: The @^ Command

The @^ command (which may appear as a caret or uparrow on your terminal) sets a tab at the current cursor position. Thus, in the following sequence, the first line clears any existing settings (inherited from the surrounding environment), the second line specifies where to set two new ones, and the third line prints a character at each of the new tab settings.

Manuscript Form:

```
@Begin(Format)
@TabClear
This line @^is for setting @^tab settings.
@\!@\!
@End(Format)
```

Document Result:

This line is for setting tab settings.
! !

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. The effects of this command are most predictable with a fixed-width font.

9.2.2 Overprinting

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

Manuscript Form:

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

Document Result:

Simple example of ≠ overprinting.

The @Ovp command is a temporary rather than a general solution to the need for special characters. Note, for example, that the "not-equal" character created above with overprinting is rather poorly formed. Documents with overprinting are not likely to 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. On lineprinter-like devices, underlining is achieved by overprinting. For these devices, use the @u environment, rather than the @Ovp command, to overprint with the underscore character (because different devices use different characters for underlining.)

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

Manuscript Form:

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

Document Result:

This line is Carefullyk

9.2.3 The Return Marker

Scribe maintains a return marker, which is a memory of a particular horizontal position on the page. You set the return marker using the @! command in your text. When @! occurs, the return marker is set to the current horizontal cursor position.

When Scribe encounters the @/ command, it moves the cursor to the position remembered by the return marker. This command works regardless of whether the cursor is to the left or right of the return marker at the time of the @/ command. One use of the return marker is to synchronize columns across widely separated parts of a document. For this reason, the return marker is permanent, that is, it stays set until you move it explicitly. This behavior is different from that of tab stops, which are usually restored to their old values when an environment ends.

9.2.4 Replication: Filling Behind Tabs

The @& command allows you to replicate a pattern often enough to reach a tab stop from the current cursor position. For example,

Manuscript Form:

```
@Begin(Format)
@TabDivide(3)
A@\B@\C
@&D@\@&EF@\G
@\@&-+=@\!
@End(Format)
```

Document Result:

```
A                    B                    C
DDDDDDDDDDDDDDDD EFEFEFEFEFEFEFEFEFEF G
                     -+=-+=-+=-+=-+=-+=-+=- !
```

The @& command serves as a marker of the left end of the pattern to be replicated. When Scribe finds the tab command, @\, it moves the cursor to the next tab setting as usual, but the space over which the cursor moves to get to that tab stop is filled with as many copies of the pattern as are needed to fill the space.

To draw a line all the way across the current environment, try this command:

Manuscript Form:

```
@Format(@TabClear()@&-)
```

Document Result:

The @) command performs *synchronous replication.* It means that it replicates a pattern (in much the same way as the @& command), but it forces the pattern to begin in a column position that is an integral multiple of the pattern's size. This feature is useful for making the fill patterns line up neatly. For example, in a pattern consisting of dots and blanks, all of the dots will line up under one another. In the following example, note the difference in behavior between the @& and @) commands (the example for @& appears first).

Manuscript Form:

```
@SubHeading(Directory)
@Begin(Format)
@TabDivide(2)
Produce@&.    @\1
Meat@&.    @\3
Housewares@&.    @\6
@End(Format)
```

Document Result:

Directory

Produce. 1
Meat. 3
Housewares. 6

Manuscript Form:

```
@SubHeading(Directory)
@Begin(Format)
@TabDivide(2)
Produce@).    @\1
Meat@).    @\3
Housewares@).    @\6
@End(Format)
```

Document Result:

Directory

Produce 1
Meat 3
Housewares 6

9.3 Multiple Columns

Multiple column output can easily be produced using Scribe. All you need to do is tell Scribe the basic information about the columns using the environment attributes listed below:

Boxed	Specifies that the environment is to be separated vertically from any text before and after the environment. It must be specified in all multiple-column environments.
ColumnMargin	Specifies how much space will be left between columns. This distance may be specified in any horizontal units.
Columns	Specifies the number of columns into which the environment will be set. The default is 1. Example: Columns 3.
ColumnWidth	The sum of the ColumnMargin and LineWidth attributes. This distance may be specified in any horizontal units.
LineWidth	Specifies the width of an individual column, *not the width of the entire line*. This distance may be specified in any horizontal units.

Although all five of the attributes may be used to specify multiple columns, only Boxed and Columns are necessary.

Using these attributes, you can either define your own multi-column environment or modify an existing environment to become multi-columned. Defining and modifying environments is discussed in detail in Chapter 15. We will show one example of modification in this chapter to illustrate the multi-column feature, but please refer to Chapter 15 for further capabilities.

In producing multiple columns, Scribe will completely fill one column with text and then move to the top of the next column. For the times when you do not want a column completed, use the @NewColumn command. It is similar to the @NewPage command in that @NewColumn begins printing text in the new column immediately in the same way that @NewPage begins printing text on a new page immediately. Multiple columns are fully recursive. That is, you may start a multi-column environment while inside one column of another multi-column environment.

As an example of multiple column output, see Figure 9-1. It is a two-columned reproduction of the enumerated list first shown in Figure 3-3 on page 26.

Manuscript Form:

```
@Modify(Enumerate, Columns 2, ColumnMargin 0.5 inch,
        ColumnWidth 2.75 inch, Boxed)
@Begin(Enumerate)
The numbers in an Enumerate environment are filled in
by Scribe.  Some styles use roman numerals instead of
numbers.
@Begin(Enumerate)
When you nest one Enumerate inside another, the numbers and
margins are adjusted appropriately.

The switch from numbers to letters is specified in the
Scribe Database file for the output device being used.
Deeper nesting will produce other kinds of
numbering.
@End(Enumerate)

@NewColumn()
@Begin(Multiple)
Normally, each blank line starts a
new item because each paragraph is an item.

When you need more than one paragraph in a single item,
use @@Multiple or the @@Begin(Multiple) and
@@End(Multiple) commands.
@End(Multiple)

When you close the Multiple environment, the next paragraph
is a new item.
@End(Enumerate)
```

Document Result:

1. The numbers in an Enumerate environment are filled in by Scribe. Some styles use roman numerals instead of numbers.
    a. When you nest one Enumerate inside another, the numbers and margins are adjusted appropriately.
    b. The switch from numbers to letters is specified in the Scribe Database file for the output device being used. Deeper nesting will produce other kinds of numbering.
2. Normally, each blank line starts a new item because each paragraph is an item.

    When you need more than one paragraph in a single item, use @Multiple or the @Begin(Multiple) and @End(Multiple) commands.
3. When you close the Multiple environment, the next paragraph is a new item.

Figure 9-1: An Example of Two-Columned Output

Chapter Ten
Hyphenation

Scribe assembles words into sentences, disassembles sentences into lines, and assembles lines into pages. It chooses the word breaks, line breaks, and sentence breaks using very simple techniques, which cannot always do the right thing. Scribe sometimes decides to break a line at an undesirable place; sometimes it might break a page between two lines that really ought to be together on the same page. This chapter discusses the mechanisms in Scribe that let you change its selection of word, line, sentence, and page breaks.

10.1 Hyphenation

In Scribe, there are several ways to specify word breaks, depending on what you consider a word to be. A ''word'' can be a sequence of letters with no spaces, or it can be a sequence of letters and spaces which is to be considered one entity. The method of breaking a word in each of these cases is different.

If you just want Scribe's standard, garden-variety method of hyphenation, there is no need for you to read the rest of this chapter. Just put the following line near the beginning of your document:

@Style(Hyphenation On)

Hyphenation is the usual method of breaking a word. It is used when lines are being filled and when a ''word'' contains no spaces. Scribe is equipped with several methods of hyphenation — dictionary, algorithm, or a combination of both methods.

There is much flexibility in Scribe's hyphenation: It may or may not care about case distinctions, one or more dictionaries can be used in a single document, dictionary lookup may be used throughout an entire document or only in sections of it, algorithmic hyphenation may be used throughout an entire document or only in sections of it, and dictionary lookup and algorithmic hyphenation may both be used in the same document. Each of these hyphenation methods will be discussed in this section. But before we get into the details, let's look at an overview of the possibilities.

10.1.1 Hyphenation Dictionaries

The Scribe hyphenation method that allows the most user control is Hyphenation by dictionary. It involves user-created dictionaries which contain lists of words with their legal hyphenation points specified. The procedure, explained very briefly, is that when Scribe needs to break a word, it looks up that word in the dictionary. If it is there, Scribe checks to see if any of the specified hyphenation points satisfy the requirements for correctly filling the line of text, and, if the requirements are met, it hyphenates it according to the dictionary listing selected. (If the word is not in the dictionary, no hyphenation is performed, and the word is listed in the .ERR file.)

A hyphenation dictionary is created with either the @DefineHyphenationDictionary command or its synonym @DefineHyphenationDictionaries. The syntax for the commands is shown here using the @DefineHyphenationDictionary command as an example:

@DefineHyphenationDictionary [*dictionary-name = delimited-word-list*]

where *dictionary-name* is the full name of the dictionary, and *delimited-word-list* is a list of dictionary entries separated by spaces, tabs, or carriage returns. The same dictionary name cannot be used for more than one dictionary. Please note that commas cannot be used to separate the entries in this command. All text characters in hyphenation dictionaries are taken literally to be part of the word entry, except for a space and the ones listed in Table 10-1.

. (period)	Discretionary hyphen; a possible hyphenation point. This character is the convention used in ordinary English dictionaries. The "-" character (hyphen) is used to represent *required* text hyphens.
@_ (at-underscore)	Discretionary hyphen; a possible hyphenation point and a synonym for "." (period). This option is included so that hyphens can be specified in the same way in .MSS files and in dictionaries. Its use is not recommended, though, because the resulting entries are very hard to read.
- (hyphen)	Required (text) hyphen. If the @Style parameter or environment attribute HyphenBreak (see Section 10.1.5) is On, Scribe will not look up a word containing text hyphens in any dictionary. Therefore, a match to a dictionary entry containing hyphens will only occur when HyphenBreak is Off.
@@ (at-at)	A literal at-sign. Since a single at-sign is taken as the beginning of a Scribe command, a means for specifying an at-sign in a word is necessary.
@. (at-period)	A literal period. Since a period is used to designate conditional hyphens, a means for specifying a period in a word is necessary.
@␣ (at-space)	A literal space. Since a space is used as an entry separator in hyphenation dictionaries, a means for specifying a space in a word is necessary.

Table 10-1: Dictionary Punctuation with Special Meaning

A few lines from a possible dictionary are shown below.

```
@DefineHyphenationDictionary[Typical (
aard.vark
aba.cus  aba.ci  aba.cus.es
ab.bre.vi.a.tion
al@_ter@_na@_tive
@@DefineTypeCase
Bologna  bo.lo.gna
MyFile@.Mss
well-disposed
)]
```

If a word appears in a dictionary without any hyphenation points, as in "Bologna" in the above example, then Scribe will not hyphenate it at all. The hyphenation dictionary entries need not be in alphabetical order.

In the above example, the words "Bologna" and "bologna" appear. Whether or not case is important is decided by the method of hyphenation you choose. (Details of the methods are discussed in Section 10.1.3.) Assuming that you choose a hyphenation method that is case sensitive, then "Bologna" (a city in Italy and a proper name) would not be hyphenated, but "bologna" (a kind of meat) would be. The scheme is not perfect, though. When a word that is not a proper name occurs at the beginning of a sentence and is thus capitalized, Scribe hyphenates it according to the entry in which the first letter is capitalized. With the above dictionary entry, the first word of the sentence, "Bologna is better than salami", would *not* be hyphenated. However, this problem should be insignificant in most instances because it is unaesthetic anyway to hyphenate the first word of a sentence.

If a word occurs more than once in a dictionary with different capitalization and you are not using a case-sensitive method of hyphenation, one of the given hyphenations will be used, but it is unpredictable which one it will be.

If a word occurs in more than one dictionary but only once in each, you will always be sure which hyphenation will be used, because Scribe searches dictionaries in the order in which you specify them. (Details on how you go about telling Scribe what dictionaries you want to use are discussed later in this section.) Therefore, the hyphenation used will be the one from the first dictionary searched that contains the word.

After you define a dictionary, you need to tell Scribe that you want to use it. "HyphenationDictionary" (synonym: HyphenationDictionaries) is both an @Style parameter and environment attribute. When you want a dictionary to be used for the entire document, you specify that name in an @Style command, using this syntax:

@Style[HyphenationDictionary="*dictionary-name*"]

For example, the declaration

```
@Style[HyphenationDictionary = "Chemical"]
```

might be included at the beginning of a document laden with chemical terminology to tell Scribe to use the dictionary named "Chemical" for hyphenation throughout the document.

When you want a dictionary to be used only in a certain environment, then you specify the "HyphenationDictionary" environment attribute for that environment. Let's say that your document contained an excerpt from a Physics text book. The following command might be included in the .MSS file:

```
@Begin[Quotation, HyphenationDictionary = "Physics"]
```

It would tell Scribe to use the dictionary, "Physics",[1] for the duration of the quotation. When the excerpt is completed and the Quotation environment closed, the Physics dictionary is no longer used.

For Scribe to be able to find your hyphenation dictionary after you've created it, you **must** include the following command as the first command in the file:

@Marker(HyphenationDictionary, *dictionary-name*)

where *dictionary-name* is the full name of your hyphenation dictionary. Also, the file must be named ***dictio*.hyp**, where ***dictio*** is the first six letters of the dictionary name.

10.1.2 Hyphenation by Algorithm

The hyphenation method that is at the opposite end of the spectrum from the completely user-controlled dictionary method is the automatic hyphenation method: by algorithm. A modified version of the algorithm for English designed by Donald Knuth and Frank Liang is part of Scribe. The algorithm is supplemented by an exception dictionary called AutomaticExceptions which contains correct hyphenations of words on which the algorithm is known to fail. The algorithm is built into Scribe and cannot be altered.

An exception dictionary is different from a hyphenation dictionary in only one respect: It is used *with* the hyphenation algorithm while a hyphenation dictionary is used *without* the algorithm. The same commands discussed in Section 10.1.1 apply to exception dictionaries, so defining an exception dictionary is the same process as defining a hyphenation dictionary. Exception dictionaries need to be defined when you want to guarantee that a word be hyphenated in a particular place. Once it is defined, you may want to use it throughout the entire document or simply in one environment, and the process of specifying that information is similar to the procedure for hyphenation dictionaries. The "ExceptionDictionary" (synonym: ExceptionDictionaries) @Style parameter is used to tell Scribe to use one or more exception dictionaries for the whole document:

@Style(ExceptionDictionary="*dictionary-name*")

The environment attribute of the same name is used for those times when one or more exception dictionaries are to be used for a particular environment:

@Begin(*environment-name*, ExceptionDictionary="*dictionary-name*")

In both cases, multiple dictionary names are separated by commas:

```
@Style[ExceptionDictionaries = "Compsci, Math"]
```

Let's look at some examples. If you had a quotation in British English, and you were using the algorithm to hyphenate, you could tell Scribe to use a separate British exception dictionary[2] by including the following lines in your .MSS file:

---

[1] The dictionary names, "Chemical" and "Physics", are used for illustrative purposes only. These dictionaries are not supplied by UNILOGIC.

[2] This dictionary is not included with Scribe, either.

```
@Begin[Quotation, ExceptionDictionary = "British"]
```
text of quotation
```
@End[Quotation]
```

When Scribe is automatically hyphenating, it will first check each word to be hyphenated against all user-specified exception dictionaries, *searching them in the order in which they are listed in the @Style command or environment.* If the word is found in an exception dictionary, that hyphenation is used. If it is not found, Scribe checks in the algorithm's own exception dictionary for the word. If it is found there, then that hyphenation is used. If it is not found, then, finally, Scribe uses the algorithm to hyphenate it. In the last @Style example above, Scribe will check the words to be hyphenated against the dictionaries, "Compsci", "Math", and "AutomaticExceptions", in that order. If the word is not found, Scribe will hyphenate it according to the algorithm.

10.1.3 Hyphenation Methods

Now that we've discussed dictionaries and the algorithm, we can talk about the different ways the dictionaries and the algorithm can be used: Should it look in hyphenation dictionaries for case-specific words, should it only use the algorithm and its exception dictionary, should it use both user-specified dictionaries and the algorithm, or should it simply not hyphenate?

The hyphenation method, or way that hyphenation is accomplished, is controlled by the @Style parameter and environment attribute, "Hyphenation". There are many different modes of hyphenation in Scribe. They are described below, listed by value of the Hyphenation parameter/attribute. In all of the methods, If the word to be hyphenated ends with a character in the set {. , ; : ! ?}, then the last such character is not considered a part of the word when hyphenation is attempted.

AutomaticExact Hyphenation is performed by the modified Knuth-Liang hyphenation algorithm and exact-case matching in the exception dictionary. Only words consisting of entirely lowercase letters are algorithmically hyphenated.

AutomaticFolded Hyphenation is performed by the modified Knuth-Liang hyphenation algorithm and exception dictionaries with case-folded matching in the dictionaries, meaning that capitalization is ignored for hyphenation purposes. Only words consisting entirely of letters, regardless of case, are algorithmically hyphenated. Synonyms: On, True, Yes.

DictionaryExact Hyphenation is performed by hyphenation dictionaries only with exact-case matching.

DictionaryFolded Hyphenation is performed by hyphenation dictionaries only with case-folded matching, meaning that capitalization is ignored for hyphenation purposes.

False No hyphenation is performed. (Hyphenation may still occur at discretionary and text hyphens, however.) Synonyms: No, Off.

No Synonym for False.

Off Synonym for False.

Old Scribe 3A/3B hyphenation with case-folded matching is performed, meaning that capitalization is ignored for hyphenation purposes. (Included for upward compatibility.) All hyphenation dictionaries used in this mode

	must be in the old format required by Scribe 3A and 3B. Synonym: OldFolded. UNILOGIC does not recommend using this value, since it will eventually be phased out.
OldFolded	Synonym for Old.
OldExact	Scribe 3A/3B hyphenation with exact dictionary lookup is performed. (Included for upward compatibility.) All hyphenation dictionaries used in this mode must be in the old format required by Scribe 3A and 3B. UNILOGIC does not recommend using this value, since it will eventually be phased out.
On	Synonym for AutomaticFolded.
True	Synonym for AutomaticFolded.
Warn	No hyphenation is performed, but a list of words that Scribe wanted to hyphenate is produced in the .ERR file.
Yes	Synonym for AutomaticFolded.

10.1.4 Using Hyphenation

Hyphenation may or may not be used in a document. The following @Style command is used at the beginning of the .MSS file to enable or disable it for the document as a whole (subject to change by later environments, however):

@Style[Hyphenation = ***value*****]**

Valid entries for *value* are any item from the list in Section 10.1.3. Hyphenation may also be enabled or disabled in a Database file by the ***DBA***.

You can also turn hyphenation on and off for a specific environment by using the "Hyphenation" environment attribute. Let's use a quotation as an example again. If you were using the hyphenation algorithm throughout your document but had an excerpt from a technical journal that you wanted to have hyphenated by a certain dictionary, you would simply specify the desired dictionary in the environment that the excerpt was to be in, as shown in this example:[3]

```
@Begin[Quotation, Hyphenation = AutomaticExact,
        ExceptionDictionary = "Technical"]
<text of quotation>
@End[Quotation]
```

When the excerpt was ended and the Quotation environment closed, the hyphenation method would again be algorithmic.

---

[3] Please notice that this example, similar to the example of a quotation in Section 10.1.2 shows both the Hyphenation and ExceptionDictionary attributes. Both attributes are necessary for hyphenation. Only the ExceptionDictionary attribute was shown in previous examples to avoid issues that had not yet been discussed and could therefore be confusing.

10.1.5 Text Hyphens

This section deals with the effect of *text hyphens*, which are hyphens (ASCII minus signs) that are explicitly inserted into the input file by the user. For example, the user might very well type in the word "self-contradictory". Scribe will not *automatically* treat this hyphen as a possible hyphenation point; the decision about whether or not to treat that character as a possible hyphenation point depends on the setting of the @Style parameter and environment attribute "HyphenBreak", which takes boolean value. If HyphenBreak is set to Off, False, or No, then text hyphens are treated as ordinary alphabetic characters. If HyphenBreak is set to On, True, or Yes, then text hyphens are taken as potential hyphenation points. A leading minus sign is never taken as a hyphenation point; thus, "-131072" will not be divided.

Do not break words across lines yourself by adding a text hyphen at the end of a line. Scribe will not recombine the pieces into an unhyphenated word because it can't tell whether the hyphen is discretionary or not.

Manuscript Form:

```
Never break an other-
wise hyphenless word
across lines.
```

Document Result:

Never break an other- wise hyphenless word across lines.

Even a hyphenated word, such as "strife-torn" should not be broken across lines after the hyphen, or an unwanted space will appear in the output.

10.1.6 Discretionary Hyphens

A *discretionary hyphen* is a marker that may be placed in a word to indicate a potential hyphenation point. If Scribe needs to hyphenate a word and it contains one or more discretionary hyphens, Scribe will consider only those user-defined hyphenation points; no other hyphenation methods that may be in effect at the time will be used. If the word is hyphenated because of a discretionary hyphen, the hyphen character will be inserted by Scribe automatically. If the word does not need to be divided, then no hyphens will appear. The Scribe discretionary hyphen is indicated by the "@_" (at-underscore) command. It gives the user flexibility to prevent a single word from being considered for any hyphenation method.

10.1.7 When Does Scribe Hyphenate?

If the next word to be placed on a line is too long to fit, Scribe must decide whether to place the word on the next line and redistribute the remaining space on the line over the existing word breaks or to hyphenate the word. Ideally, Scribe wants to place the word on the next line. Whether or not it does that depends partially on how much inter-word space would result. Sometimes, moving the word down a line can cause some of the word breaks to appear unsightly, as in the following example:

Excruciatingly unattractive
example illustrating
hypothetically unacceptable
accumulations of whitespace.

As Scribe is filling a line with words, it checks to see if the next word will fit. If it does not, Scribe attempts to justify the line by expanding rubber spaces as equally as possible. If this can be done without leaving too much space, then there is no problem, and the word that did not fit will merely be placed on the next line. If, however, the trial fill leaves too much space, Scribe checks to see whether any hyphenation mode is in effect. If you have requested no hyphenation or the particular word is not to be hyphenated, then there is nothing more that Scribe can do. The word will be placed on the next line, the current line will be filled and excessive spaces will be left between words, as was the case in the above example. The only solution to that situation is for you to reformat the entire paragraph.

How much whitespace is considered unsightly is a matter of choice. Scribe provides the @Style parameter and environment attribute "WidestBlank" to allow you to choose that distance. (Environment attributes are discussed in Chapter 15. The @Style command was discussed in Chapter 4, Section 4.5, so you know that using WidestBlank as an @Style parameter effects the entire document. Using it as an environment attribute produces the same result, but for only the current environment.) WidestBlank takes as value a horizontal distance. For example:

```
@Style(WidestBlank 2.5 chars)
```

If the above command were in your document, then Scribe would attempt to leave no more than a 2.5-character space between words.

It is important to note that WidestBlank is not an inviolate parameter. It is a guideline only. Scribe will attempt to justify lines so that WidestBlank will be obeyed, but may not always be able to because a word is defined to not be hyphenated or there is no hyphenation in effect at the time. As we said before, Scribe can do nothing else but override WidestBlank and place the word on the next line. In fact, of all of the constraints under which Scribe operates, WidestBlank is the *least* sacred. Scribe would much rather exceed WidestBlank than hyphenate incorrectly or allow a word to extend into the right margin.[4]

You need never worry about setting WidestBlank. A default value is specified in the document definition file. But if you need to change it for a particular document or environment, the value may be changed with an initial @Style command or with the WidestBlank environment attribute, respectively. A setting of WidestBlank 0 has a special meaning. It will cause the word that is broken to be hyphenated at the rightmost possible point. (Since WidestBlank 0 cannot literally be satisfied, there seems to be no harm in this definition.)

Whether a word will even be looked up in a hyphenation dictionary is determined by the Hyphenation mode that is in effect (see Section 10.1.3) and by the values of the @Style parameters ShortestHyphenatable and LongestHyphenatable. These parameters take integer values indicating, respectively, the length in characters of the shortest and longest words that

---

[4] There are occasions when Scribe will let a word creep into the right margin, however. If a word to be output is longer than the current value of LineWidth, the word will necessarily extend into the margin.

Scribe should attempt to hyphenate by lookup. For instance, if you did not want any word less than 6 characters or greater than 20 characters to be hyphenated, then you could include this command at the beginning of your .MSS file:

```
@Style(ShortestHyphenatable 6, LongestHyphenatable 20)
```

As with WidestBlank, you do not have to specify those parameters. Default values of ShortestHyphenatable 5 and LongestHyphenatable 99 are in effect when you don't mention them. You can, of course, specify one parameter without specifying the other. An attempt to set ShortestHyphenatable to less than five will not succeed. If there are words in the dictionary whose lengths fall outside the ranges specified by these parameters, such words will not be used for hyphenation.

10.1.8 Verifying Scribe's Hyphenation

Scribe has two supplementary output files which it can produce upon request that are helpful in checking the hyphenation of words. The first file, the Hyphenation Decision (.HYD) file, is a listing of every hyphenation decision that was made in the order that they occurred, where the word was actually broken, what hyphenation method was in effect at the time, what the possible hyphenation points were, and where the word occurred in the .MSS file.

A .HYD file is requested through the use of the command-line switch, "HYD". Seeing that switch, Scribe will produce an additional output file having the same name as the input file, but with extension .HYD. Here is an example of a Hyphenation Decision file:

```
@Comment{LineEndings of SAMPLE.MSS by Scribe 4(1400) on15 July 1985 at 11:13}

SAMPLE.MSS, 02300/1:   "bi.o.graph.i.cal"  found in dictionary Myown
                              where the central problem is that biograph-
                                                                    ical

SAMPLE.MSS, 04600/1:   "in.val.i.date"   found in dictionary Standard
                             use not in accord with instructions will in-
                                                                 validate

SAMPLE.MSS, 13700/1:   "in.herited"   automatically hyphenated
                                        such traits are not commonly in-
                                                                  herited
```

The second part of the broken word is shown flushed right underneath the first part for convenience. It, of course, is not printed this way in the actual document. Words that were hyphenated because of the presence of a discretionary hyphen will appear in the .HYD file. Words that were hyphenated because of the presence of a text hyphen when HyphenBreak is On will not be included.

The second file that Scribe can produce is the Lexicon file (.LEX). It has two forms. The first version, requested via the "V" or "Vocab" command-line option, contains an alphabetical list of all the words in the document and how many times each word appeared. The second version of the .LEX file, requested via the "HV" or "HypVocab" command-line switch, contains each word in the document, with all possible hyphenation points in each word shown according to the hyphenation method in use at the time the word was encountered. The second version of the Lexicon file is particularly useful for checking the correctness of hyphenation dictionaries, exception dictionaries, and the hyphenation algorithm.

10.1.9 The @| Command: Where to Break a Word without a Hyphen

The @| command is used to inform Scribe where, in a filled environment, it is allowed to break a word without using a hyphen. It is useful when you have a word in your .MSS file that can be printed as a single word or as two separate words equally well. For example, if your text contained "Abra@|cadabra", then the output document could contain "Abracadabra" on a single line or "Abra" on one line and "cadabra" on the next line, depending on how the text filling took place.

One handy use for the @| command is to permit line breaks in huge numbers. For example,

```
There are exactly 3,025,679,212,512,@|489,
113,206,955 grains of sand at Coney Island.
```

The @| command actually functions as a null word in a document. Therefore, @| surrounded by spaces results in two spaces between words in the document rather than one space (when the word break wasn't necessary, of course). In proportional fonts, it is often difficult to see the difference, but in the following example, the difference is clear.

Manuscript Form:

```
Four score and @| seven years ago
        vs.
Four score and@| seven years ago
```

Document Result:

```
Four score and  seven years ago
        vs.
Four score and seven years ago
```

You do need to have one space, though, either before or after the command, or the two words on either side of the command run together in the document when no line break is necessary.

10.1.10 Significant Blanks: Where Not to Break a Word

A significant blank is a space that is part of a word (rather than being a word separator). The need for significant blanks often occurs in mathematical formatting. For instance, if you're going to write x = y, you don't want the x on one line and the = y on the other. The blanks in this case are significant ones.

One way to request a significant blank is via the @␣ (@-sign followed by a space) command. Scribe then treats that space as if it were a letter. Thus, you could write the following line to make a single word out of that equation:

```
x@␣=@␣y
```

For long and complicated equations, it gets a bit tedious to put @␣ instead of every blank; it's also quite hard to read. Another way to specify a significant blank is by using the @w command, which makes all of the blanks in its range significant blanks. So, enclosing the previous equation within an @w command has the same effect as making the blanks into significant blanks.

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

The @| command discussed in Section 10.1.9 tells Scribe where it is allowed to break a word without hyphenating it. Note that if you use an @| command inside an @w group, the @| command will win, and a word break will be allowed there.

10.2 Sentence Breaks

Normally, Scribe uses these rules for deciding whether or not a punctuation mark is the end of a sentence:

1. A ''?'', ''!'', or ''.'' character followed by two or more spaces or by the end of a line is considered to be the end of a sentence. If closing parenthesis or closing quote characters come between the sentence-end character and the spaces, it is still considered to be the end of a sentence.
2. A single capital letter followed by a period followed by any number of spaces is considered to be part of an abbreviation, and the spaces after the period are converted into one significant space.
3. A single capital letter followed by a period followed by the end of a line is not considered to be an abbreviation but to be the end of a sentence. If, however, a space character occurs between the period and the end of the line, then the sequence is considered as part of an abbreviation.

Two commands change Scribe's notions of what to do with periods and spaces: the @. command to create an abbreviation period and the @: command to force a sentence break.

The full-stop character in each of the following lines is taken as the end of a sentence.

Manuscript Form:

```
demonstration.  Unless
government.''  They
congress!  He
Senator?)  However,
```

Document Result:

demonstration. Unless
government.'' They
congress! He
Senator?) However,

However, the full-stop character in each of the the following lines is treated as part of an abbreviation.

Manuscript Form:

```
to Mrs. James
from I. F.   Stone
a computer called C.mmp
```

Document Result:

to Mrs. James
from I. F. Stone
a computer called C.mmp

Sometimes you need to use a period in the middle of a sentence to end an abbreviation. In this case, put just one blank after the period, and Scribe treats it as an abbreviation. If you don't want to worry about the number of blanks, then use @. instead. The @. command is the abbreviation-period command; it generates a period and forces a (single) significant space after it. For example,

Manuscript Form:

```
we heard from Mrs@. Smith that her children had returned.
```

Document Result:

we heard from Mrs. Smith that her children had returned.

Whenever Scribe finds a single capital letter followed by a period and a space, it assumes that the letter is an abbreviation, and it reduces all space after it to one significant space. Thus, these two examples are equivalent:

Manuscript Form:

```
The journal was edited by I@. F@. Stone.
The journal was edited by I. F.    Stone.
```

Document Result:

The journal was edited by I. F. Stone.
The journal was edited by I. F. Stone.

Under some circumstances, you need to use a single letter at the end of a sentence and, therefore, would like the period after it to be a sentence-end period instead of an abbreviation-end period. For this purpose, you need to use the @: command, which forces a sentence break:

```
No one was more gullible than I.@:  F. Stone, however,
fell for the ploy completely.
```

or

```
We wanted to start numbering with Appendix I.@:
```

or, worse,

```
We show a complete summary in Appendix @Ref(Summary).@:
```

10.3 Line Breaks

In normal text, Scribe adds words to a line until the line is full. (This action is called *filling*.) Then, and only then, it starts a new line.

In a filled environment like Quotation or Text, you can force Scribe to start a new line even though the old one is not full, by using the @* command. In filled text, @* causes a ''line break'', which is to say that the current line is ended without being justified. Two @* commands in a row do not cause a paragraph break. That is, @* is not the same as a carriage return. When line filling is not taking place (in those unfilled environments like Display that generate one output line from each input line), then @* has the same effect as an ordinary end of line.

Warning: most uses of @* are ''incorrect'', in that there is a better way to accomplish what you are trying to do. @* is included in Scribe primarily for use inside text forms, page headings, and the like. You should never need to use it in running text, and if you find yourself wanting to use it in running text, you are probably doing something wrong.

10.4 Paragraph Breaks

Scribe uses a ''blank line'' to denote a paragraph break. A blank line is defined as one that has no printing characters between two or more consecutive carriage returns. Thus, any line that looks blank on your terminal looks blank to Scribe as well.

Sometimes it is ambiguous whether a particular carriage return in a manuscript file is a real end-of-line carriage return or part of a command. Consider the following example:

```
1. Quasi-stellar objects @Tag(quasars)
   are

2. Quasi-stellar objects
   @Tag(quasars)
   are
```

In the first case, the carriage return following the @Tag should be considered part of the text, whereas in the second case the carriage return following the @Tag should be considered part of the @Tag command. Scribe, in general, handles carriage returns properly, but sometimes its analysis of whether a carriage return is part of the text or part of a command does not agree with yours. To this end, Scribe has two commands for modifying its handling of carriage returns.

The @; command is a non-command; it does nothing at all. Computer people refer to this sort of command as a no-op. However, if you use it in your manuscript file, it ensures that the carriage return is part of the text. To explain, let's return to our @Tag example from above, but now put @; commands into it:

```
1. Quasi-stellar objects @Tag(quasars)@;
   are
```

Scribe always looks around after a command to decide whether or not to throw away the carriage-return that follows the command. If it does not immediately find a carriage return, it

stops looking. In this example, during the processing of the @Tag command, Scribe finds not a carriage return, but an @; command. Therefore, it stops its search for a carriage return. Scribe next processes the @; command, which does nothing, because that is its job. The carriage return after the @; command is now certain to be taken as a text carriage return.

Sometime you might need to force Scribe to ignore a carriage-return that it wants to treat as text. For most users, this circumstance is very rare. You can use the @~ command for this purpose. @~ causes Scribe to ignore all input characters in the manuscript file until the next printing character. Here's an example of using @~ for entering long lines for a Verse environment.

Manuscript Form:

```
@Verse[
Along the roads the telephone poles stand @~
alone like words in a broken conversation
     The wire between them hums a solilo@~
        quy in an endless encore with no ovation
Like all the poetry that has poured from @~
                my lips to fall on fields empty of ears
     Good land is prose, but what grows @~
        there is the poetry of years and years
]
```

Document Result:

Along the roads the telephone poles stand alone like words in a broken conversation
The wire between them hums a soliloquy in an endless encore with no ovation
Like all the poetry that has poured from my lips to fall on fields empty of ears
Good land is prose, but what grows there is the poetry of years and years

10.5 Page Breaks

Producing correct page breaks is a very hard problem, and Scribe doesn't in general do a perfect job of it. The interactions among text lines, footnotes, floating figures, and titles result in some pagination problems with no practical solutions.

Normally, Scribe starts a new page when the old one is full, regardless of what it is doing at the time. The @Newpage command allows you to force a page break before the current page is full. Used alone, @Newpage causes Scribe to put the next output at the top of the next page. The @Newpage command can also appear with a number telling Scribe how many blank pages to leave before resuming text.

```
@Newpage(3)
```

@Newpage is the same thing as @Newpage(0); it leaves no blank pages. These pages are not totally blank; they do have running headers. For completely blank pages, just put blank paper

in your document at that point and increase the page number with @Set if necessary. (See also the description of @Blankpage on page 77.)

Normally, when Scribe is processing a manuscript file and a page becomes full, it simply starts a new page. Some environments, however, have definitions that cause their text to be kept together on a page, sometimes moved if necessary.

The environments Example and Display have the property that their contents are never split across a page unless you explicitly permit it. If an Example environment or a Display environment does not fit on a page, then Scribe starts a new page (leaving white space at the bottom of the old one) and places the entire environment at the top of the new page. We call this property *Group* and call Example and Display *grouped environments.*

Sometimes grouped environments are very long or contain several groups of lines where it is acceptable to break the environment across a page. The @Hinge command tells Scribe when it is allowed to break the environment and start a new page. @Hinge always causes one blank line to be inserted in the document when the page does not need to break.

Figures and Tables have the property that, if they do not fit on the current page, Scribe is free to move them to a new page after first filling the current page. We call this property *Float* and refer to Figure and Table as *floated environments.*

10.6 Widows and Orphans

The word "widow" is defined as the final line of a paragraph appearing at the top of a fresh page instead of appearing with the rest of the paragraph at the bottom of the previous page. An "orphan" is the first line of a paragraph appearing alone at the bottom of a page. It is generally agreed that widows and orphans are ugly, but there is no well-defined method for getting rid of them.

Careful publishers normally eliminate widows by working with the author and copy editor to rewrite the text so that no widows occur. They find it equally ugly to have pages of differing length, so the obvious solution (jamming an extra line at the bottom of one page or the top of another page) is not acceptable. If the widowed line has a footnote attached to it, then it cannot be forced to a different page, because there wouldn't be room for the footnote on the other page.

Scribe gives you four options with respect to widows; you select among them by using the WidowAction @Style parameter.

@Style(WidowAction=Force)
: Forces widow lines to appear at the bottom of the page instead of the top of the next page, even if they intrude slightly into the bottom margin. When the widow line has a footnote, it is put at the top of the next page regardless of the setting of this Style parameter. (This value is the default setting in most document types.)

@Style(WidowAction=Warn)
: Prints a warning message but does not attempt to move the widow line. This action allows you to repair widow lines manually by rewriting the text.

@Style(WidowAction=ForceWarn)

Forces widow lines to appear at the bottom of the page instead of the top of the next page, even if they intrude slightly into the bottom margin, and prints a warning message.

@Style(WidowAction=Ignore)

Pays no attention to whether a line is a widow line and produces no warnings or errors.

Scribe offers no automatic method of controlling orphans. The best way for you to handle orphan lines in your document is by using either the Group environment or the Group or Need environment attribute. The Group environment is used when you want to ensure that the text of an environment is kept together on one page. If, for example, you had an example that was not to be broken across a page, you would surround the Example environment with the Group environment:

```
@Begin(Group)
@Begin(Example)
Text of the example to be printed on one page
@End(Example)
@End(Group)
```

If text that is delimited by the Group environment fits on the current output page, then Scribe essentially ignores the Group environment. If, on the other hand, the text does not all fit on the current page, then Scribe leaves the rest of the current page blank and prints the text at the top of the next page.

The Group and Need environment attributes are discussed in detail in Chapter 15. Very briefly, the Group environment attribute simulates the Group environment for the environment for which it's specified. The Need environment attribute tells Scribe that the environment can not be printed on the current page unless *n* amount of space is left on it. You can specify *n* in any vertical distance.

Chapter Eleven
Mathematical Output

It is easy to specify complicated mathematical equations and formulas in documents prepared by Scribe with the Scribe mathematical facility. This facility has been designed to satisfy the needs of most users in a simple way. It has been used to produce highly complex equations and mathematical manuscripts. The range of conceivable formulas makes it impossible to give complete directions for specifying them in this manual. Please contact your ***DBA*** if you need assistance. A wide variety of mathematical examples and how to construct them is shown in Appendix G.

11.1 Output Devices

The capabilities of the new mathematical facility may be used with any device. However, only devices capable of fine character positioning and having suitable alphabets will produce true mathematical output. Other devices will usually produce one-character substitutions for special characters and are suitable only for proofing your copy. In general, photocomposers and page printers can be used for quality mathematical output.

11.2 Including Mathematics in Your Document

Scribe's mathematical capabilities are not included in the standard document type definitions. If you want to use them, you must include the following command at the beginning of your .MSS file:

@LibraryFile[Mathematics*n*]

where *n* represents the point size of the document's running text. Normally, *n* may take the value 10 or 12. Check with your ***DBA*** to find out which values are available at your site. The @LibraryFile command causes the library file from Scribe's Database that contains all the definitions for the mathematical facility to be read in.

11.3 Mathematical Environments and Forms

Mathematical typesetting in Scribe is accomplished with a combination of built-in features and Database definitions. As a general rule, structures that are font or output device dependent are defined in the Database. Invoking these facilities is easy. You will be using a combination of environments and mathematical forms. As we stated in Chapter 4, forms are advanced Scribe constructs roughly analogous to a subroutine or macro. The complete discussion of forms is included in the *Scribe Database Administrator's Guide*, but you do not need to understand that material to produce math with Scribe. Just follow the instructions in this chapter.

Two new environments have been defined for the mathematical facility. One of them must be used when any of the environments or forms discussed in this chapter are used.

1. **Math:** used for mathematical terms, formulas, equations, *etc.* in running text.
2. **MathDisplay:** used for mathematical terms, formulas, equations, *etc.* in display format.

A formula in the MathDisplay environment generally is taller that the same formula in the Math environment and is surrounded by more white space. It may use taller special characters, higher superscripts, and lower subscripts. The formula $\sum_{x=0}^{5} n$ in running text would be produced by the following manuscript form:

```
@Math<@Sum(From "x=0", To "5") n>
```

The same formula looks like this example

$$\sum_{x=0}^{5} n$$

if it is produced by the following @MathDisplay manuscript form:

```
@MathDisplay<@Sum(From "x=0", To "5") n>
```

Note that this manuscript form is identical to the previous manuscript form except for the mathematical environment used.

11.3.1 Properties of the Mathematical Environments

The Math and MathDisplay environments are essential to mathematical output. If you forget to put a formula in one of those environments, it is likely to look unpredictably wrong in your final document. Except where otherwise noted, definitions for the mathematical facility work as specified *only* within the Math and MathDisplay environments. Use of them in text within other environments may give unpredictable and erroneous results. Both the Math and MathDisplay environments have the properties listed below.

- Spaces in the manuscript file are ignored; they do not cause word breaks or cause spaces to appear in the document. Methods for controlling the spaces that appear in the document are described in Section 11.4.
- Hyphenation is disabled, and text hyphens are not treated specially.
- A "mathematics FontFamily" is used for all characters (except where there is an explicit switch to some other font, such as @r or @b). In the mathematics font, all letters appear in Italics.

- Digits and punctuation marks appear normally, with the exceptions shown in Table 11-1.

Manuscript Form	Document Result	Comments
#	a thin space	See Section 11.4
-	– (minus-sign)	Differences between the hyphen and minus-sign are length and possibly the height above the baseline
*	× (multiplication-sign)	
:	· (centralized dot)	Suitable for indicating multiplication in some formulas: $a \cdot b$
%	% (division-sign)	
'	, (small comma)	Result depends on device
=	=	Font choice matches – and %
+	+	Font choice matches – and %
'	′ (prime-sign)	

Table 11-1: Special Punctuation Characters in the Mathematical Facility

11.3.2 Normal Text in the Mathematical Environments

At times, the special treatment of certain characters, as described in Table 11-1, may be inconvenient. To get the "normal" appearances of these characters from within the Math or MathDisplay environment, use the **r** environment: **@r(*)** or **@r(#)**. Other forms, described in Section 11.5, are available to print "normal" characters, such as an asterisk.

11.4 Spacing

Because spaces are ignored in the Math and MathDisplay environments, control over the spacing of mathematical formulas in the document is achieved by means other than the spaces typed in the .MSS file. Therefore, you must explicitly specify the places where you want spaces to appear.

Besides the usual mechanisms for tabbing, centering, and right- and left-flushing of text, the following spacing specifications are available:

- The # character becomes a thin space (equal in width to the exclamation point character).
- The command @␣ (@ followed by a space) becomes, as always in Scribe, an en space.
- The form **@Quad** becomes a quad space (em space).

Here is an example that uses all the above spacings:

Manuscript Form:

Manuscript Form:

```
@Math(+ # + @␣ + @Quad +)
```

Document Result:

+++ +

11.5 Special Characters

Various ''special characters'' which are not among the letters, numbers, and punctuation marks of the standard (ASCII) set but are used commonly in mathematical text, are available in the mathematical facility. Some of them are provided in the forms described in Section 11.7. Most of them, however, are provided by Scribe forms described in this section.

Capital script letters are provided by the environment **Scr**. Remember that you will only get Script characters if your output device is capable of printing them.

Table 11-2 completely lists the special characters made available on the Apple LaserWriter (POSTSCRIPT) device by the mathematical facility. Tables for other devices which can produce true mathematical output are in Appendix G. Each such character may have several names in informal usage; for instance, the character ''*'' is known as an ''asterisk'' and as ''star''. In the table, we give as many names as we can for ease of reference. In the case of the asterisk, you will find an entry for both Asterisk and Star, but the Star entry will simply be a cross reference to the Asterisk entry. The LibraryFile of mathematical definitions, however, only recognizes one name; that name is given in the second column of the table.

The fourth column in the table indicates the *availability* of the character, which is one of the following possibilities:

- *Normal*: The character supplied should be fully satisfactory for all documents.
- *Substitute*: The character supplied resembles the character requested, but the resemblance may not be completely satisfactory for some documents.
- *Fake*: A character *or character sequence* is supplied. This character or sequence does not resemble the character requested, but should clearly indicate to the reader what character was requested.
- *Blank*: A blank space is supplied.

The same special character may have a different availability on different devices, which is why the special characters for each device are listed in separate tables.

The same special character may have a different availability on different devices, which is why the special characters for each device are listed in separate tables.

Of all the capabilities in Scribe, mathematical output is the most device-dependent. Your site must have a suitable output device, a collection of appropriate fonts, and Scribe Database support for your site's printer/font configuration. If you are trying to get mathematical output and are having difficulty, contact your ***DBA***.

Table 11-2: Special Characters for the Apple LaserWriter (POSTSCRIPT)

Informal name	Scribe name	Example	Availability
Aleph	@Aleph	ℵ	Normal
And	*See "Intersection (logical)"*		
Angle	@Angle	∠	Normal
Approximate equality	@Approx	≈	Normal
Asterisk	@Ast	*	Normal
Back arrow	*See "Left arrow"*		
Bottom	@Bot	⊥	Normal
Bullet (hollow)	@Circ		Blank
Bullet (solid)	@Bullet	•	Normal
C-set	@CSet	**C**	Fake
Circle (small)	*See "Bullet (hollow)" and "Degrees"*		
Degrees	@Degr	°	Normal
Delta	@Delta	Δ	Normal
Divided-by	@Div	/	Normal
Dot-in-circle	@ODot		Blank
Down arrow	@DownArrow	↓	Normal
Empty set	@EmptySet	∅	Normal
Equality	@Eq	=	Normal
Equivalence	@Eqv	≡	Normal
Existential quantifier	@Exists	∃	Normal
For-all	*See "Universal quantifier"*		
Greater	@Gt	>	Substitute
Greater-or-equal	@GtE	≥	Normal
Greater-or-less	@GtLt		Blank
H-bar	@HBar		Blank
In	*See "Membership"*		
Inequality	@Neq	≠	Normal
Infinity	@Infty	∞	Normal
Integers	@ZSet	**Z**	Fake
Intersection (logical)	@And	∧	Normal
Intersection (set)	@Inter	∩	Normal
Intersection (square)	@SqInter		Blank
Left angle bracket	@LAngle	⟨	Normal
Left arrow	@LeftArrow	←	Normal
Less	@Lt	<	Substitute
Less-or-equal	@LtE	≤	Normal
Less-or-greater	@LtGt		Blank
Membership	@In	∈	Normal
Minus	@Sub	−	Normal
Minus-in-circle	@Ominus		Blank
Minus-or-plus	@Mp		Blank
Much-greater	@MuchGt		Blank
Much-less	@MuchLt		Blank
Nabla	@Nabla	∇	Normal
Natural numbers	@NSet	**N**	Fake
Negation (logical)	@Not	¬	Normal
Non-equivalence	@NEqv	≢	Substitute

Non-membership	@NotIn	$\notin$	Normal
Not	*See "Negation (logical)"*		
Not-in	*See "Non-membership"*		
Operator (generic)	@Op	**OP**	Fake
Or	*See "Union (logical)"*		
Partial derivative	@Partial	∂	Normal
Planck's constant	*See "H-bar"*		
Plus	@Add	$+$	Normal
Plus-in-circle	@OPlus	$\oplus$	Normal
Plus-in-u	@UPlus		Blank
Plus-or-minus	@Pm	$\pm$	Normal
Q.E.D.	@Qed		Blank
Q-set	@QSet	**Q**	Fake
Right angle bracket	@RAngle	$\rangle$	Normal
Rational numbers	@RSet	**R**	Fake
Right arrow	@RightArrow	$\rightarrow$	Normal
Similarity	@Similar	$\sim$	Normal
Similar or equal	@SimEq	$\approx$	Normal
Slash	*See "Divided-by"*		
Slash-in-circle	@ODiv		Blank
Square	*See "Q.E.D."*		
Star	*See "Asterisk"*		
Subset	@Subset	$\subseteq$	Normal
Subset (proper)	@PrSubset	$\subset$	Normal
Superset	@Supset	$\supseteq$	Normal
Superset (proper)	@PrSupset	$\supset$	Normal
Times	@Mult	$\times$	Normal
Times-in-circle	@OTimes	$\otimes$	Normal
Top	@Top		Blank
Triangle	*See "Nabla" and "Delta"*		
Union (logical)	@Or	$\vee$	Normal
Union (set)	@Union	$\cup$	Normal
Union (square)	@SqUnion		Blank
Universal quantifier	@ForAll	$\forall$	Normal
Up arrow	@UpArrow	$\uparrow$	Normal
Vertical bar	@VBar	$\mid$	Normal
Vertical bar (double)	@DVBar	$\parallel$	Substitute

11.6 Other Special Notations

11.6.1 Common Mathematical Text

Occasionally, it is desirable to get normal, non-italic text into a formula. Two general examples are:

- when using standard operators, functions, and other words, such as **cos** and **lim**. These should ordinarily be typeset in a Roman face. The mathematical facility provides a set of forms to achieve the Roman font; see Table 11-3.
- when using other short phrases in the middle of a formula, such as **if**, **and**, or **otherwise**. To get the right spacing and font for these phrases, use the environment **Sr** (which stands for Spaced Roman), as shown in this example:

Manuscript Form:

```
@Abs(x) = -x @Sr[if] x@Lt 0,@Quad x @Sr[otherwise]
```

Document Result:

$|x| = -x$ if $x<0$, x otherwise

There are some words and phrases common in mathematical text that are ordinarily expected to be set in normal (Roman) type to set them off from symbols, that are in Italic, but for which it is inconvenient to specify the font switching and spacing explicitly by using the Sr environment. A set of forms for such common text have been supplied. They are listed below, broken into two groups: Those that take an argument and those that do not.

Table 11-3: Common Mathematical Text

If you want:	Use:
arctg	@Arctg
atan	@Atan
cos	@Cos
cot	@Cot
csc	@Csc
deg	@Deg
det	@Det
inf	@Inf
lg	@Lg
lim	@Lim
liminf	@Liminf
limsup	@Limsup
ln	@Ln
log	@Log
$\log_2$	@Log2
max	@Max
min	@Min
mod	@Mod
sin	@Sin
sup	@Sup
tan	@Tan
tg	@Tg
trace	@Trace

O(*text*)	@BigO(text)
Ω(*text*)	@Omega(text)
Θ(*text*)	@Theta(text)
det(*text*)	@Detrm(text)
exp(*text*)	@Exp(text)
gcd(*text*)	@Gcd(text)
⌈*text*⌉	@Ceiling(text)
⌊*text*⌋	@Floor(text)
|*text*|	@Abs(text)
‖*text*‖	@Norm(text)
$\sqrt{text}$	@Sqrt(text)

Each of the forms in the second group above will produce ugly and possibly unacceptable output if its parameter is unusually tall. Thus, text that uses the multi-line forms described in Section 11.7 is not suitable as a parameter to the forms of this section. This restriction will be removed in a future release.

11.6.2 Ellipsis

There are two forms for specifying the ellipsis, a series of three dots. **@LDots** produces three dots on the baseline, and **@CDots** produces three dots raised to align with such things as plus-signs and minus-signs. These forms can be used *outside* of the Math or MathDisplay environment as shown here:

Manuscript Form:

```
a@Down(1), a@Down(2), @LDots, a@Down(k)

1 + 2 + @CDots + n
```

Document Result:

$a_1, a_2, \ldots, a_k$

$1 + 2 + \cdots + n$

11.6.3 Numeration

There are also four forms for numeration. They are among the few forms from the mathematical facility that can be used without being inside the Math or MathDisplay environment. Examples of them are given below:

Manuscript Form:

```
1@st, 2@nd, 3@rd, 4@th.
```

Document Result:

1st, 2nd, 3rd, 4th.

11.6.4 Miscellaneous

In mathematical text, it is preferable to use the **@Up** and **@Down** environments rather than @+ and @-. There are certain special characters that require different sizes depending on whether they are scripted or not; for example, the integral sign. @Up and @Down will produce the appropriate size character.

The assignment "operator" :=, common in computer program text, can be obtained with the form **@Get**.

The following embellishments can (at present) be used only on single characters. **@Vec** and **@Overline** can be used only on lower case letters; **@OverlineCap** can be used only on upper case letters and digits.

$\vec{v}$ @Vec(v)

$\overline{v}$ @Overline(v)

$\overline{V}$ @OverlineCap(@r(V))

11.7 Multi-Line Formulas

The mathematical facility provides forms for constructing multi-line formulas such as summations, integrals, fractions, and binomial coefficients. The forms all have a similar syntax in that they all take delimited strings as values of their specific arguments. The syntax of the individual forms is shown below, followed by examples of each form.

@Sum[From=*delimited-value***, To=***delimited-value***]**

@Prod[From=*delimited-value***, To=***delimited-value***]**

@Int[From=*delimited-value***, To=***delimited-value***]**

@Over[Num=*delimited-value***, Denom=***delimited-value***]**

@Choose[From=*delimited-value***, Chosen=***delimited-value***]**

@Brace[Top=*delimited-value***, Bot=***delimited-value***]**

@Limit[As=*delimited-value***]**

@Ss[Sub=*delimited-value***, Super=***delimited-value***]**

@SmallFraction[Num=*delimited-value***, Denom=***delimited-value***]**

An example of each form that produces a multi-line formula follows. Use in the

MathDisplay environment is shown first, followed by the same formulas in the Math environment.

Examples in the MathDisplay Environment

Manuscript Form: Document Result:

```
@Sum(From"n=1",To"m")n
```

$$\sum_{n=1}^{m} n$$

```
@Prod(From "p=2", To "@Infty") p
```

$$\prod_{p=2}^{\infty} p$$

```
@Int(From "-1", To "1") xdx
```

$$\int_{-1}^{1} x dx$$

```
@Over(Num "p", Denom "(p+1)(p+2)")
```

$$\frac{p}{(p+1)(p+2)}$$

```
@Choose(From "n", Chosen "n-m")
```

$$\binom{n}{n-m}$$

```
@g(d)@Down(ij)=@Brace(
  Top "1 @Quad @Sr[if]i=j",
  Bot "0 @Quad @Sr[otherwise]")
```

$$\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{otherwise} \end{cases}$$

```
@Limit(As "x @RightArrow @Infty")@~
  f@Down(i)(x)
```

$$\lim_{x \to \infty} f_i(x)$$

```
a@Ss(Sub "j", Super "i")
```

$$a_j^i$$

```
@SmallFraction(Num "1", Denom "2")
```

$$\tfrac{1}{2}$$

Examples in the Math Environment

Manuscript Form:

Document Result:

```
@Sum(From "n=1", To "m") n
```

$$\sum_{n=1}^{m} n$$

```
@Prod(From "p=2", To "@Infty") p
```

$$\prod_{p=2}^{\infty} p$$

```
@Int(From "-1", To "1") xdx
```

$$\int_{-1}^{1} x\,dx$$

```
@Over(Num "p", Denom "(p+1)(p+2)")
```

$$\frac{p}{(p+1)(p+2)}$$

```
@Choose(From "n", Chosen "n-m")
```

$$\binom{n}{n-m}$$

```
@g(d)@Down(ij) = @Brace(
  Top "1 @Quad @Sr[if]i = j",
  Bot "0 @Quad @Sr[otherwise]")
```

$$\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{otherwise} \end{cases}$$

```
@Limit(As "x @RightArrow @Infty")
  f@Down(i) (x)
```

$$\lim_{x \to \infty} f_i(x)$$

```
a@Ss(Sub "j", Super "i")
```

$$a_j^i$$

```
@SmallFraction(Num "1", Denom "2")
```

$$\frac{1}{2}$$

The result of one multi-line formula can be the argument to another:

Manuscript Form:

```
@Over(Num "x+1", Denom "y + @Over(Num [x-2], Denom [x])")
```

Document Result:

$$\cfrac{x+1}{y+\cfrac{x-2}{x}}$$

Chapter Twelve
Producing Bibliographies with Scribe

A scholarly paper normally includes in its text *citations* to the literature and a *Bibliography* or *list of references* at the end. The citations refer to entries in the Bibliography, which give complete data about the cited publications.

Citations can have many different "correct" formats. Some journals want a number in square brackets: [7]. Others want a superscripted number,[7] like a footnote. Others want a parenthesized reference to the primary author and the year of publication: (Knuth 1968) and some publishers use the first few letters of the author's name followed by the last two digits of the year: [Knut68] or [KNU68].

Formats for bibliography entries have even more variety than citations. Some authorities want journal titles abbreviated, others want them spelled out. One journal would like you to use the abbreviation *"J. ACM"* to stand for the *Journal of the Association for Computing Machinery*, while another journal wants you to use the abbreviation "*J. Assoc. Comp. Mch.*" to stand for the same thing. A third journal might want dates in parentheses, another would surround them with commas. Volume and issue numbers might be italicized or boldface or in parentheses.

Experienced authors tend to handle these nuisance variations in bibliography format rules by ignoring them and using the same format for all of the papers that they submit, passing to the journal editors the job of cleaning up the Bibliography. Inexperienced authors tend to spend days getting every last comma in the right place. To help bring both of these lamentable practices to an end, Scribe provides an automatic Bibliography and citation system to take care of as many of the grubby details as possible.

To use Scribe's Bibliography mechanism, you need two things — a file containing bibliography information in a special format and @Cite command in your text. The scheme is very similar to that for cross referencing (see Chapter 7) in that you use a code word in the @Cite command to refer to items in the special bibliography file. Scribe finds the items being referred to and replaces the @Cite commands with the correct citations in the correct format.

Furthermore, once you have cited something, Scribe automatically adds the reference to the Bibliography at the end of your paper. The Bibliography is printed either in alphabetical order (by primary author) or in numerical order (by citation sequence), depending on the reference format that you have chosen.

To use the Bibliography mechanism in Scribe, you have to do these things:

- Get access to a bibliography database file, and tell Scribe its filename. You can use someone else's, or you can build your own. Most people will want to build a personal database containing an entry for every paper, article, or book that is relevant to their work. Section 12.5 tells you how to make your very own bibliographic database.

  Scribe always looks first for a bibliography file with the same name as your .MSS file and extension .BIB. You can tell Scribe to look in a different file to find the references.

  The @Use command with a Bibliography parameter tells Scribe which bibliography file to read. For example,

  ```
  @Use(Bibliography="GENERL.BIB")
  ```

- Decide on a reference style. If you are submitting a paper to a journal and Scribe knows about that journal's format, use it. Section 12.2 tells you about your options in choosing a reference style.

- For the reference items you want in the Bibliography, put @Cite commands with codewords into your manuscript file in the spots where citations should appear.

Scribe will do the rest.

12.1 An Example

Let's look at a simple example. We will write a one-sentence paper with one bibliographic reference and show how all the pieces fit together.

Manuscript Form:

```
The Fortran I compiler@Cite(Fortran) is
probably the granddaddy of optimizing compilers.
```

The document file has the following sequence as a result.

Document Result:

The Fortran I compiler [Backus 57] is probably the granddaddy of optimizing compilers.

Whenever Scribe sees an @Cite command, it knows to create a "References" section in the document. It does this step automatically at the end of the document, giving it the title "References".

Manuscript Form:

```
@Device(Postscript)
@Style(References=STDAlphabetic)
@Heading(The Paper)
The Fortran I compiler@Cite(Fortran) is
```

```
probably the granddaddy of optimizing compilers.
```

Document Result:

The Paper

The Fortran I compiler [Backus 57] is probably the granddaddy of optimizing compilers.

References

[Backus 57] Backus, J.W. et al.
The FORTRAN Automatic Coding System.
In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957.
Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.

12.2 About the Format of Citations and References

A citation is placed in your text at each spot where you use an @Cite command. The style of the citation is determined by the reference format that you have chosen. For each different work cited, a bibliography entry appears at the end of your file. The format and layout of this bibliography entry is also determined by the reference format that you have chosen.

When you don't ask for a reference format, Scribe uses the one named "STDNumeric". You can ask for a different format by providing a "References" parameter in an @Style command (described in detail in Section 4.5). Thus, the following command requests references in IEEE format.

```
@Style(References=IEEE)
```

A reference format is defined by an entry in Scribe's Database. The standard version Database defines the reference formats listed below. As of the time of this manual (15 July 1985), the bibliography formats have not been made entirely consistent with one another. They were all written at different sites by different people, and not all of them implement all forms of all of the reference types. For definitive documentation, look at the Database files with file type .REF. Some of the .REF files call library files. For those files, look in the file FILENA.LIB, where FILENA is the first six characters of the parameter inside the @LibraryFile command.

1APA	Similar to the APA format except that it contains an Annote field that is treated as a Comment.
1APADraft	Similar to the 1APA format except that it is double-spaced.
AnnAPA	Similar to the 1APA format except that the Annote field is treated as text.
AnnAPADraft	Similar to the 1APADraft format except that the Annote field is treated as text.

AnnotedSTDAlphabetic
: Similar to the STDAlphabetic format except it includes annotations and has unfilled lines.

AnnotedSTDIdentifier
: Similar to the STDIdentifier format except it includes annotations and has unfilled lines.

AnnotedSTDNumeric
: Similar to the STDNumeric format except it includes annotations and has unfilled lines.

AnnSTDAlphabetic
: Similar to the STDAlphabetic format except it includes annotations and has unfilled lines.

AnnSTDNumeric
: Similar to the STDNumeric format except it includes annotations and has unfilled lines.

APA
: American Psychological Association format. Inconsistent and irregular format, but we try to approximate it.

APADraft
: Similar to the APA format except that it is triple-spaced.

CACM
: Packed format, numeric citation, alphabetical order.

ClosedAlphabetic
: Similar to the STDAlphabetic format.

ClosedNumeric
: Similar to the STDNumeric format.

IEEE
: Packed format, numeric superscript citation, in citation order. This format is acceptable for IEEE *Computer* magazine. Many of the other IEEE journals use different reference formats, although we suspect that they claim to have a uniform standard.

IPL
: Information Processing Letters.

SIAM
: Society for Industrial and Applied Mathematics.

STDAlphabetic
: Open format, alphabetic citation.

STDIdentifier
: Open format, reference identifier for citations rather that a generated label.

STDNumeric
: Open format, numeric citation.

Citations placed in the text because of your @Cite commands are forward references. Section 7.3 on page 69 explains the nature of forward references. What this statement means in the context of bibliographic citations is that if you change reference formats, you have to run Scribe over the manuscript file twice before the document will show the new citation style everywhere.

To give you an idea of the various formats available, let's take two bibliography entries and print them in several of the various formats.

We introduce our examples by showing the way that they would look in the default STDNumeric format. Citations would be [2] and [1], and the bibliography entries would look like this example:

[1] Backus, J.W. et al.
The FORTRAN Automatic Coding System.
In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957.
Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.

[2] Knuth, D. E.
The Art of Computer Programming. Volume I: *Fundamental Algorithms*.
Addison-Wesley, 1968.

If we were to print these same references in IEEE[1] format, both the citations and the order of the references would change. In IEEE format, our citations would be [1] and [2], superscripted numbers. In use, they would be put singly[1] or in pairs[2,3] at the appropriate point in the document, and the corresponding bibliography entries would look like this example:

1. Knuth, D. E. *The Art of Computer Programming*. Volume I: *Fundamental Algorithms*. Addison-Wesley, 1968.
2. Backus, J.W. et al. The FORTRAN Automatic Coding System. In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957. Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.

Notice that the Bibliography is in citation order, not in alphabetical order.

If we were to change to CACM[2] format, then the citations would again be numbers in brackets, [2] and [1]. The references would now be in alphabetical order and look like this example:

1. Backus, J.W. et al. The FORTRAN Automatic Coding System. In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957. Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.

2. Knuth, D. E. *The Art of Computer Programming*. Volume I: *Fundamental Algorithms*. Addison-Wesley, 1968.

If we change to STDAlphabetic format, our citations appear as [Knuth 68] and [Backus 57], and our references now look like this example:

[Backus 57] J.W. Backus et al.
The FORTRAN Automatic Coding System.
In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957.
Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.

---

[1] The Institute for Electrical and Electronic Engineers (IEEE) publishes some 20 journals in the computer and electronics field.

[2] The *Communications of the Association for Computing Machinery* is a widely-read journal in the computer science field.

[Knuth 68] Knuth, D. E.
The Art of Computer Programming. Volume I:
Fundamental Algorithms.
quotation-Wesley, 1968.

The American Psychological Association's official reference format is very irregular and not at all suitable to automated citation. Nevertheless, Scribe provides a format that comes close to one form of the APA requirement. If you changed reference format to APA, our citations would come out as (Knuth, 1968) and (Backus, 1957), and the references would look like this:

> Backus, J.W. et al. The FORTRAN Automatic Coding System. In *Proceedings of the Western Joint Computer Conference*, pages 188-198. AFIPS, February, 1957. Also in S. Rosen, editor, *Programming Systems and Languages*, McGraw-Hill (1967), pages 29-47.
>
> Knuth, D. E. *The Art of Computer Programming*. Volume I: *Fundamental Algorithms*. Addison-Wesley, 1968.

APA references are in alphabetical order by the Key field and by year within author.

12.3 The @Cite Command

The @Cite command uses a codeword to find the item you want to cite and puts its citation into the document. The format of those citations, as we explained in the previous section, is dependent on the reference format that you have chosen.

In an @Cite command, you must use the codeword that was chosen for the item by the person who built the bibliographic database. If you are using a personal database that you built yourself, then of course you know the codewords for the items. In general, we expect people to use common, shared databases. You need to obtain a list of the items and their codewords for each reference that you want to cite. The @Cite command should never be separated from the word it follows. If a blank is inserted, the space added may be subject to justification and the citation may float away from the point at which it was inserted.

If the bibliographic database is small and if it is not a "binary" file, you can just print it and look at it. If the database is large or if it has been encoded into a special binary file, you will need help. Every database should have people responsible for its maintenance; talk to them.

The standard document type, "Bibliography", produces a Bibliography without an accompanying document. This two-line manuscript file creates a Bibliography:

```
@Make[Bibliography]
@Use[Bibliography "MINE.BIB"]
```

Once you have found the codeword for a reference, you must use it in an @Cite command to produce a citation. An @Cite command can contain one or more codewords:

Manuscript Form:

```
Volka and James@Cite(Bogosity) describe the
                          origin of "bogus."
...are made by Hoolinan and Quincy@Cite(Hool64,Quincy).
```

Scribe produces a multiple citation formatted according to the reference format when you give it more than one codeword in a single @Cite command:

Document Result:

Volka and James [9] describe the origin of ''bogus.''

...are made by Hoolinan and Quincy [3,8].

Many journals require that multiple numbers in a single citation be in increasing order or alphabetical order, but Scribe does not sort them for you. You have to arrange the codewords in the appropriate order in the @Cite command itself.

The correct use of citations in sentences is a matter not of technology but of writing style. Since style is a matter of personal preference and local custom, it is not the place of this manual to suggest or dictate. For enlightened narrative discussion, copious examples, dogma, and entertainment, we refer the reader to Mary-Claire van Leunen's wonderful book, *A Handbook for Scholars* (Alfred A. Knopf, 1978).[3] (No student should begin writing a thesis without having read it cover to cover.)

Any delimited text inserted in an @Cite command after the codeword is actually included in the document's text. For example, this command:

Manuscript Form:

```
Volka and James@Cite(Bogosity ", p. 15") describes the
```

produces this output:

Document Result:

Volka and James [9, p. 15] describes the

You can flag selected entries for inclusion in the Bibliography without actually citing them in the document by using the @CiteMark command:

```
@CiteMark(Jones4)
```

@CiteMark has precisely the same effect as the @Cite command, except that it does not generate a citation in the text. The entry still appears in the Bibliography.

A generalized version of what you do with @CiteMark would be to force *all* of the entries in your bibliography database file, cited or not, to appear in your document's Bibliography. The BibSelect parameter in the @Style command requests this inclusion.

---

[3] The *Scribe User Manual* does not have a Bibliography, as it is not a piece of scholarly writing, so we have imbedded this reference instead of using a @Cite command.

```
@Style(BibSelect=Complete)
```

This @Style command should be at the beginning of your manuscript file, before the first text.

12.4 Placing the References Section

Scribe normally prints the Bibliography at the end of your document automatically and includes in it everything that was cited in the text with an @Cite command. It receives a heading, "References", automatically.

You can change either the placement of the Bibliography or the heading it receives. In fact, to change either, you must specify both of these commands:

```
@Unnumbered[References]
@Bibliography
```

This example illustrates how to move the Bibliography. The @Unnumbered command specifies the heading for the Bibliography. (This way, the heading appears in the Table of Contents; otherwise it doesn't.) The @Bibliography command tells Scribe to create the Bibliography at the current spot in the document. You could use @Bibliography to place the Bibliography before the end of the document (for example, before an appendix). However, only @Cite commands that occur before the @Bibliography command cause items to appear in the Bibliography.

To leave the Bibliography at the end of the document as usual but to change the title of it to "Bibliography", these commands are necessary at the end of the document:

```
@UnNumbered(Bibliography)
@Bibliography
```

Books often need a References section for each chapter or section. Scribe can produce more than one reference list. You need to use an @Style command with the MultipleBibliography parameter at the beginning of the manuscript file:

```
@Style(MultipleBibliography On)
```

Then you need a heading for each reference section and an @Bibliography command for each reference section, at the appropriate place in the manuscript. Each reference section includes the items for all the @Cite commands since the previous @Bibliography command.

12.5 Bibliography Database Files

A database is a collection of information organized in such a way that you can find things. A computer file with seven telephone numbers typed in to it is just as much a database as the Social Security Administration's billion-character wonders. Let's just talk about bibliographic databases, though.

12.5.1 Its Organization and Contents

The simplest form of bibliographic database would be a file into which you had typed the information about some books and articles, with a label in front of each one. But Scribe, being a dumb computer program, wouldn't be able to make much use of that. You must pick the entries apart like cooked lobsters, telling Scribe what every field should be. The database entries for the two examples that we used throughout Section 12.2 look like this example:

```
@InProceedings(Fortran,
        Author="J.W. Backus et al",Pages="188-198",
        Title="The FORTRAN Automatic Coding System",
        Month="February",Key="Backus",Year="1957",
        Publisher="AFIPS",BookTitle="Proceedings of
        the Western Joint Computer Conference",
        Note="Also in S. Rosen, editor, @i[Programming
              Systems and Languages] McGraw-Hill
              (1967), pages 29-47")

@Book(Volume1,Author="Knuth, D. E.",Key="Knuth",
        Title="Fundamental Algorithms",
        Series="The Art of Computer Programming",Volume="I",
        Publisher="Addison-Wesley", Year="1968")
```

Look at them carefully. Notice that they define values for various things like "Publisher" and "Author", which are pretty obvious, but also for things like "Key", which is probably not so obvious.

Every bibliography entry must have at least three things:

- A *type*. Our examples above are of type "inproceedings" and "book".
- A *codeword*. Our examples have "Fortran" and "Volume1" for their codewords.
- A *key*, which is used for alphabetization. The keys in our examples above are "Backus" and "Knuth".

The codeword is necessary to find the entry in the database (it must match the codeword used in the @Cite command), the key is needed to determine the correct alphabetization, and the type is needed to determine how to format it.

Here we should confess that a key is not actually required for a bibliography entry, although it is very often included. If it is missing, the codeword is used for alphabetizing the entry. We still suggest that you use a key.

Of course, to be useful, an entry should have other fields, like author and title. Every type of entry needs a different set of fields. The "book" type shown above uses a "series" field and a "volume" field. "Series" would probably be meaningless in a journal article, but "volume" means a lot. On the other hand, a "journal" field in the entry for a book would not be of much use.

12.5.2 Building Your Bibliographic Database

Your own bibliographic database file is an ASCII file that you build with your favorite text editor. It contains only bibliography entries and possibly @String commands to define abbreviations. Put no other Scribe commands in this file. Because Scribe sorts your Bibliography into the appropriate alphabetic or numeric sequence, any other commands in the file would come out in strange places.

12.5.2.1 Strategy

To enter a reference in a database, you have to do these three things:

1. Classify it. Figure out what kind of a reference it is. Book? Article? Conference proceedings?
2. Find out what fields are needed. Decide upon their value. What is the title? Who is the author? What is the volume number?
3. Get it into the computer. If you are building a private database, you would type it in using a text editor into a garden-variety file, in a format similar to the one used in our examples above. If you are adding it to some public database, you would use an entry program devised by the keepers of that database; talk to them.

(Some people have written form-filling programs to help with the tedious business of commas, field names, and delimiters. Ask around at your site!)

12.5.2.2 Classification

If you think about classification too hard, the task becomes almost impossible. Since you are classifying a reference only to get its format to come out correctly, you don't have to worry about the fine details of classification that give librarians ulcers. Scribe has adopted the naming and classification scheme suggested by van Leunen, and we strongly suggest that you find a copy of her book if you intend to make serious use of the Scribe Bibliography system.

In van Leunen's book, there are 11 "easy" types of reference,[4] 8 "hard" types of reference,[5] and 7 "obscure" types.[6] Don't think too much — just look over the following list and figure out which category best suits your reference.

Scribe requires that you classify every reference into one of the following categories. The field names ("Volume", "Key", and so forth) are defined in detail in the next section. This list is general; the fields listed as required are those required for all reference formats. The other fields listed are optional based on the reference format. Appendix E.9 lists the required and optional fields for each reference format individually.

Article An article from an academic journal or a magazine. The required fields

---

[4] Books, compilations, collections, editions with editors, translations, multivolume works, journal articles, reviews, pamphlets, works not yet published, and theses.

[5] Archives, works published by their author, conference proceedings, correspondence, court cases, government documents, microfilm and microfiche, newspapers, and reference works.

[6] Physical buildings, works of art, named computer programs, movies, musical works, radio and television programs, and unpublished interviews.

are **Title** and **Journal**. The optional fields are **Note**, **Number**, **Month**, **Pages**, **Volume**, **Author**, **Year**, **FullAuthor**, **Date**, and **Key**.

Book Something published on its own, usually by a publishing house that is not the same as the author. *cf.* TechReport. The required fields are **Title** and **Publisher**. The optional fields are **Key**, **Author**, **Year**, **Series**, **Volume**, **Address**, **Note**, **Editors**, **Editor**, **Number**, **HowPublished**, **FullAuthor**, and **Date**.

Booklet Something published and bound, but having neither an explicitly-named publisher nor a sponsoring institution. Very few scholarly papers reference booklets. The only required field is **Title**. The optional fields are **Author**, **Key**, **Address**, **HowPublished**, **Note**, **Publisher**, **Month**, **Year**, and **FullAuthor**.

Conference The conference name. The required fields are **Author**, **Title**, and **Year**. The optional fields are **Date**, **Meeting**, **Month**, **Note**, **Organization**, and **Society**.

InBook If you want to refer to a piece of a book, rather than to the entire book, then make your entry be of type **InBook**. The required fields are **Title** and **Publisher**. The optional fields are **Series**, **Volume**, **Address**, **Chapter**, **Pages**, and **Note**, **Author**, **Editor**, **Key**, **Year**, **BookTitle**, **Editors**, **FullAuthor**, **HowPublished**, **Edition**, **Date**, and **Number**.

InCollection Something composed of papers or chapters previously published elsewhere. The required fields are **Title**, **BookTitle**, **Publisher**, and **Year**. The optional fields are **Address**, **Chapter**, **Author**, **Key**, **Editor**, **Note**, **Pages**, **Series**, **Volume**, **Editors**, **Number**, and **FullAuthor**.

InProceedings A reference to a paper in a conference proceedings or the like. The required fields are **Title** and **Year**. The optional fields are **Key**, **Author**, **Organization**, **Editors**, **BookTitle**, **Editor**, **Address**, **Pages**, **Month**, and **Note**, and **FullAuthor**.

Manual An instruction manual or piece of technical documentation. The required field is **Title**. The optional fields are **Author**, **Address**, **Edition**, **Key**, **Year**, **Organization**, **Note**, **Date**, and **FullOrganization**.

MastersThesis A Masters' thesis. The required fields are **School** and **Title**. The optional fields are **Month**, **Note**, **Year**, **Key**, **Author**, **Date**, **Address**, and **FullAuthor**.

Misc Any category not mentioned in this list. There are no required fields. The optional fields are **Author**, **HowPublished**, **Note**, **Title**, **Key**, and **FullAuthor**.

PhDThesis A Ph.D. thesis. The required fields are **Title** and **School**. The optional fields are **Month**, **Note**, **Key**, **Author**, **Year**, **Address**, **Date**, and **FullAuthor**.

Proceedings The proceedings of a conference or some similar document. The identifying characteristics of a **Proceedings** are that its publisher and author are identical and often no editor's name appears. There are no required fields. The optional fields are **Year**, **Key**, **Publisher**, **Editor**, **Organization**, **Title**, **Address**, **Note**, **Date**, **Pages**, **Month**, and **Editors**.

TechReport A technical report is similar to a book, except that it is published by a research institution instead of by a publisher and that it usually has an assigned "report number". Use your judgment in deciding whether some particular reference is a book or a technical report. A thesis is neither.

The required field is **Title**. The optional fields are **Institution, Year, Author, Key, Number, Month, Type, Note, Date, Address,** and **FullAuthor.**

UnPublished Some paper that is in preparation or that has been printed but not published. Many random documents fall into this category. There are no required fields. The optional fields are **Key, Author, Title, Note, Date, Year, Month,** and **FullAuthor.**

12.5.2.3 Field Names

In the previous section, we gave you lists of the required and optional fields for various bibliography entry types. Now we will tell you what they mean. Not every field is used in every bibliography entry type.

All entries are specified in the same syntax:

Name="value"

You can leave out the equals sign and, if the value is a number, you can omit the delimiters. These three forms are all equivalent to one another:

```
Year="1984"
Year "1984"
Year=1984
```

If a field is long, you can continue it across more than one line:

```
Author="Davis, Miles",  Title="In a Silent
                Way"
```

Very well; let's get on with the list of fields: (This information is repeated in tabular form in Appendix E; see page 237.)

Address The address of the publisher. (By convention, the address for a major publisher is just a city name. For minor publishers, a full address is in order.)

Author The name(s) of the author or authors, in the order and style that you would like them printed out. If you need to abbreviate the author's name for some journals and keep it intact for others, you can use a FullAuthor field as well.

Annote A short annotation, one or two sentences at most. It is optional for any bibliography entry type. The annotation appears only in the various APA and STD reference formats.

BookTitle The title of the book containing the chapter or article being cited. (For example, with **InBook**, Title is the title of the chapter and **BookTitle** is the title of the book.)

Chapter When you are referring to something that is published as part of a larger work, it helps to pinpoint the particular piece of the larger work. You can use either Chapter or Pages, not both, to do this pinpointing.

Date The date of the classification.

Edition The manual's edition number.

Editor The name of the editor of **Proceedings, InProceedings, Book, InBook,** and **InCollection**. The names appear exactly as entered in this field.

Editors The name of the editors of **Proceedings, InProceedings, Book, InBook,** and **InCollection.** The names appear exactly as entered in this field.

FullAuthor The "full" name of the author for bibliography types and reference formats permitting first names rather than just initials. FullAuthor is always optional. You can include both Author and FullAuthor fields in any bibliography entry.

FullOrganization The "full" name of the organization for mailing purposes.

HowPublished When you are citing something that has not been formally published, then you don't specify a publisher, but you explain how this thing came to be in print. A reference entry for a canoeing guide, for example, might contain the field:

```
HowPublished="A Pennsylvania AYH Publication"
```

Institution For technical reports, one provides the name of the sponsoring institution instead of the name of a publisher.

```
Institution="Carnegie-Mellon University"
```

Journal The name of the journal or an abbreviation for it. The reference database contains definitions for the abbreviations for all the common journals in the field covered by that journal. When no abbreviation is available for a journal, you must either type in the name of the journal as a delimited string, in the exact format you want it printed in, or, in a personal database, use @String to define an abbreviation for it.

Key The Key field is normally the primary author's last name. It is used as the sort key for alphabetization of the Bibliography, and it is used as the key author's name in reference formats that need one. The value for Key should not be an abbreviation defined with an @String command.

Meeting The name of the meeting. Used with the value of the **Society** field name.

Month The abbreviation for the month:

```
Month=Oct
```

You could provide the name of the month explicitly although it is hard to imagine when that would be necessary.

```
Month="October"
```

Note Any comment that you care to make about the publication or availability of the reference. The Note field is printed at the end of the reference, after the publication entry. Use Note to help the reader find your reference:

```
Note="Also available on microfilm, # A060494"
```

Number The number of the journal containing an article.

Organization The organization that sponsored the conference that generated the proceedings.

Pages Use the Pages field to pinpoint the location in a journal where an article appears or to pinpoint the spot in a proceedings or a book where some section appears. In cases where the Chapter field is applicable, you can use either Pages or Chapter, but not both. This field should contain the page or pages on which the article appears, not the page or pages to which you are making specific reference. Typical entries might be:

```
Pages="156-202"
Pages="301"
```

Publisher The name of the publishing house that published the reference. For obscure publishers, you should also include an Address field, giving the address of the publisher.

School The name of the School, College, University, or whatever granting the degree for **MastersThesis** and **PhDThesis**.

Series The number of a book, in a publisher's series.

Title The title of the item being referred to, whether **Article**, **Thesis**, **TechReport**. (See also BookTitle.)

Type A field in **TechReport**. Scribe assumes that the entry for a TechReport is actually titled "Technical Report ..." and so prefixes the report number (when present) with the words "Technical Report". However, the institution publishing the report might actually call it something else, like "Research Report". In that case, you would have a field in the entry like this:

```
Type="Research Report"
```

Scribe would then prefix the report number with "Research Report" instead of "Technical Report".

Volume The volume number for a journal containing an article or for a book in a multivolume set.

Year The year of publication.

12.5.2.4 Abbreviations

Every journal has its own standards for abbreviating titles, and they are all different. The standard Scribe Database defines a standard set of abbreviations of journals in the field of Computer Science, and each reference format provides the correct expansion of those abbreviations.

To use an abbreviation, just place it, not delimited or quoted, in the spot where you would normally put some long field inside delimiters. For example, you could replace this field entry:

```
Journal="Communications of the ACM"
```

with this abbreviated entry:

```
Journal=CACM
```

Every standard reference format type supplied with Scribe contains an expansion for CACM according to the rules of its journal.

You can define your own abbreviations using @String commands (See Section 4.2). Put these commands at the beginning of your bibliography database file, before the first data entry. The abbreviation in the example above would have been defined:

```
@String(CACM="Communications of the ACM")
```

Do not use string abbreviations in the Key field, though.

12.6 Some Examples

```
@UnPublished(SapsfordStudy
        ,Key="Sapsford"
        ,Year="1978"
        ,Author="Sapsford, Mark A."
        ,Title="PDP-10 Usage Study August 78 to December 78"
        ,Note="The summary results of this study are in a
                        notebook owned by Richard Swan."
        )
@TechReport(PUB
        ,Key="Tesler"
        ,Author="Tesler, Larry"
        ,Title="PUB: The Document Compiler"
        ,Institution="Stanford University Artificial
                        Intelligence Project"
        ,Year=1972
        ,Number="ON-70"
        ,Month=sep)
@InCollection(Bell77
        ,Key="Bell"
        ,Author="Bell, C. Gordon"
        ,Title="What Have we Learned from the PDP11?"
        ,Publisher="Academic Press"
        ,Year="1977"
        ,Editor="Jones, Anita K."
        ,Booktitle="Perspectives on Computer Science"
        ,Chapter="1")
@Book(Volume3, Key="Knuth", Author="Knuth, Donald E."
        ,Title="Sorting and Searching", Publisher="Addison-
                        Wesley"
        ,Year="1973", Series="The Art of Computer Programming",
        Volume="3",Address="Reading, Mass.")
@InProceedings(Nonprocedural
        ,Key="Leavenworth"
        ,Author="Leavenworth, Burt M. and Sammet, Jean E."
        ,Title="An Overview of Nonprocedural Languages"
        ,Organization="ACM-SIGPLAN"
        ,Booktitle="Proceedings of a Symposium on Very High
                        Level Languages"
        ,Year="1974"
        ,Month="April"
        ,Note="Published as Volume 9, Number 4, of @I[SIGPLAN
                        Notices]")
```

```
@Article(CommTypo
        ,Key="Gottschall"
        ,Author="Gottschall, Edward M."
        ,Title="Communications Typographics"
        ,Journal="IEEE Transactions on Professional
                        Communication"
        ,Volume="PC21",Year="1978",Number="1",Month="March"
        ,Pages="18-23")
```

Chapter Thirteen
Producing Large Documents

Big documents are much harder to produce than small ones. The amount of work needed to produce one 200-page document is much more than 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 (though only a minute or two of actual run time) just to process a 200-page document through Scribe when the system is heavily loaded.

Scribe has several mechanisms that make it easier to produce large documents. These large-document facilities include:

- An **@Include** command, that lets you compose a large document from any number of separate files, each one of which is of manageable size.
- An **@Part** command that lets you process a component file independently of the whole document, yet still have correct page numbers, section numbers, chapter numbers, and cross references.
- An **@Use** command to request that Scribe use some private or custom edition of its Database.
- An **outline** of your document, automatically generated by Scribe in a separate file, showing the sectioning structure of your document and its cross reference points, to help you manage its organization.
- A **word counter** and **vocabulary analyzer**.

13.1 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, and then resume processing of the original manuscript file. For example, suppose you have this manuscript file:

```
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.
@Include(SUBFIL.MSS)
At first the Witch was tempted to run 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.
```

This file has a request to include the file whose name is SUBFIL.MSS:

```
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.
```

The finished document file would contain the following 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 can be nested. That is, a file being processed because of an @Include command can itself contain other @Include commands naming yet more files.

The example above shows how @Include works. However, piecemeal inclusion of text is more confusing and harder to manage than a system that partitions the document in some way that reflects its logical structure. One of the best strategies is to build a small *root file* that contains any commands that must be specified at the beginning of the document before first text, such as the @Make, @Device, and @Style commands, followed by a series of @Include commands to include each of the component files. Note that the @Include command constitutes first text. Even though it looks like only a Scribe command, it causes text to be processed and so is considered to be text. If an @Make command, for example, were placed after an @Include command in your .MSS file, Scribe will produce an error stating that the @Make command is only allowed at the beginning of the manuscript file.

It is often 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. It also helps to use comments at the beginning of included files describing their contents or any other pertinent bookkeeping information that needs to be stored with them. (See the @Comment command in Section 4.7, page 42.)

Figure 13-1 shows an example root file for a user's manual of a program that generates graphs and plots. It prints the title page and then includes, in turn, each of the chapter files.

```
@Make(Manual)
@Use(Bibliography "<Plot>Plot.bib")
@String(Version="9.0 (1)")
@String(Plot="P@C[lot]")
@PageHeading(Left "@C{Plot User's Manual}",
             Right "@C{Page @Value(Page)}")
@Include(COVER.MSS)
@Include(PREFACE.MSS)
@Include(INTRO.MSS)
@Include(BEGIN.MSS)
@Include(CURABS.MSS)
@Include(AXIABS.MSS)
@Include(GRAABS.MSS)
@Include(TEXABS.MSS)
@Include(GENERAL.MSS)
@UnNumbered(References)
@Bibliography
@Include(HACKER.MSS)
```

Figure 13-1: Sample Root File

13.2 Separate Processing of Component Files

You can process component files independently as manuscripts in their own right. By judicious use of the @Part command (described in this section), you can process the components of a large document separately for draft copy purposes and yet be able to process the whole thing without having to change any of the components.

13.2.1 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 being 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 can serve both as included parts of a large document and as documents in their own right.

Scribe does not process certain commands when they occur in an included file but processes the same commands when they occur in a root file. This conditional inclusion lets you put commands into a file that are essentially ignored if it is part of a larger document but processed if it is being produced as a document in its own right.

The commands **@Device**, **@Make**, and initial **@Style** commands are not processed if they are found in an included file when the root file is being run through Scribe (although Scribe will produce warning messages about them being specified after first text), but they are processed when the part file is run through Scribe independently. Of course, the commands are processed if they are found at the beginning of the root file. Therefore, you can put the commands necessary to process a file as a stand-alone document at the beginning of the file, trusting that they will not take effect if it is included as part of a larger document.

13.2.2 Separate Processing of Component Parts: The @Part Command

For very large documents, the processing time for the full document is large enough that it is wasteful to reprocess and reprint all of the document when only a piece of it has changed. A person working on a small section of a large document needs to be able to produce a draft copy of that section without the rest of the document. When each chapter is produced by a different author, the several authors each need to be able to get proof copies of their chapters apart from the other text.

For situations like these, Scribe provides an @Part command, which makes it possible to process fragments of a document separately while maintaining their identity as part of the document. That is, the part document has page numbers, section numbers, and cross references appropriate to its position in the document.

To take advantage of this separate compilation facility, include an @Part command as the first command in the file to be processed separately. The @Part command must provide two items of information: A name for the part, and the name of the root file of the document. Its syntax is:

@Part(*part-name***, Root=***delimited-root-filename***)**

The following @Part command defines a Section entitled ''BasicConcepts'' which is a part of the document whose root file is PRIMER.MSS.

```
@Part(BasicConcepts, Root "primer.mss")
```

The name that you choose for the part is just a name and so can be any combination of letters, digits, and hyphens.

Having put an @Part command, each with a different part name but all specifying the correct root file, into the various subfiles, you must process the entire document once so that Scribe can record the part information. When Scribe finds an @Part command in an included file while processing the root 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 auxiliary (.AUX) file for the document.

Once you have an .AUX file for a document with part information stored in it, you can process any part by itself while maintaining the part's relationship to the whole document. When Scribe finds an @Part command at the beginning of the file it is processing, it assumes that the file is part of a larger document. Scribe then enters a special *sub-compilation mode.* Scribe does three special things in sub-compilation mode:

1. Finds the root file and processes the commands that are in it before the first text.

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

3. Reads in the cross reference label definitions for the entire document (not just for the part being processed) from the .AUX file.

The result of this complicated series of acts is that the part document receives the correct page and section numbers and (if it contains any cross references to labels defined in some other part), correct cross references. The Index and Table of Contents produced by processing a part reflect only the contents of that part.

At the end of processing a part document, Scribe updates and rewrites the .AUX file, correcting the stored page number information for parts that follow this one, so that page numbers for the other parts remain correct, even though the number of pages in this part might have changed. This update will not work properly unless all of the included files, even those that you do not plan to compile separately, have @Part commands at the front.

When you add a new part, you must process the whole document once in order to incorporate information for the new part correctly into the .AUX file. Otherwise, Scribe issues some warning messages and processes the part as if it started on page 1.

13.2.3 String Definitions in Multiple Part Documents

Text strings, defined using @String, are sometimes available when you are processing a part file separately. Their availability depends on where they were defined. Text strings defined in the root file, before any text lines, are available to any separate part file because during subcompilation, Scribe reads the commands at the beginning of the root file.

String definitions are not stored in the .AUX file. Therefore, strings defined in one part file are not available to another part file during separate processing. Also, because they are not in the .AUX file, strings must be defined (with the @String command) before they are used (with the @Value command). That is, forward references are impossible with strings during separate compilation. (They do work fine, however, when the root file is run through Scribe.)

13.3 Filenames and the @Use Command

The @Use command directs Scribe to use Database, bibliography, and auxiliary files other than the ones it normally would. For example, Scribe normally looks for an .AUX file with the same name as the manuscript file: if your manuscript file is "PRIMER.MSS", then it looks for "PRIMER.AUX". The following command tells Scribe to look for and use OTHER.AUX as the auxiliary file.

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

The extension must be .AUX; that is, you cannot ask Scribe to use something like OTHER.TMP as the auxiliary file.

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

```
@Use(Bibliography "<Bovik>PROSYS.BIB")
```

In this case, you must supply a file extension but any extension is fine; it does not require file type .BIB.

Normally when Scribe is searching for a Database file, it first looks in its own directory, and if it fails to find what it is looking for, it then searches your directory. You can use the @Use command to direct Scribe to search some other directory besides your own when it is unable to find the file in its directory. The following command at the beginning of a root file directs Scribe to look in some location other than your directory for *all* the Database files it needs but cannot find in its own directory.

```
@Use(Database "<XSCRIBE>")
@Use(Database "[100,113]")
```

The precise nature of the location string depends on the details of your computer system. *Do not* specify a filename in the @Use(Database) command.

13.4 Managing Cross References and Document Organization

Scribe creates an outline file (with file extension .OTL) containing information about section numbers, titles, and cross reference labels. The format of the information in the .OTL file has been designed to help you keep track of labels, tags, section names, and so on as the document grows. The .OTL file has two parts, a sequential part and an alphabetical part.

The first part of the .OTL file contains all the titles from the Table of Contents, all the names of @Label and @Tag parameters, the document page where they occur, and their position in the manuscript source file. The following sample lines from the .OTL file for an early draft of this chapter show the various fields and their format.

```
Thirteen Producing Large Documents                    137 AIDS.UM1, 00200/1
                            BigDocuments              137 AIDS.UM1, 00300/1
  13.1 Breaking a Manuscript into Several Smaller Files 137 AIDS.UM1, 05400/1
                            IncludeCommand            137 AIDS.UM1, 05500/1
                            RootFigure 13-1           138 AIDS.UM1, 15400/1
  13.2 Separate Processing of Component Files         139 AIDS.UM1, 15600/1
                            SepCom                    139 AIDS.UM1, 15700/1
    13.2.1 Component Files as Separate Documents      139 AIDS.UM1, 16500/1
    13.2.2 Separate Processing of Component Parts: The 140 AIDS.UM1, 20100/1
    13.2.3 String Definitions in Multiple Part Document 141 AIDS.UM1, 29000/1
                            AuxNoString               141 AIDS.UM1, 29100/1
  13.3 Filenames and the @Use Command                 141 AIDS.UM1, 31300/1
                            UseCommand                141 AIDS.UM1, 31400/1
  13.4 Managing Cross References and Document Organizat 142 AIDS.UM1, 39200/1
                            OTLfile                   142 AIDS.UM1, 39300/1
  13.5 Word Counts and Vocabulary Construction        143 AIDS.UM1, 48900/1
```

The outline is in the same sequence as your document. Each time Scribe sees a sectioning command (such as @Section or @Chapter) or a cross reference marking command (@Label or @Tag), it puts an appropriate line into the outline file. Every line has the manuscript file name and position and the document page number (137 through 143 in this example); the entries for @Tag definitions (RootFigure and VocabularyFigure in this example) also show the number of the thing that the @Tag marks (14-1 and 14-2).

The second part of the .OTL file contains an alphabetized listing of the codenames for @Tag and @Label. In each line, it records the codename, the document page on which the @Label or @Tag command occurred (available from @PageRef), the numeric value associated with the codename (available from @Ref), and the location of the command in the manuscript file.

For example,

Alphabetic Listing of Cross-Reference Tags and Labels

Tag or Label Name	Page	Label Value	Source file Location
AUXNOSTRING	141	13.2.3	AIDS.UM1, 29100/1
BIGDOCUMENTS	137	13	AIDS.UM1, 00300/1
INCLUDECOMMAND	137	13.1	AIDS.UM1, 05500/1
OTLFILE	142	13.4	AIDS.UM1, 39300/1
ROOTFIGURE	139	13-1	AIDS.UM1, 15400/1
SEPCOM	139	13.2	AIDS.UM1, 15700/1
USECOMMAND	141	13.3	AIDS.UM1, 31400/1

If there are codewords that were used but never defined, Scribe produces a table of these labels showing the location in the manuscript file of the first @Ref command using the codeword. This table is placed both in the outline file and the error log file. A sample undefined-label table looks like this example:

Undefined Label	First reference
CHARMAP	(ENVIRS.UM1, 02900/19)
EXAMPLES	(ENVIRS.UM1, 01100/7)
GREEKMAP	(ENVIRS.UM1, 02600/19)
MAKEFORM	(SERMON.UM1, 04600/1)
PRIVATEFONTS	(FORMAT.UM1, 04000/7)
STYLECHAPTER	(ENVIRS.UM1, 02200/5)
STYLECOMMAND	(TITLES.UM1, 09700/3)
TOPMARGIN	(ATOD.UM3, 02300/1)

13.5 Word Counts and Vocabulary Construction

Sometimes you want to know how many words are in a document. Some journals have a minimum or maximum; other times you're just curious. One of the few things that computers can do flawlessly is count, so Scribe has in it a word-counting mechanism to tell you how many words are in your document.

Unfortunately, it's not always easy to establish just what is a word and what isn't a word. For the purposes of this word count, a word is the sequence of characters between two consecutive word breaks; read Section 10.1 on page 93 if you don't understand what a word break is.

To get Scribe to count words, just supply the W or WordCount command-line option when you run the program. Section 2.4, beginning on page 8, talks about options and how to use them.

When Scribe finishes processing the manuscript file, it prints a line showing the word count totals on your terminal and in the error log file.

```
**Word Count: 1200 (1041 in text, 159 in display).
```

For the purposes of this tally, "display", means any unfilled environment, for example, Example, Display, Center, or MajorHeading; "text" is everything else. Text in page headings and page footings is not counted at all.

Scribe can also build you a list of the words that you use in a document, that is, the document's vocabulary. If you include the V or Vocabulary option, it builds this vocabulary list in a file. The name of the file is the same as that of the manuscript file; its file extension is .LEX.

The vocabulary list contains, in alphabetical order, all of the words that you use in your document and the number of times that word is used. (If the word is used only once, no number is shown.) For the purposes of this vocabulary list, a word is any sequence of letters, numbers, hyphens, and apostrophes.

Below is part of a sample vocabulary list; it's taken from this manual.

```
ABBREVIATIONS          11
ABILITY                 3
ABLE                   11
ABOUT                  71
ABOVE                  51
ABOVE-MENTIONED
ABSENCE
ABSOLUTE               10
ABSOLUTELY
ACADEMIC                4
ACCEPT                  2
ACCEPTABLE              4
ACCEPTANCE              2
ACCEPTED
ACCEPTS                 2
ACCESS                  4
```

The number in the right-hand column is the number of times that the word appears. If a word appears only once, then no number is printed. The vocabulary analyzer uses a lot of memory; if you are making a very big document (500 pages or more) with a big index (500 entries or more), you might find that your computer doesn't have enough memory to process the whole file.

Chapter Fourteen
Messages From Scribe

Scribe produces several kinds of information while it is processing a manuscript file. This chapter describes what Scribe is telling you and what, if anything, you have to do about it.

Three classes of messages appear on your terminal while Scribe is working:

- Phase of processing. You can tell by the information on the terminal whether Scribe is reading its Database or is actually working on your file.
- Warnings and error messages. Scribe informs you of all problems it finds while processing the file.
- Final summary. Scribe finishes up with a summary of any problems and a list of the files it created during the run.

The phase and summary messages are primarily for your information, so you can be sure that Scribe is actually doing what you intended. The warnings and error messages identify problems with the run that are caused either by something in your manuscript or by something in the Database. A discussion of these messages appears in Section 14.2.

14.1 Informational Messages

Let's examine again what happens when Scribe processes a manuscript file. We'll use the following short but reasonably complicated manuscript file so that you can see the different kinds of messages that appear.

```
@Part(MemoDescript, Root "local.mss")
@Chapter[Corporate Memo Document Type]
@Label[MemoType]
A corporate memo requires four environments in
the following order.
@Display<
@@To[...]
@@From[...]
@@Subject[...]
@@Begin[Body]
...
```

```
@@End[Body]
>
A manuscript file for a simple memo looks like this example:
@Begin(Verbatim)
@Include(locex3.fig)
@End(Verbatim)
```

This short file is named LOCMEM.MSS. We can see from the @Part command that it is part of a larger document whose root is named LOCAL.MSS. Also, we can see from the @Include command that it includes another file named LOCEX3.FIG. Let's examine the information messages from Scribe as it works on the part file LOCMEM.MSS.

```
@scribe
Scribe 4(1400)  Copyright (C) 1981, 1984  UNILOGIC, Ltd.
*locmem
[Processing LOCMEM.MSS.3 (Part MEMODESCRIPT of root LOCAL.MSS)
    [Device "LPT"]
    [Document type "manual"]
    [Subfile LOCAL.AUX.4]
 7 (2.3) 8 9
    [Subfile LOCEX3.FIG.3]
 10]
(Index has 8 entries)
[#INDEX]

[CONTENTS 11]
 i.

Error found while finishing up after the end of the manuscript:
Cross references to 2 labels could be wrong.
Run the file through Scribe again if you need to be sure they
        are right.

**LOCMEM.LPT for device LPT has 6 pages.
**LOCAL.AUX written.
**LOCMEM.OTL written.
**LOCMEM.ERR lists 1 error.

@
```

When it begins, Scribe reports the name of the file it is going to process. In this case, the file is a portion of a larger document, so Scribe also reports both the part name and the name of the root file. In the next two lines, Scribe reports both the device type it is going to create the document for (LPT in this case) and the document type it is going to use (Manual in this case). You can verify that Scribe is going to do what you intended by looking at these lines. Next, Scribe locates and reads the .AUX file for your whole document, if one is present, and reports it in the line [Subfile LOCAL.AUX.4]. At this point in processing, Scribe has completed its setup phase and is ready to start working on your manuscript. (The setup phase takes a few seconds of computer time on a typical document type. On small, slow, heavily-loaded systems, it can take several minutes to complete.)

As Scribe processes the file, it prints page numbers and section numbers on your terminal. Section numbers are in parentheses. In the example above, page numbers 7, 8, and 9 are announced, and Section 2.3 is announced.

When Scribe processes the included file, LOCEX3.FIG, it again reports the name of the subfile it is using. After the last page in the text, it reports writing an Index (with 8 entries in this case) and a Table of Contents page.

At this point, Scribe starts finishing up. First, a summary of errors found at the end of the manuscript and then a list of the names of the files that this run is created. In this case, the main document is in LOCMEM.LPT because we used device type LPT.

Scribe writes an error log file only when some error occurred during the run. So, you can be sure when it does not report writing an .ERR file that no errors occurred.

You can keep the error messages off your terminal by using Q or Quiet as a command-line option. Those options stop error messages from printing while Scribe is processing. While it can be useful to keep messages from appearing during processing, it can also cause problems. Do not disregard Scribe's error messages unless you know what you are doing and have a good reason for it. In particular, don't report a problem to your ***DBA*** until you have eliminated all possible errors listed in the .ERR file.

14.2 Warnings and Errors

Scribe produces three classes of messages while it is working.

- Warnings. These are messages of an informative nature that describe circumstances that Scribe suspects are incorrect but that don't necessarily mean that there is anything wrong. You might want to fix what is causing the message or you might not.
- Errors. Problems exist in the manuscript that you probably want to fix some day, but Scribe can process your file fairly well in the meantime. Errors do not prevent Scribe from producing a document file, but its appearance might be less than perfect.
- Serious Errors. Serious problems exist in the manuscript file or the Database. These problems are often fatal, forcing Scribe to give up without producing a document file. Fatal errors are normally the result of operating system errors, missing Database entries, or physical limitations imposed on Scribe by the computer, such as lack of sufficient memory.

A list of Scribe's message texts follows. Each message is classified according to its severity and has a short explanation of its meaning. Some of the phrases in the messages are in italics inside angle bracket delimiters, for example, *<filename>* or *<distance>*. These indicate spots where Scribe fills in the message with whatever phrase applies to your particular problem. For example, it would replace *<filename>* with whatever file it was processing when it found the problem.

The messages in the table are in alphabetical order according to the non-italic portions of their text.

14.3 Error Message Texts and Explanations

A line together with its footnotes is bigger than the page!

(Error) A page can hold a certain number of text lines. Scribe always puts the text for a footnote on the same page as the line referencing that footnote. If a line has too many footnotes, or if one footnote is too big, then the line together with its footnotes cannot fit on one page. Scribe puts as much of the footnote text as it can on the page with the footnoted line. If you don't think that your footnote text is too big and are puzzled by this message, check for a missing delimiter at the end of an @Foot command.

A word is wider than the output line. It will extend beyond the right margin.

(Error) Some environments, such as Enumerate, Quotation, and Display, narrow the line width by widening the margins when they are entered. If these environments are nested too deeply, the resulting output line is sometimes so narrow that long words are wider than the entire line. Sometimes, if you accidentally omit an @End(Enumerate) or @End(Quotation) or the like, you end up nesting environments without realizing it.

An integer was required, but you used "<*non-integer number*>". Truncated to <*integer*>.

(Warning) Some environment and @Style parameters insist on an integer value, but you have provided something that is not an integer. If this entry is not just a typo, then you probably don't understand the meaning of the parameter and should refresh your memory from the appropriate table in this manual.

Bad keyword "<*name*>"; must be <*list of legal parameters in this context*>.

(Error) Many @Style and environment parameters take on values from a set of names. For example, the WidowAction @Style parameter must be given a value that is one of {Ignore, Warn, Force, ForceWarn}. If Scribe sees something like the following, then it reminds you, with this error message, of the list of possible parameters.

```
@Style(WidowAction 3)
```

The list of possibilities is printed only if there are six or fewer of them.

Cross references to <*integer*> labels could be wrong. Run the file through Scribe again if you need to be sure they are right.

(Warning) This message means that some forward references might be wrong. Read the section about forward references, Section 7.3, on page 69, to find out what this term means. If you run the manuscript file through Scribe a second time, the problem will be cured, but if this copy is not your final draft of the document, there's no point in doing that.

Current font (*font name*) does not contain a "<*something*>" character.

(Error) Not all fonts available on all printing devices contain all 127 ASCII characters. Your text contains a request for a character that is not present in the font that you are using. Contact your ***DBA*** if you can not fix the problem yourself.

Database entry "<*name*>" missing for device <*device name*>.

(Serious) Scribe cannot continue without the required Database entry. This error often indicates some problem with the Scribe Database at your site and is very rarely caused by anything you are doing wrong.

Default database entry missing for <*entry name*> on <*device name*>. Can't continue.
(Serious) The ***DBA*** at each Scribe site decides what the defaults are for various formats when you have not explicitly requested one. Scribe is looking for a Database entry for something that is supposed to be the default, but cannot find that entry in its Database. Contact your ***DBA***.

Definition file for @Make(*document type*): no FaceCode clause in level-1 @Begin.
(Serious) This problem is an error in the document type definition in the Database, not an error in your manuscript file. What it most likely means is that the document type that you are using has not been defined correctly by the person who made it, although it is possible for this error to mean that something awful has happened to one of the ordinary document types. Contact your ***DBA***.

Definition file for @Make(*document type*): no Font clause in level-1 @Begin.
(Serious) See above. A slight variation on the previous message. Not your fault.

Definition file for @Make(<*document type*>): no text allowed before @Begin.
(Serious) See above. Not your fault.

Device <*device name*> not known.
(Serious) Scribe will not continue. This error message means that Scribe was unable to find in its Database a definition file for the device that you have asked for with the @Device command.

Error in definition or use of <*environment name*>: the BREAK attribute must be used with GROUP.
(Serious) The named environment has been defined incorrectly. Scribe has detected this error at a time that makes it impossible to recover gracefully.

Font <*Font name*> does not include <*FaceCode name*>. Using R (if possible) instead.
(Error) You have begun an environment that is requesting the named FaceCode in the named Font. That Font as defined in the Database does not include that FaceCode. Scribe proceeds as if the environment had asked for FaceCode R, which is the Regular or Roman FaceCode. This omission might be an error in the Font definition, or it might be an error in the environment definition, but it is probably not your fault, unless you have been using your own FontFamily files.

Font database does not contain <*device-specific font name*>
(Error) The Scribe font that you have selected with a @Style(FontFamily) command is referencing a device font that does not exist. This problem might be an error in the Scribe Database, or it might be an error in the font data provided by the printer manufacturer. Contact your ***DBA***.

Format error in AUX file : <*problem encountered*>.
(Error) Scribe has found something in the .AUX file that does not look right, indicating that you might have changed the file. The entire line on which the error was found is ignored.

Format error in file <*Database filename*>; line ignored.
(Error) The named Database file does not contain the required @Marker commands. This is an error in the Database, of course. Contact your ***DBA***.

"<*name*>" is not a recognized code. Something like "INCHES" or "TRUE" needed here.
(Error) You have used an environment or @Style parameter with an incor-

rect value. The list of possibilities for correct arguments is too long to list for you. Consult the manual.

"*<undefined name or parameter>*" **is used where** *<particular kind of parameter>* **is needed.**
(Error) Some environment attributes, such as Use, Counter, and Within, must refer to an existing defined name. This message is printed when you have not provided a valid predefined name as a value to that parameter or when you have provided a name of the wrong type.

"*defined name*", **a** *<wrong type>*, **is used where** *<right type>* **is needed.**
(Error) You have provided a name of the incorrect type as a parameter argument. Types include Counters, Environments, Commands, and Integers.

<name> **is a** *<type>*, **but the** *<environment parameter>* **keyword needs a** *<type>*.
(Error) Same sort of error as above.

@Begin(*name*) **is meaningless:** *name* **is not a defined environment type.**
(Error) Scribe pretends that you said @Begin(Verbatim) instead.

<unknown name> **is not a recognized command keyword.**
(Error) This message is printed by the parameter evaluator in Scribe when it is so deeply entangled in processing that it has lost track of what command you used. The parameter is wrong nevertheless. Check spelling.

<unknown name> **is undefined; the** *<environment parameter>* **keyword must refer to** *<operand type>*.
(Error) Some environment parameters, like Use, Counter, and Within, need an operand of some particular type. This operand is undefined, which means it has no particular type and is certainly wrong.

<integer> **labels were referenced but never defined. See error log file.**
(Error) You have made cross references with @Ref to labels that you were supposed to define with @Label or @Tag, but no definitions were found. The error log gives you a list of the label names and where you referenced them; go check their spelling. You can look in the .OTL file for a list of all of the @Label and @Tag codewords that *were* defined.

Line too wide; lost "*<some text>***".**
(Warning) Unfilled environments normally chop the lines to fit within the margins. This line was too long and had to be chopped. Perhaps you should use an environment like Verse that wraps long lines or perhaps the margins are set wrong.

@*<name>* **must be followed by an opening delimiter; you have put "***<some character>***" instead.**
(Error) This message means exactly what it says. You have typed something like @Begin-Itemize or @End)Enumerate, failing to use appropriate delimiters.

No *<device name>* **font named** *<font name>*; **the default will be used.**
(Warning) The font that you have asked for with @Style(FontFamily), or perhaps the font that the document type designer has asked for with a @Style(FontFamily) inside the document type definition, is not available. Scribe uses the default font instead of the one requested.

No *<device name>* *<Database item type>* **named** *<Database entry name>* **was found in the data base.**
(Serious) Pieces of the Database are missing; contact your ***DBA***.

No definition found in the database for document type <*name*> on device <*name*>.

(Serious) The document type definition that you have requested with the @Make command cannot be found in the Database for the device you specified with the @Device command. Scribe cannot continue and must stop without creating a document file. Verify that you have requested a valid device in the @Device command and a valid document type in the @Make command. If you have, then contact your ***DBA***.

Only <*integer*> fields are needed in the <*command name*> command (at <*where in file*>). The rest are being ignored.

(Warning) You have provided more fields to a command than the command was expecting. If this error is not just a typo, then you probably don't understand the syntax of the command. If this message is printed about a command in the .AUX file, the real error is probably an invalid character in a @Label or @Tag name from an earlier run, which for some reason was not detected at the time.

Only bibliography commands are permitted in a BIB file. <*name*> is not a bibliography command.

(Warning) The offending command is actually executed but peculiar things might happen, since the .BIB file is sorted before it is processed and the command may not be read at the time you expect to see it.

Opening delimiter <*some left delimiter*> not matched by a closing <*right delimiter*>.

(Error) Scribe never found the closing delimiter. Maybe you forgot it. If it's really there, maybe it's so far forward in the file that Scribe gave up looking for it. You can set the StringMax @Style parameter (Appendix E.7, page 217) to a larger number so that Scribe looks farther ahead without giving up.

Output device <*device name*> cannot achieve <*special effect name*>: ignored.

(Error) This problem is usually an error in the device definition in the Database, but it is not detected until the first attempt to use the incorrectly-defined environment. The output device does not have the ability to perform some function like underlining or boldfacing, yet the definition of this environment is asking for that effect. The request is ignored.

String begun at <*file location*> was never terminated with "@End(Comment)".

(Serious) You have begun a long-form comment but never ended it. The entire remainder of your file, beginning at the mentioned file location, has been ignored.

Tab sequence (@\) used, but cursor has already passed the last tab stop.

(Error) The tab request is ignored. You might have set too few tab stops, or your text might have carried you past one of them already.

The "<*some character*>" character that is supposed to be the end of the "*name*" command (at <*file location*>) is missing.

(Error) Scribe was unable to find the end of a footnote, chapter heading, or other command encompassing text. If the delimiter is really there, then try increasing the Stringmax @Style value (see page 217).

The <*parameter name*> parameter to the <*command name*> command needs a delimited string.

(Error) Commands such as @Pageheading, in which you provide a series of text fields, insist that the values of those fields be delimited. Thus, you must say

```
@Pageheading(Left="Book")
```

rather than

```
@Pageheading(Left=Book)
```

The <*environment type*> begun at <*file location*> is too big vertically to fit on one page.

(Error) Figures, tables, equations, and that sort of thing must each fit on one page. This one doesn't.

The <*command name*> command is allowed only in <*restricted location*>.

(Error) The command is restricted to Database files, as indicated, but you have used it somewhere else.

The @Caption command is meaningful only inside a figure or table. It is being ignored.

(Error) You have tried to give a caption to something that isn't a table or a figure or other Database environment for which captions are valid. Check your delimiters to make sure you haven't accidentally put it in the wrong place.

The <*command name*> command needs a <*parameter name*> keyword, which you have not provided.

(Error) Some commands require that you provide specific parameter values.

The <*parameter name*> keyword is not meaningful in the <*command name*> command.

(Error) You have used a counter attribute in an environment or an environment attribute in a @Style command, or something like that.

The <*@Style parameter name*> style keyword must be specified at the beginning of the document, before the first text.

(Error) Some Style parameters are restricted to appear only at the beginning of the document. This one, for example, is so restricted. If you think that it is at the beginning of the document and Scribe thinks it isn't, carefully check for extra closing delimiters and other spurious text. The "beginning" of a document is defined to be that part of it up to but not including the first text or the first command that, if processed, might cause text to be generated.

The @<*environment name*> environment begun at <*file location*> was never closed.

(Error) You forgot the closing delimiter of a short-form environment. The file location listed is the location of the beginning of that environment; it's up to you to decide where the end should have been.

The @/ command is not meaningful unless the return marker is defined; it is being ignored.

(Warning) The @/ command means "move the cursor to the return marker." The return marker is not set. You probably meant to use the @\ command, which means "tab to the next tab stop."

The @Begin(*environment name*) at <*file location*> was never closed.

(Error) You forgot the @End at the end of a long-form environment entry. The file location given is the location of the @Begin.

The AUX file <*filename*> is missing. Page numbering will start at 1.

(Warning) You are compiling a subpart file, but there is no .AUX file. The processing continues, but the page, chapter, and counter numbers in the subpart all begin at 1.

The AUX file for this document shows no record of a part named <*name*>. Page numbering will start at 1.

(Warning) You are compiling a subpart file, but the .AUX file shows no record of it ever having been part of this document. Processing does continue with all the page, chapter, and other counter numbers in the subpart beginning at 1.

The bibliography tag "*codeword*" has already been defined (at *<file location>*). This one will not be included in your complete bibliography.

(Warning) You have asked for the entire bibliography file to be included by specifying @Style(BibSelect Complete). Scribe has found a bibliography entry whose codeword duplicates a codeword found earlier in this or a previous .BIB file. This later entry will not be included in the bibliography listing in your document.

The character "*<some character>*" is invalid as used in the template "*<template>*".

(Error) Numbering templates are used in counters, like Chapter and Section, and numbered environments, like Enumerate and Theorem. Errors in templates are detected when they are first used, not when they are defined. The current command references a template with an invalid code. Figure 15-2 on page 176 lists the valid template codes.

The characters "*<nonsense characters>*" are invalid here and are being ignored. Probable cause: missing close-delimiter earlier in the file.

(Error) In Scribe commands that take parameters, the parameters or parameter/value pairs are separated by commas. This message is printed when Scribe finds text after a parameter or parameter/value pair but before a comma or closing delimiter. The most common cause of this error condition is failing to put a closing delimiter on a parameter's value earlier in the command, which disrupts the delimiter balance for the rest of the command.

The command *<command name>* is allowed only at the beginning of a manuscript.

(Error) Some commands are restricted to appear only at the beginning of the document. If you think the command in question is at the beginning of the document and Scribe thinks it isn't, carefully check for extra closing delimiters and other spurious text. The "beginning" of a document is defined to be that part of it up to but not including the first text or the first command that, if processed, might cause text to be generated.

The current font (*font name*) does not include special character number *<integer>*.

(Error) This message indicates an error in a Database file; the precise nature of the error depends on the printing device. Contact your ***DBA***.

The delimited string beginning at *<file location>* was never terminated by a closing *<some character>*.

(Error) When a parameter gets a delimited string for an argument, as in some command like `@Style(Font "Helvetica 12")`, Scribe gets unhappy when the closing delimiter is missing. If the delimiter is not really missing, then you need to increase the value of the StringMax @Style parameter.

The label *<codeword>* has already been defined (at *<file location>*). This redefinition is being ignored.

(Error) @Label and @Tag codewords must be unique. You have used one twice.

The last *<integer>* tab stops set by this command were past the right margin.

(Warning) You can't set tabs past the current right margin.

The left and right margins overlap; this leaves no room for any text.

(Error) The margins that you have specified are so big that the sum of the left margin and the right margin is greater than the paper width. Remember that the right margin value is measured from the right edge of the paper, and the left margin value is measured from the left edge of the paper.

The manuscript file contains no text.

(Warning) If you want output, you've got to provide input. Your manuscript file is either empty or else consists of nothing but setup commands.

The name <*name*> is used in the @Value command, but it has not been defined with @String.

(Error) Appendix E.6 on page 216 contains a list of predefined strings. Any name not in that list must be defined with @String before it can be used in an @Value command.

The keyword <*@Make parameter name*> is not part of document type <*document type name*>. It is being ignored.

(Warning) @Make commands can take parameters to modify the document type, but the parameter that you have provided in this @Make command is not a correct parameter for this document type. Contact your ***DBA***.

The sequence "@<*some character*>" is not meaningful; treated as text.

(Warning) You have put an "@" character in your manuscript file and followed it by some punctuation symbol that Scribe does not recognize. If you really meant to have an @-sign appear in the document, use two in a row.

The symbol <*name*>, <*type name*>, cannot be redefined as <*different type name*> at block level 0. This statement must go at block level 1.

(Error) This situation is an error in the document type definition file from Scribe's Database. It means that lines in the Database file are in the wrong order. Contact your ***DBA***.

This @End does not have a matching @Begin.

(Error) Somehow you have managed to get more @End commands than @Begin commands or have them improperly nested.

This @Part command is effectively at the beginning of the manuscript and is therefore being ignored.

(Warning) You don't need a @Part command at the beginning of the very first subpart file if you have no text at all in the root file.

This @Part command is missing a ROOT parameter. Can't subcompile without it, so subcompilation mode is being turned off.

(Error) If you want to use subcompilation mode, you have to provide the name of the root file in the @Part command.

This copy of Scribe has been configured without a driver for device <*device name*>.

(Serious) The copy of Scribe that you are running does not contain the necessary program code to process text for the device that you have selected. Contact your ***DBA*** to find out why.

Too many arguments to <*command name*>; ignoring all past "*parameter*"

(Warning) You have provided more parameters to this command than it knows what to do with.

Unable to access file <*filename*> because of operating-system problem: <*octal code returned from Operating System*>.

(Serious) The operating system has refused Scribe access to the named file, and it returned the indicated numeric status code as the reason why. You have to check with a systems programmer to find out what this code means. Tell her it is an error code from OPEN, ENTER, or GTJFN.

Unmatched names: @Begin(<*environment name*>) [at <*file location*>] closed with @End(*environment name*).

(Warning) The names in @Begin and @End commands have to match; you can't say @Begin(Display) and @End(Center), for example. This might mean that you got confused and have some @End commands in the wrong order.

We can't let you EQUATE the name <*name*>, because it's already defined as <*some type*>.

(Error) @Equate can only be used to define new names, not change the meaning of old ones.

Widow line.

(Warning) You have specified either @Style(WidowAction Warn) or @Style(WidowAction ForceWarn) requesting that you be warned about widow lines. This message is your warning; it printed for each widow line.

You are trying to use @F<*some character*>, but you haven't defined special font <*some character*>.

(Error) Before you can use @F1 or @F2 or other special-font environments, you must define them with the @SpecialFont command.

You can't define a string named <*name*> because it's already defined as <*type*>.

(Error) Names that you define with @String cannot duplicate other names that you define with other commands, like @Textform or @Define. This name does duplicate one.

You can't EQUATE something to the undefined name <*name*>.

(Error) You have attempted to equate a name to something that is not currently defined. Check your spelling and make sure that the @Equate command comes after the definition of the thing being equated to.

You can't specify all three of LineWidth, LeftMargin, and RightMargin.

(Warning) No more than two of the @Style parameters LineWidth, LeftMargin, and RightMargin can be specified in a given document. Sometimes, the document type designer has specified one or more of them in the document type definition file, in which case your selection has clashed with hers.

You have already defined part <*part name*> (At <*file location*>). The new definition is being ignored.

(Warning) The names in @Part commands must be unique; you have used one of them twice. Scribe keeps the first one it sees and ignores all others with the same name.

You have nested subscripts and superscripts deeper than the limit of <*integer*>.

(Warning) Superscript and subscript commands cannot be nested deeper than the named limit. You didn't really mean `@+[@+[@+[@+[@+[@+[...]]]]]]` anyhow, did you?

You may not start a new paragraph while still inside a subscript or superscript.

(Error) If Scribe encounters a paragraph break while in a subscript or superscript, it closes the scripting. If you really mean to start a new paragraph inside the script, close the script and reopen it in the new paragraph.

Chapter Fifteen
Changing Things

We hope you didn't turn here first; you really shouldn't be trying to change the basic system until you have used it enough to understand it fairly well. Even then, you might reconsider.

We believe that one of the main reasons Scribe is easy to use is that it predefines almost everything you need. We believe strongly in the advantages of using standard, unembellished document types. Perhaps this chapter is a good place to repeat explicitly the major advantages.

1. You don't waste a lot of time thinking about the typographic aspects of what you are doing. During most of the writing process, you should be concentrating on content rather than form.

2. When you use a standard document type, your manuscript files are completely portable. You can move them to any site running Scribe or include parts of them in any other document, and they always work.

3. When learning a new system, people often find its behavior unexpected or surprising. This reaction is natural; it's all part of building a mental working model of the new system. Unfortunately, many experienced programmers react to their initial feeling by trying to change the system rather than by trying to change their mental model. We don't want you to change the system. Scribe's model of the document preparation process is genuinely different from that used by many previous systems, and we believe it is worth learning.

We do agree, however, that for special applications it is sometimes necessary to modify some aspects of a document. Most simple changes that apply to the whole document are handled with @Style commands; changes for individual environments are handled differently. This chapter outlines the basic procedures for making changes to the appearance of an environment.

There is nothing in this chapter that you need to know in order to be a marvelously competent Scribe user. If you read it, you may find yourself in the twilight zone between being an experienced user and having the full power of a ***DBA***. You may find that you can't live without reading the *Scribe Database Administrator's Guide*. Be forewarned: It's complicated and even longer than this manual.

15.1 Defining Command-Name Synonyms: The @Equate Command

This section actually describes changes to the appearance of your manuscript file rather than to the appearance of the document. It belongs here, however, because it is a change that can make your subfiles nonportable.

Scribe environment and command names are full words in every case except the FaceCode environments. To use any of these names, you must type the full name, properly spelled. Scribe ignores capitalization, although we show most examples in mixed case because we find them easier to read. Some people find the extra typing annoying. Remember, though, that the manuscript is keyed only once but is read many times. Abbreviations in the .MSS file will give you fits next year when you try to revise the Annual Report.

Scribe does permit you to define synonyms for command or environment names. Use the @Equate command to define a synonym for any existing name. For example,

```
@Equate(List=Enumerate)
```

As a result of that command, you could now use either @Begin(List) or @Begin(Enumerate) to produce a numbered list. If you are British, you may feel more comfortable with the spelling of the Center environment created with this command:

```
@Equate(Centre=Center)
```

You cannot use @Equate to change the meaning of a command or environment that already exists. That is, you cannot change the definition of one name by equating it to another.

15.2 Environments

Modifications are straightforward in Scribe once you understand the nature of what you are trying to change. The characteristics of the various formatting environments are declared in Scribe's Database files. Each environment definition consists of a set of attribute-value pairs. The attributes specify aspects of the environment, like spacing, left margin, type face, or filling. The values taken on by the attributes specify what kind of formatting Scribe does.

When you use an environment, say Itemize, Scribe looks up what it found in the Database for Itemize and uses those attributes and values in processing your text. Thus, to change anything about Itemize, you would have to figure out which attribute controls the aspect you want to change and what value for that attribute would produce the effect you need.

Let's examine the definition for Itemize as it comes from the factory. You could go read the Database file to see it, but we'll print it here to save you the trouble.

```
@Define(Itemize, Break, Continue, Fill, LeftMargin +5,
        Indent -5, RightMargin 5, Numbered <- @,* >,
        NumberLocation LFR, BlankLines Break,
        Spacing 1, Above 1, Below 1, Spread 1)
```

Break, *LeftMargin*, and *Blanklines* are all *attributes*. Most attributes have *values*: The Blanklines attribute has the value "Break". Some of the attributes appear not to have values. Scribe assumes a *default* value when you don't specify one. For example, the default value when you use the Fill attribute is On. On is a *Boolean* value, which is a value from the set {On, Off, Yes, No, True, False}.

For many of these attributes, you can probably guess what aspect of the Itemize environment it controls. For example, LeftMargin, Indent, and Spacing all mean exactly the same things as they do in the context of the @Style command. (See page 39.) Others have remarkably obscure meanings, but this chapter does not attempt to explain them.

Whenever you have trouble understanding something, go read the Database. All of Scribe's Database files, except for some font information provided directly by printer manufacturers, are ordinary text files. Seeing how the various attributes behave in familiar environments is the best way to increase your understanding of their meanings. Ask the ***DBA*** at your site where to find the Database files. Look at both the .DEV file for the device you are using and the .MAK file for the document type you are using.

15.3 Modifying Environments

You can modify an environment either *locally* or *globally*. A local modification applies only to one particular instance of an environment — the one you are changing. A global modification applies to all of the instances of that environment in your document.

15.3.1 Local Changes

A local modification requires that you add an attribute-value pair for the change to the long form of the environment.

@Begin (*environment-name*, *<list-of-changes>*)

Suppose that for just one instance of Itemize, we needed the items to appear without any blank lines between them. We know that Spread is the attribute controlling the extra space between items. Therefore, we change the value for the Spread attribute to 0:

Manuscript Form:

```
@Begin(Itemize,Spread 0)
Tom

Dick

Harry
@End(Itemize)
```

The result in the document looks like this list:

Document Result:

- Tom
- Dick
- Harry

The change to the value for Spread would last only until the end of this particular Itemize environment. It would apply to any other environment nested in this one that *inherits* (rather than redefines) its Spread value. It would not apply to any other environment following this one.

You don't have to worry about "turning off" any of the changes you make. They are automatically turned off during the @End command. As you can see in the example, the @End command for the environment has its normal form, without any additional attributes.

If you find yourself using this form of environment modification more than once or twice in a document, you should consider defining a new environment or making a global change.

15.3.2 Global Changes: The @Modify Command

A global modification requires that you specify the attribute-value changes using a command called @Modify. This command needs the name of the environment being changed and the list of attribute-value changes to make:

@Modify (*Name,<list-of-changes>*)

Suppose you are doing a reference chart for something that uses a lot of short commands. It is clearly a place for using Description, but you find that Description leaves too much space (horizontally) between the header word and the descriptive text. Can we change it? Let's look at the Database definition for Description:

```
@Define(Description, Break, Continue, Above 1, Below 1,
        Fill, LeftMargin +16,Indent -16, Spacing 1)
```

Description works by moving the left margin in and then doing a hanging indent (negative indent) to move the header words out into the margin. Therefore, we can change Description to suit our needs by changing the values for LeftMargin and Indent:

```
@Modify(Description,Leftmargin 0.5in,Indent -0.5in)
```

@Modify commands may appear at any point in the .MSS file, although we suggest that you include them at the beginning of the document along with the other commands that define the document's overall characteristics. The modified form of the environment, Description in our example, takes effect as soon as the @Modify command is processed. By always including @Modify commands at the beginning of the manuscript file, you are sure of what form of the environment you will get. If you put @Modify commands halfway through a document, you may edit an early portion of the document to use one of the environments mentioned in an @Modify command and be disappointed by getting the standard form of the environment instead of the modified form.

@Modify commands can be included in subfiles specified with the @Include command (see Chapter 13), but when the entire document is run through Scribe (or in other words, the root file is run through Scribe), the modification is only in effect from that point in the document on. When a subfile is compiled separately, the modification is only available if the @Modify command is actually in that file. For example, if you had a root file with four @Include commands in it and the third included file contained an **@Modify(Enumerate, Spacing 3 Lines)** command, the fourth chapter would produce the standard form of Enumerate when it is run through Scribe itself. Only information stored in the Auxiliary file is available to subfiles during separate compilation, and modifications are *not* listed in the .AUX file.

The essence of using @Modify effectively is to recognize which of the standard environments is most similar to the effect you are trying to achieve. Think carefully about the charac-

teristics of what you are trying to produce. When you can express it as "that's just like *x*, except...," then you have the problem solved!

15.4 Defining New Environments

Defining completely new environments is usually the job of a .MAK file, which contains document type definitions, in the Scribe Database. While you are waiting for the Database doctor to arrive, you can define your own new environments in a manuscript file. Be wary if you find yourself wanting a new environment at every step. You might simply not be familiar with the standard ones provided by Scribe, or you might be able to modify others to suit your needs.

You can define new environments by basing them on existing environments (much like the @Modify command) or by creating them completely from scratch. In either case, you use the @Define command to define a new environment. The guidelines for where to place the @Define commands and when they are then in effect is the same as for the @Modify command. They can be included anywhere in your .MSS file, but we urge you to place them in the beginning of the file.

15.4.1 New Environments From Old

Defining a new environment based on an existing one requires both the name of the old environment and the list of differences:

@Define (*NewName=OldName, <list-of-differences>*)

This command creates a new environment by first copying all of the attributes and values for the old one. It then takes the list of differences, changes the values for attributes already in the definition, and adds any new attribute-value pairs. It is much the same as @Modify, except that it leaves the original environment unchanged.

As an example, suppose you have a document that requires an extra kind of example, one that is like ProgramExample, but capitalizes all of its text and ensures that the entire text is together on one page:

```
@Define(CapProgramExample=ProgramExample,
            Capitalized=True, Group)
```

15.4.2 New Environments From Scratch

Creating completely new definitions can be a tricky business. This chapter does not attempt to explain the process in any detail. n

The best way for you to proceed is by analogy with existing definitions. Read the Database files. Experiment with modifying standard environments until you think you understand what each of their attributes is contributing. Then launch out with your own @Define commands:

@Define (Newname, *<list of attribute-value pairs>*)

15.5 Environment Attributes for the @Define and @Modify Commands

This section contains a list of the most common attributes and the kinds of values they can take on. Some attributes take Boolean values: Justification can be On or Off. Some take a parameter value: Spaces can be Kept, Ignore, Ignored, Compact, Tab, Null, Normalize, Normalized, or NoBreak. Others take numeric values: LeftMargin can be any kind of horizontal distance unit.

This list of the environment attributes is not complete. Some of the Database definitions use attributes that are not in this list. We have omitted attributes that are either esoteric or dangerous. Please don't attempt to use any of the attributes that you happen across in the Database unless they are in this list.

Above Controls the amount of white space above the environment by overlapping with the Below value for the preceding environment. Use a vertical distance, for example, Above 3Lines.

AfterEntry A delimited text string that is evaluated each time the environment is entered. This text will be placed in front of the first actual text of the environment. For example, AfterEntry=<@TabClear()>.

AfterExit A delimited text string that is evaluated immediately after the environment is exited. This text will be placed in front of the text that follows the environment.

Anchor Anchor means the same thing as Group, but it has this name for mnemonic value in the @Modify command. It turns off Float.

Anchored Synonym for Anchor.

BackgroundColor Specifies the color of the background over which this environment will be printed. This attribute is ignored for devices which cannot change background colors. The value of this attribute must be a color defined for your site. See your *DBA* for details.

BeforeEntry A delimited text string that is evaluated immediately before the environment is entered. This text will be placed after any text that precedes the environment but before any text in the environment itself. Differs from AfterEntry in that the BeforeEntry string appears with the attributes that are in effect *before* the environment is entered.

BeforeExit A delimited text string that is evaluated immediately before the environment is exited. This text will be placed after any text that is part of the environment proper.

Below Controls amount of white space below the environment by overlapping with the Above value for the following environment. Use a vertical distance, for example, Below 2cm.

BlankLines Controls what happens to blank lines in the environment. Use a value from the set {Ignore, Ignored, Break, Kept, Hinge, HingeBreak, HingeKeep}. Ignore and Ignored disregard blanklines in the .MSS file completely, resulting in no paragraph breaks. Break converts blanklines into paragraph breaks. More than one blankline together results in a single paragraph break. Kept retains all blanklines. More than one blankline gives the same number of blanklines in the output. Hinge simulates an @Hinge command for each blank line in the environment. HingeBreak both simulates an @Hinge command for each blank line in the environ-

ment and causes a paragraph break. HingeKeep both simulates an @Hinge command for each blank line in the environment and retains all blanklines in the .MSS file. For example, normal text uses BlankLines Break; unfilled text uses BlankLines Kept. Default value: Kept.

Boxed The Boxed attributed must be specified whenever the Columns attribute specifies output in multiple columns. Do not attempt to use this attribute in other contexts or specify a value for it.

Break Controls whether beginning or ending the environment causes a line break. Use a value from the set {Before, After, Around, Off}. Before always causes a paragraph break before an environment. After always causes a line break for the last line in an environment. (The final line break may also be a paragraph break. See the Continue attribute.) Around is the combination of Before and After. Off results in no line breaks before or after the environment. Default value when Break is mentioned: Around. Default value when it is not mentioned: Off

Capitalized Controls whether the text is converted to all upper case or left alone. Use a Boolean value. Default value: True.

Centered Controls placement of a line between prevailing margins. Centered does not have a value. Centered, Fill, FlushLeft and FlushRight are mutually exclusive attributes. Only one of them can be in effect at a time.

Color Specifies in what color the running text of the document should be printed. The value of the attribute must be a color defined for your site. See your ***DBA*** for details.

ColumnMargin A horizontal distance indicating the space between columns in a multi-column environment. Default unit: Characters.

Columns A positive integer indicating the number of columns into which the text of the environment is to be formatted.

ColumnWidth A horizontal distance indicating the width of each column into which the text of the environment is to be formatted. It is the sum of ColumnMargin and LineWidth.

Continue Controls whether the line following this environment resumes the same paragraph or starts a new one. It applies only when this environment definition also specifies either Break Around or Break After. Use a value from the set {Allowed, Force, Forced, Off}. Allowed tells Scribe that a new paragraph may be started after the environment is ended; the final decision is based on whether there is a blank line between the end of the environment and the next line of text. Off never produces a new paragraph. Force and Forced always results in a new paragraph. Default value when Continue is mentioned: Allowed. Default value when it is not mentioned: Off.

Copy References an existing environment and copies that environment's definition into the new environment. Differs from Use in that the copy is made right away rather than at the new environment's invocation. The difference is only significant if the environment being copied will be changed before the new environment's invocation.

Counter Controls which counter applies to the environment and which counter @Tag finds. Use a counter name as the value, for example, Counter FigureCounter.

CRBreak Controls whether a carriage return in the manuscript file causes a para-

graph break in the document. Use a Boolean value, for example, CRBreak On means every carriage return causes a paragraph break. Default value: True.

CRSpace CRSpace causes carriage returns in the environment to be treated normally; that is, as one or two spaces, depending on whether the character preceding the carriage return is a sentence-ending punctuation mark. It is the equivalent of CRBreak Off.

ExceptionDictionaries
Specifies the exception dictionary to be used when an Automatic hyphenation method is in force. Use a user-defined dictionary name as a value.

ExceptionDictionary
Synonym for ExceptionDictionaries.

FaceCode Selects a FaceCode from those available for the current font. Use a FaceCode name, for example, FaceCode R.

Fill Specifies line filling, that is, putting as many words on a line as possible. Fill has no value to specify. Centered, Fill, FlushLeft, and FlushRight are mutually exclusive attributes. Only one of them can be in effect at a time. (See Justification attribute.)

Fixed Places text for this environment in a fixed vertical position on the page. Use a relative vertical distance, for example, Fixed 4.4inches. Positive values specify the distance of the top of the environment from the top of the paper; negative values specify the distance of the bottom of the environment from the bottom on the paper. Default unit: Lines.

Float Permits text in the environment to be moved along to the first page in the document where the whole environment fits on a page. Float has no value to specify. Figure and Table environments have the Float attribute set normally. Float, Group, FloatPage, and Free are mutually exclusive attributes. Only one of them can be in effect at a time.

FloatPage Moves text in the environment to the next available whole page. FloatPage has no value to specify. The FullPageFigure and FullPageTable environments have the Floatpage attribute set normally. Float, Group, FloatPage, and Free are mutually exclusive attributes. Only one of them can be in effect at a time.

FlushLeft Flushes each line of text in the manuscript left against the prevailing left margin. FlushLeft has no value to specify. Centered, Fill, FlushLeft, and FlushRight are mutually exclusive attributes. Only one of them can be in effect at a time.

FlushRight Flushes each line of text in the manuscript right against the prevailing right margin. FlushRight has no value to specify. Centered, Fill, FlushLeft, and FlushRight are mutually exclusive attributes. Only one of them can be in effect at a time.

Font Specifies a font for this environment. Use a font name (from those specified in the Database for this device), for example, Font BodyFont.

Free No grouping constraints on the text in this environment. Its major application is for modifying other grouping attributes. Free has no value to specify. Float, Group, FloatPage, and Free are mutually exclusive attributes. Only one of them can be in effect at a time.

Group Starts a new page if insufficient space remains on the current page for this environment. Group has no value to specify. Float, Group, FloatPage,

and Free are mutually exclusive attributes. Only one of them can be in effect at a time.

Hyphenation Specifies the method of hyphenation that is in effect for the environment. Use a value from the set {AutomaticExact, AutomaticFolded, DictionaryExact, DictionaryFolded, Off, False, No, On, Yes, True, Old, OldExact, OldFolded, Warn}. See SubSection 10.1.3 for details about the possible values. Default value when Hyphenation is mentioned: Old. Default value when it is not mentioned: Off.

HyphenationDictionaries
Specifies the hyphenation dictionary or dictionaries to be used when a Dictionary hyphenation method is in force. Use a user-defined dictionary name as a value.

HyphenationDictionary
Synonym for HyphenationDictionaries.

HyphenBreak A boolean that specifies whether hyphens (minus signs) in the .MSS file can be taken as hyphenation points. This kind of hyphenation is independent of that specified by the Hyphenation attribute. Default value: True.

Increment Requires a counter name. Increments the counter each time the environment is invoked.

Indent Controls the displacement of the indented margin from the prevailing margin. Specified by horizontal distance relative to the prevailing left margin. For example, Indent 0, Indent +3, or Indent -5.

Indentation Synonym for Indent.

Justification Justified text has the words in the text aligned with the prevailing right margin. Justifying requires filled text (see Fill attribute). Use a Boolean value. Justification On for an individual environment requires global justification on (see @Style). Default value: True.

LeadingSpaces Controls what to do with leading spaces in this environment. Use a value from the set {Ignore, Ignored, Compact, Kept, Null, Tab, Normalize, Normalized, NoBreak}. Ignore and Ignored produce no spaces in the output, but the spaces still cause word breaks. Null disregards leading spaces and does not cause word breaks. NoBreak retains the spaces, but they do not cause word breaks. Compact reduces the number of spaces to one between words and after commas and semicolons and two after sentence-ending punctuation. Normalize and Normalized reduce the number of spaces to one in all cases. Kept retains all spaces, regardless of position. Tab reduces all cases of three or more spaces to a tab command. Default value in unfilled environments: Kept. Default value in filled environments: Ignored.

LeftMargin Controls width of the left margin. Use a horizontal distance. Absolute distances are measured from the global left margin; signed distances are measured from the prevailing left margin. For example, LeftMargin 5, LeftMargin +0, or LeftMargin -16. LineWidth, LeftMargin, and RightMargin all contribute to defining line layout. You can specify at most two of the three.

LineWidth Specifies the maximum width of a line of text. Use a horizontal distance, for example, LineWidth 65. In a multi-column environment, LineWidth specifies the width of *one* column of text. LineWidth, LeftMargin, and

RightMargin all contribute to defining line layout. You can specify at most two of the three.

LongLines Specifies what Scribe is to do with unfilled lines that are too long for the current margins. Use a value from the set {Chop, Keep, Wrap}. Chop truncates lines at the right margin and prints an error message. Keep permits lines to be printed past the right margin. Wrap prints a line to the right margin and then continues that line onto the next output line, begun at the indent margin (see the Indent attribute). Default value: Chop.

Need Starts a new page unless the amount of space requested remains on the current page. Use a vertical distance, for example, Need 8lines.

NoFill Synonymous with FlushLeft.

Numbered Specifies a format for numbering the paragraphs in the environment. Use a counter template as a value, for example, `Numbered <@1. >`. (See also the Referenced and NumberFrom attributes and Section 15.8.)

NumberFrom Gives an initial value for numbering the paragraphs. Use a numeric value, for example, NumberFrom 4. (See Numbered.)

NumberLocation Controls where paragraph numbers appear. Use a value from the set {LFL, LFR, RFL, RFR}. The first letter specifies the margin, left (L) or right (R). FL and FR stand for Flush Left and Flush Right. LFL and RFR relate to the global margins; the number is placed *within* the global margin. LFR and RFL relate to the prevailing margins; the number is placed *outside* the prevailing margins. Default value when NumberLocation is mentioned: LFR. Default value when it is not mentioned: RFR.

OverStruck Specifies the number of extra strikes to obtain boldface on devices capable of overstriking. Use a numeric value, for example, Overstruck 1.

Pagebreak Controls whether beginning or ending the environment causes a page break. Use a value from the set {Off, Before, After, Around, UntilOdd, UntilEven, EvenAround, OddAround}. Before ensures that the environment begins on a new page. After ensures that the text immediately after the environment begins on a new page. Around is a combination of Before and After. UntilOdd ensures that the environment begins on an odd-numbered page, even if an even-numbered blank page must be produced. UntilEven ensures that the environment begins on an even-numbered page, even if an odd-numbered blank page must be produced. UntilOdd and UntilEven are the same as Before if the document is singlesided. Off causes no pagebreaks before or after the environment. EvenAround is a combination of UntilEven and After. OddAround is a combination of UntilOdd and Around. (See the @Style parameter DoubleSided.)

Pageheading Controls what happens to running headers during this environment. Use a Boolean value. Default value: True.

Pageheadings Synonymous with Pageheading.

Referenced Like Numbered, but provides a template to be used for cross references to this generated number. The Enumerate environment, for example, specifies `Numbered <@1. >, Referenced <@1>`, so that the cross-reference tags won't pick up the period and space in the Numbered template.

RightMargin Controls width of the right margin. Use a horizontal distance. Absolute distances are measured from the global right margin; signed distances are

measured from the prevailing right margin. For example, RightMargin 0.5inch, RightMargin +0, RightMargin -10. RightMargin, LineWidth, and LeftMargin all contribute to defining line layout. You can specify at most two of the three.

Script Controls the position of the baseline of this environment relative to the ordinary text baseline. Use a signed vertical distance, for example, Script +0.5 lines.

Sink The Sink attribute is similar to Above, save that it specifies a minimum distance from the top text margin rather than from the previous environment. Sink takes a vertical distance as a value. If an environment has, for example, the attribute "Sink 2 inches", then the first line of text in that environment will be 2 inches from the top edge of the normal text area of the page, which is 2 inches + TopMargin from the top edge of the paper.

Slant Specifies a counterclockwise angle by which the characters in this environment will be slanted. Meaningful only for devices that can slant characters automatically. Positive angles are measured counterclockwise; negative angles clockwise.

Spaces Controls what happens to spaces from the manuscript file in this environment. Use a value from the set {Kept, Compact, Ignore, Tab, Null, Ignored, Normalize, Normalized, NoBreak}. Ignore and Ignored produce no spaces in the output, but the spaces still cause word breaks. Null results in no spaces in the output and no word breaks. NoBreak retains the spaces, but they do not cause word breaks. Compact and Tab reduces the number of spaces to one between words and after commas and semicolons and two after sentence-ending punctuation. Normalize and Normalized reduce the number of spaces to one in all cases. Kept retains all spaces, regardless of position. Default value for filled environments: Compact. Default value for unfilled environments: Kept.

Spacing Controls the distance from the baseline of one line to the baseline of the next in this environment. Use a vertical distance, for example, Spacing 1 means single spacing. Warning: Spacing 2 corresponds to double spacing, but because of typographical considerations, it does not use twice as much space as Spacing 1 produces. See the *Scribe Database Administrator's Guide* for more information.

Spread Controls the amount of space that is added to the Spacing value to give the vertical space between paragraphs. Use a vertical distance, for example, Spread 1 with Spacing 1 gives one extra line between paragraphs.

TabExport Controls whether tab settings from this environment are cleared at the end of it or left for the surrounding environment. True keeps all tab settings; False clears them. Default value: True.

Underline Controls which characters in the environment are underlined. Use a value from the set {Off, Alphanumerics, NonBlank, All}. Off causes no underlines. All underlines words, spaces, and punctuation. NonBlank underlines words and punctuation. Alphanumerics underlines words. The environment "U" specifies NonBlank. The environment "UN" specifies Alphanumerics. The environment "UX" specifies All. Default value when Underline is mentioned: NonBlank. Default value when it is not mentioned: Off.

UnNumbered Specifies that an environment does not have numbered paragraphs. Its only application is for modifying environments with the Numbered attribute. UnNumbered has no value to specify.

Use Takes an environment name as a value and has the effect of including the definition of the referenced environment in the one under definition. The effect of Use is not the same as the effect of Copy. See Copy for further details about their differences.

WidestBlank A horizontal distance which partially controls hyphenation. If Scribe is about to create a word space whose size exceeds WidestBlank, it will attempt instead to hyphenate the first word on the next line to avoid this, if Hyphenation is specified. Of all of the parameters associated with hyphenation, WidestBlank is the weakest in the sense that it will be violated before any of the others. For example, Scribe will never attempt to hyphenate a word having five or fewer letters, no matter how wide a word space it must leave. Default unit: Characters.

Within Provides a parent counter for a template defined in a Numbered attribute. Use a globally-defined counter name, for example, Within Chapter.

Figure 15-1 on page 171 shows pictorially the relationship of some environment attributes to one another and to the page.

15.6 Library Files for Commands

Section 13.1 described how to include one file in another by using the @Include command. Files included this way cannot contain any of the commands that must go at the beginning of a manuscript file. Where should you put commands like @Define and @Modify?

A few definitional commands can just go directly into your manuscript file. Large numbers of specialized commands should go in a separate document type (consult the *Scribe Database Administrator's Guide*). In-between cases are ones where you have a set of custom definitions large enough to make it tedious to type them separately in each manuscript file but you don't have enough to make it worth the work of creating a new document type. In these cases, you need to create a *Library file* for the definitions.

A Library file is basically just a miniature Database file (just like the document type and device type files) containing definitions. You need to tell Scribe in the manuscript file where to find your definitions by using the @LibraryFile command. It is much like @Include except that it is for files full of commands instead of files full of text. In fact, a Library file must contain only commands — no text and no commands that generate text, such as the @Include command, are permitted.

The syntax of the command is simple:

@LibraryFile (*library-file-name*)

Some examples of possible @LibraryFile commands are

```
@LibraryFile(Formulas)
@LibraryFile(InfoKeys)
```

When it looks for your definition file, Scribe uses the first six letters of the Library filename and the file extension ".LIB". Thus, for the Library files above, Scribe would look for FORMUL.LIB and INFOKE.LIB.

So much for the manuscript file. In the Library file itself, you need a command line to tell Scribe the full name of the Library. This line is the @Marker command, and it has to be the first line in the file. For the example above, the @Marker commands would be:

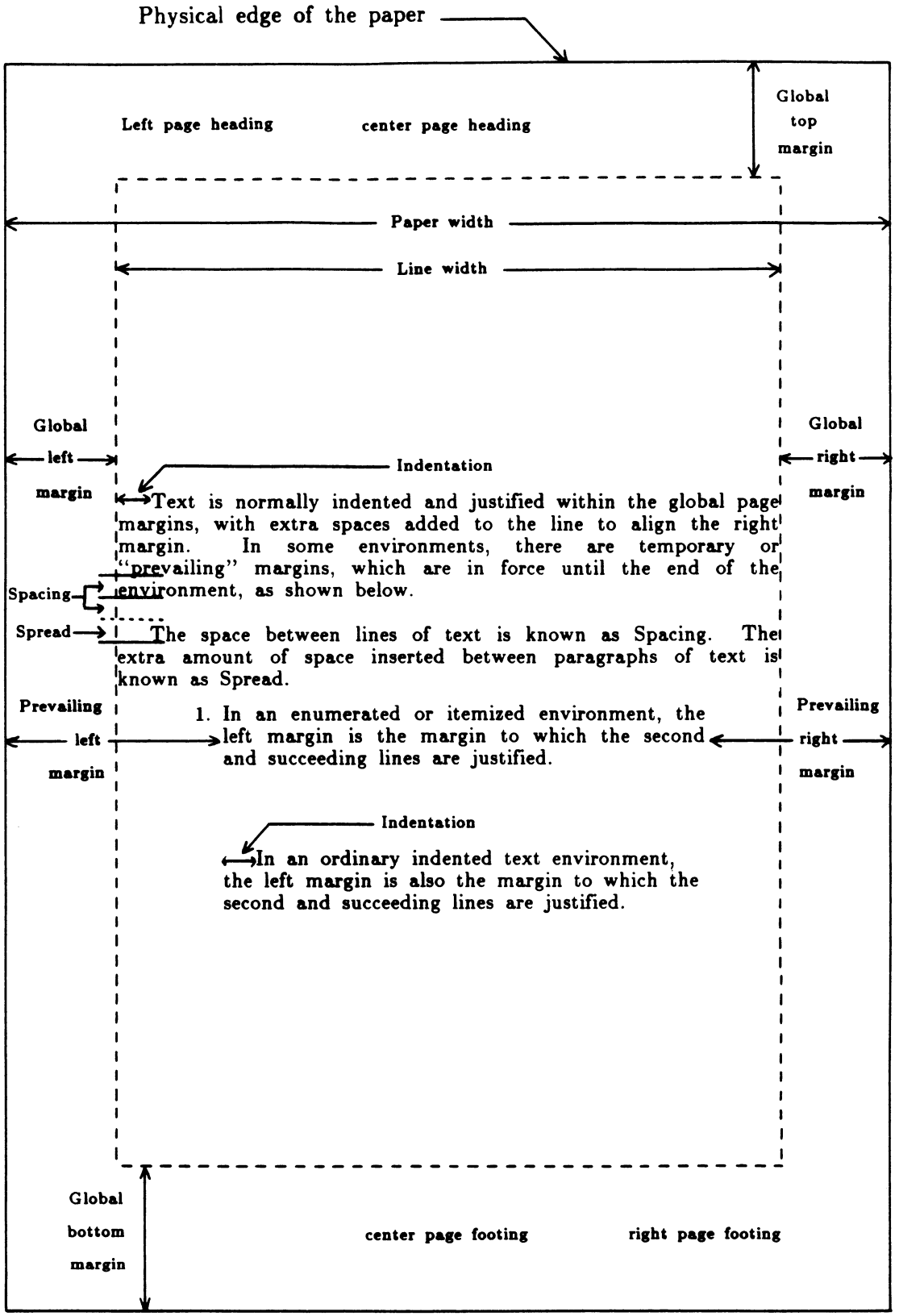

Figure 15-1: Page Layout

```
@Marker(Library,Formulas)
@Marker[Library,InfoKeys]
```

If you look at any of the Scribe Database files, you will see more complex-looking @Marker commands. Explaining those commands is beyond the scope of this manual.

15.6.1 Multiple-Level Indexing

Simple indexing was discussed in Section 4.4. Two Scribe forms produce multiple-level Index entries: **@IndexSecondary** and **@SeeAlso**. These forms are discussed in this Section because they require the @LibraryFile(MultiLevelIndex) command be included in the .MSS file. The file `multil.lib`, which is read by Scribe as a result of the @LibraryFile command, contains the definitions of the two multiple-level indexing forms. If you try to use one of those forms without including the @LibraryFile command, Scribe will produce error messages about the forms @IndexSecondary and @SeeAlso not being defined.

In a multi-level Index, there are two types of Index entries: multiple-line secondary entries and multiple-line reference entries. The @IndexSecondary and @SeeAlso forms correspond to those entries, respectively.

15.6.1.1 The @IndexSecondary Form

The @IndexSecondary form gives you a two-level Index entry; the secondary term is listed indented and under the primary term. The syntax of the form is

@IndexSecondary[Primary=*delimited-primary-entry*,
Secondary=*delimited-secondary-entry*]

The parameters of the form are discussed below.

Primary The primary entry. It is printed flush-left in the Index and does not receive a page reference.

Secondary The secondary entry. It is printed indented under the primary entry and receives a page reference.

The primary term of the @IndexSecondary form does not receive a page number. If you need that term to have a reference, don't fret; you simply insert an @Index[*primary-term*] command in the text. The @Index command will number the term, and the @IndexSecondary form certainly won't remove that number.

This line in the .MSS file

Manuscript Form:

```
@IndexSecondary[Primary="Documentation production",
        Secondary="Scribe"]
```

will produce this result

Document Result:

```
Documentation production
   Scribe   172
```

To produce output in which ''Documentation production'' also has a reference number, an @Index command is required:

Manuscript Form:

```
@Index[Documentation production]
@IndexSecondary[Primary="Documentation production",
        Secondary="Scribe"]
```

will produce this result

Document Result:

```
Documentation production  173
   Scribe   173
```

15.6.1.2 The @SeeAlso Form

There are times when under a term in the Index, you want to refer readers to some other term in the Index. The @SeeAlso form allows you to accomplish that task. The form produces an entry that, like the @IndexSecondary form, is listed under a primary entry and indented.

The syntax of the form is

@SeeAlso[Primary=*delimited-primary-entry*,
Other=*delimited-reference-entry*]

The parameters of the form are discussed below.

Primary	The primary Index entry. It receives no page reference.
Other	The reference term. It is labeled ''See also'' and receives no page reference.

How much the reference term listed as ''See also'' is indented depends on whether or not there is an @IndexSecondary form which lists the same Primary term as the @SeeAlso form. Consider the following examples.

Manuscript Form:

```
@SeeAlso[Primary="Index", Other="Library files"]
```

Document Result:

Index
  See also Library files

Manuscript Form:

```
@SeeAlso[Primary="Index", Other="Library files"]
@IndexEntry[Primary="Index", Secondary="Multi-level index"]
```

produce this Index entry:

Document Result:

Index
  Multi-level index 173
    See also Library files

The @SeeAlso form does not give any page reference in the entry because it's assumed that there would be an Index entry for the term listed as "See also", and so the user would simply cross reference that term in the Index. The entry for that text can be achieved with the @Index command or either of the other indexing forms. For example, the above illustration can be expanded to show the "Library files" entry:

Manuscript Form:

```
@SeeAlso[Primary="Index", Other="Library files"]
@IndexEntry[Primary="Index", Secondary="Multi-level index"]
@Index(Library files)
```

Document Result:

Index
  Multi-level index 174
    See also Library Files

Library files 47

15.7 Counters

Scribe counters count things. They are similar to environments in that they have definitions which consist of pairs of attributes and values. Counters are defined with @Counter commands.

Scribe counter definitions provide a name, a rule for when to add one to the counter, and a pair of templates to use for printing the counter value:

```
@Counter(TheoremCounter, Numbered <@1.>, Referenced
          <@1>, IncrementedBy Reference)
```

This chapter does not explain Scribe counters in enough detail for you to define your own or to attempt major modifications to existing counters. It does show you how to make minor modifications, though. Only three attributes of counters are often candidates for modification: the Numbered, Referenced, and Within attributes. Numbered and Referenced both assume template values. (Templates are discussed in Section 15.8.) The Numbered template generates the number that is actually printed for the thing being numbered — the page number, the theorem number, or the chapter number. The Referenced template generates the number that is used when the thing is cross referenced. The Within attribute works for counters the same way it does for environments; it provides a parent for the counter to be numbered within.

You modify counters the same way as you modify environments — with the @Modify command. Two examples are shown below. The first example changes the numbering of the Appendix to "Appendix A.", "Appendix B.", ... , "Appendix F." and the references to the

Appendix to "A", "B", ... , "F". The second changes the page numbering from beginning at one and continuing throughout the document sequentially to being reset at each chapter (referred to as page numbers within chapters).

```
@Modify(Appendix, Numbered <Appendix @A.>, Referenced
        <@A>)

@Modify(Page, Within Chapter)
```

Remember that you can also control the appearance of some counters' values, for example, page and footnote numbers, with @Style commands. See Chapter 4, Section 4.5 for more information.

15.8 Templates

Numbering *Templates* specify the format or appearance of counter values. A template is a very simple pattern that is used to convert a number into a string of characters that represents the number. For example, a template can convert the number 1 into the character "1", into the letters "One", into the characters "001", or into the Roman numeral "I".

Templates contain a mixture of template codes and literal text. The template is *expanded* to produce an actual text string, which is then processed by Scribe as if you had typed the expanded form in your manuscript file. Thus, if the expanded template has any Scribe commands in it, they are processed as if you had just typed them in.

In a template, all codes are flagged by an "@" character; anything not flagged by an @-sign is literal text and goes into the expanded string unchanged. To get an "@" character to appear in the expanded text string so that it can be part of a Scribe command after the expansion, use the special code, @@@@, which copies an @-sign literally into the output string.

When templates are expanded into text strings, a literal character in a template is copied directly into the string for the counter. In strings like "Figure 3-1" and "Figure 3-2", the hyphen is a literal character, but the 3, the 1, and the 2 were generated by template codes. Every character that is prefixed by an @-sign is a special template code. The template code controls what goes into the counter string. For example, the template code that puts ordinary digits into a counter string is `@1`.

The best way to understand numbering templates is to examine some templates that control familiar numbers. All templates in this list are delimited by angle brackets, but any delimiter pair is fine.

Counter	*Template*
Enumeration counter	`<@1. @,@a. @,@i. >`
Page number	`<@1>`
Chapter counter	`<@1.>`
SubSection counter	`<@#@:.@1>`
Equation counter	`<(@1)>`
FootNote counter	`<@@+[@1]>`

Appendix counter `<@I.>`

Although template codes are prefixed with an @-sign, they are not at all similar to ordinary Scribe commands. There are a small number of template codes that are named the same as Scribe environments, for example @i. Even though the same name is used for two very different purposes, Scribe will not get them confused because there are only a few instances where Scribe expects to see a template: as values to the Numbered and Referenced attributes or as a value to the PageNumber or Footnotes @Style parameter.

Figure 15-2 below is a list of the various template codes and what they do.

Figure 15-2: Codes for Numbering Templates

Code	*Result*
@1	Prints arabic cardinal numbers (1, 2, 3, ...) starting from 1.
@2 ***through*** **@9**	Prints the specified number of arabic cardinal numbers starting from 1 with leading zeros, if necessary, to fill out the number to the required length. For example, @4 produces the numbers (0001, 0002, 0003, ...).
@'	Prints abbreviations for ordinals (1st, 2nd, 3rd, ...).
@A	Prints capitalized alphabetical sequence (A, B, C, ...) starting from A.
@a	Prints lowercase alphabetical sequence (a, b, c, ...) starting from a.
@F	Prints ordinal numbers with the first letter capitalized. (First, Second, Third, ..., Ninety-Ninth) starting from First.
@f	Prints lowercase ordinal numbers (first, second, third, ..., ninety-ninth) starting from first.
@I	Prints "capitalized" Roman numerals (I, II, III, IV, ...) starting from I.
@i	Prints "lowercase" Roman numerals (i, ii, iii, iv, ...) starting from i.
@O	Prints cardinal numbers with the first letter capitalized (One, Two, Three, ..., Ninety-Eight, Ninety-Nine) starting from One.
@o	Prints lowercase cardinal numbers (one, two, three, ...) starting from one, up to ninety-nine.
@*	Prints a sequence of asterisks (*, **, ***, ****, ...) up to a maximum of 12.
@#	Prints the value of the parent counter for this counter. (The Within attribute in a counter definition specifies a parent counter.) For example, the definition for the SubSection counter includes *Within Section*, meaning that the Section counter is the parent counter for SubSection. The number printed by the @# command is the value of the Referenced attribute, not the Numbered attribute. (Often the values of the two attributes are identical except for punctuation.)
@:character	When a parent counter exists and has a nonnull value, Scribe prints the specified character following the colon. That is, the character is a literal that appears only when the counter has a parent counter with a defined value.

@;*character*	When no parent counter exists (see @#), it prints the specified character following the semicolon. That is, the character is a literal that is printed only when there is no parent counter for the counter being defined.
@,	Separates templates in a list. With a list of values, Scribe changes to the next template in the list each time the environment is nested within itself (for example, Itemize and Enumerate).
@$	Selects one of several literals, depending on the value of the counter. Unlike other template codes, the @$ code requires a delimited string as an argument; that delimited string must contain a series of literals separated by @, sequences. If the value of the counter is 1, the first literal is used; if the value of the counter is 2, the second literal is used, etc. Since the characters between the @, sequences are literal, you cannot use other template codes in there, and you do not double "@" characters if there are Scribe commands in the literal.
character	Any character not prefixed by an @-sign is a literal and will be copied directly into the output string without interpretation.
@@	Two @-signs in a row result in a Scribe command being executed. In the example of the Footnote counter on page 175, the two at-signs result in the superscripting of the appropriate footnote number.
@@@@	Four @ characters in a row result in a literal at-sign in the output.

15.9 Fonts

Most changes to fonts involve Database modifications, which are not discussed in this manual. If you just want to use a private font for your printing device, and you can get enough local help from systems programmers at your site to know how to get the font file accessible to Scribe, then all you need to do to use it is to declare it with the @SpecialFont command and refer to it with one of the special font environments @F0 through @F9.

To declare a special font, use the @SpecialFont command:

```
@SpecialFont(F2="Baskerville12BIR")
@SpecialFont(F0="25VG")
@SpecialFont(F1="SHD40")
```

These commands declare, respectively, special fonts 2, 0, and 1. The names given to them are for all accounts and purposes black magic: If you have access to a font named "25VG", then you will recognize the name instantly; if you don't know what it means, then you don't have access to it. These special fonts are, in general, only available on the various Xerox printing devices known to Scribe, although it is possible to construct special fonts for most devices.

Having declared a special font, you use it via the environment of the same name:

```
Text in @F2(Special font 2)

@F0[This text] is in special font 0

@Begin(F1)
A whole line of text in special font 1
@End(F1)
```

Most uses of the @SpecialFont command are sufficient only for limited purposes, such as giving you access to a single special character in a manufacturer's font. You don't want to use @SpecialFont to produce a document in French — it's just too restricted in scope. See your *DBA* for more advanced font facilities.

Chapter Sixteen
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 don't care to know, 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 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 yields a completely different set of formats which were designed to look attractive on the new device.

The cost of this approach is that it is not always easy 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 correctly. If you catch yourself constantly thumbing through the list of commands and @Style parameters trying to find the command or parameter that will force some particular character into a particular column, then you are not using Scribe properly. Think carefully about what that problem 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 solution 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 requesting a different Form in the @Make command; see Section 5.3 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 get a listing of the Database files, take your copy of the *Scribe Database Administrator's Guide*, curl up in a comfortable place, and read them "cover to cover". If that doesn't stop you, then roll up your sleeves, and go make your own document style.

One final behest: If you *do* break down and design your own document type, and if you

think it's good enough that other people might want to use it, please send it back to us (documented) so that it can be included in Scribe distribution for others to use. Share the wealth.

Appendix A
Operating Systems Dependencies

This Appendix contains information that is operating system-dependent. While Scribe attempts to be as consistent as possible at the user level across system boundaries, some differences cannot be hidden. One example is the command you must type at the terminal to run Scribe. A feature that is not system-dependent is that you may omit the extension .MSS when specifying a manuscript filename.

Some command-line options are system-dependent. Those options are given in this Appendix. To be consistent with the format of Appendix E.1 in which options valid for all operating systems are discussed, an option and any of its valid abbreviations are listed on one line with choices in boldface and separated by commas.

A.1 TOPS-10 Systems

To run Scribe on a TOPS-10 system, type "r scribe". The command-line switch character for TOPS-10 is "/".

The following command-line switches are valid for TOPS-10, in addition to those listed in Appendix E.1:

NoH, NoHyp, NoHyphenate
: Turn off hyphenation.

NoHV, NoHypVocab
: Don't create a Scribe .LEX file. This option differs from NoHyd in that it turns off the .LEX file that would have listed the hyphenation points of each word in the document.

NoHyd
: Don't create a Scribe .LEX file. This option differs from NoHV in that it turns off the .LEX file that would have listed each hyphenation decision.

On a TOPS-10 system, you can type the following at Scribe's command line:

```
*output.ext=file.mss
```

where *.ext* is the extension that Scribe will use for the OUTPUT file. This line allows you to specify the name of the output file you desire. In the above example, Scribe would normally create the FILE.EXT file, but will now create the OUTPUT.EXT file. To look at another example, if the file DOCS.MSS file had the @Device(x9700) command in it, you could have Scribe

produce the file OTHER.X97 instead of the expected file DOCS.X97 by typing this line at the command line:

```
*other=docs
```

This same result can be achieved by using the "Doc:" or "Document:" command-line switch, which is discussed in Appendix E.1 and is available for all operating systems.

A.2 TOPS-20 and TENEX Systems

To run Scribe on TOPS-20 and TENEX, type **scribe**. You may type the filename on the command line:

```
@scribe file.mss
```

The command-line switch character for TOPS-20 and TENEX is "/".

As on a TOPS-10 system, you can type the following at Scribe's command line when you're on a TOPS-20 system:

```
*output.ext=file.mss
```

where *.ext* is the extension you want for the OUTPUT file. This line allows you to have Scribe write out the output into a file other than the file it would normally use. In the above example, Scribe would normally create a file named FILE.LPT file, but will now create OUTPUT.EXT instead. To look at another example, if the file DOCS.MSS file had the @Device(x9700) command in it, you could have Scribe produce the file OTHER.X9700 instead of the expected file DOCS.X9700 by typing this line at the command line:

```
other=docs
```

If you want to display .LPT and .DOC files at a CRT under TOPS-20, be sure that you have set "TERM NO INDICATE"; otherwise, the last line of each output page will be overwritten by a displayed "^L" that is output after each page.

A.3 VMS Systems

Scribe runs on the VAX computers in native mode. To run Scribe under VMS, type **scribe**. The command-line switch character for VMS is "/".

A.4 UNIX Systems

To run Scribe on a UNIX system, type **scribe**. The command-line switch character for UNIX is "-".

A.5 Apollo/Aegis Systems

To run Scribe on the Apollo/Aegis operating system, type **scribe**. The command-line switch character for Aegis is "-".

A.6 Prime/Primos Systems

To run Scribe on the Primos operating system, type **scribe**. The command-line switch character for Primos is "-".

A.7 IBM VM/CMS Systems

To run Scribe under CMS type **scribe**. IBM Scribe allows filenames to be written with either . or a space separating the components. For example, a filename can appear as either TEST MSS A or TEST.MSS.A. Most Scribe documentation uses the . character. The command line follows normal CMS conventions and can be used to specify device type as follows:

```
SCRIBE TEST MSS (DEVICE=X9700)
```

Appendix B
Printing Devices

We've tried to make the explanations in the main body of the manual relatively independent of printing-device characteristics. However, at some point, we've got to get down to the nitty-gritty and talk about the printing devices, what they are, and how to get your files printed on them.

Printing devices generally fall into four different *classes*, often with variations.

1. Line-printer class devices. These devices have fixed character width and positioning and fixed vertical spacing. They cannot print subscripts or superscripts and cannot change fonts. The devices in this class include line printers, most hardcopy terminals (DECwriters, TI Silent 700), and most CRT terminals.

2. Diablo-class devices. These devices are somewhat more flexible than line printers, because they are able to position characters rather finely, normally to within 1/100" or so. They can print subscripts and superscripts, and they can use line spacing that is not an integral number of lines, for example, spacing 1.2. The Diablo HyType 1600-series terminals, the Xerox 1700-series terminals, the Agile A1, the Diablo 630 and 630 ECS, the NEC Spinwriter, the Qume, the Santec S700 Variflex, the DTC300, AJ-832, and the AJ-833 fall into this class.

3. Photocomposers. These devices are optical/mechanical devices that produce output by forming photographic images of letters on light-sensitive paper and then developing that paper with standard photographic chemicals. The print quality is unsurpassed, but they are slow (2 to 10 minutes a page). Scribe can produce output for the Mergenthaler Linotype VIP, Compugraphic 8600, Mergenthaler Omnitech 2000, Graphics Systems Incorporated C/A/T-4, and Graphics Systems Incorporated C/A/T-8, the equivalent of the Wang 38 and others.

4. Laser printers. This class of printing devices, also known as non-impact printers, offers a rather astounding set of capabilities. Now a single printing device is able to handle reasonably well such diverse applications as high-speed data printing, letter-quality office applications, and even graphics arts printing with multiple fonts and point sizes. Laser printers are cleaner than photocomposers, since they require no chemicals or developers, and quieter than letter-quality and line printers, since they utilize an electronic non-impact technology. Scribe can produce output for a wide spectrum of these printers, including the Quality Micro Systems Lasergrafix 1200, Symbolics LGP-1, Xerox Dover, Xerox 9700, Xerox 8700, Xerox 2700, Xerox 2700 Model II, Imagen Imprint-10 and the other Imagen printers.

B.1 Line Printers

Most computer users are quite familiar with the line printer, as it is normally their primary printing device. Scribe produces files for the line printer that already have heading and footing information in them; many systems insist on adding a heading and footing as the page is being printed, causing the page to overflow. Contact your ***DBA*** to find out how to print a file without having the system generate extra headings. It might be the LIST or PRINT command that you need to use; you might have to use the COPY command to copy a file directly to the line printer, or there might be a special command like LLIST or VPRINT.

If your line printer does not have very good print quality, you might find that underlined letters look more like w-than like w. If this situation is the case, you might want to try setting the underline character to be a period, using the UnderscoreCharacter parameter to the @Style command (see page 217 for information about the @Style command<).

The Printronix line printer is inexpensive and popular, but it has a quirk that makes it not entirely compatible with ordinary line printers. To underline text on a Printronix printer, the text must be sent first and then followed with an underline. Scribe normally prints the underline first and then follows with the text, so that if the file is typed on a CRT, the text will be visible. @Device(Printronix) sets a flag that causes Scribe to generate an underline sequence that will work on the Printronix; in all other respects, @Device(Printronix) is the same as @Device(LPT).

B.2 Diablo Typers and Similar Devices

A Diablo is a robot typewriter that is capable of very fine gradations of horizontal and vertical spacing and can achieve a very superior print quality. Scribe output files for the Diablo are given the file extension .POD, which stands for Prince Of Darkness (the Spanish word for devil is "diablo"). These files cannot normally be typed directly on the Diablo because they contain binary control codes that most computers do not process properly during a "type" or "print on terminal" operation. Therefore, a program may be necessary to print files on the device. UNILOGIC does not distribute such a program.

When Scribe generates a .POD file, it does so with the assumption that a particular print wheel is mounted on the Diablo. If the expected typewheel is not mounted, letter spacings may be wrong. In particular, Scribe controls all of the horizontal and vertical spacing, overriding the spacing switches inside the machine. Therefore, if you have a Pica typewheel in the machine and have the switches set to 10-character spacing, Scribe still uses 12-character spacing when it prints the file if you specified the Elite typewheel in your .MSS file.

Many Diablo printers are capable of bidirectional printing, in which the print head prints one line on a left-to-right stroke and another line on the right-to-left stroke. Although it is fascinating to watch a printer work this way, studies have shown that the true increase in throughput is less than 5%. Nonetheless, Scribe supports bidirectional printing.

B.3 Photocomposers

There are three basic types of photocomposers: mechanical, CRT, and laser. Mechanical photocomposers such as the VIP or the Wang/Graphic Systems C/A/T Photocomposers are amazing electromechanical devices. The fonts for these machines are actually photographic negatives that mount on spinning wheels. In order to use a new font, a new negative must be obtained.

CRT photocomposers have a cathode ray picture tube (similar to a black and white TV) that they pass the photographic paper over. New fonts generally arrive on floppy disks.

Laser photocomposers are just high-quality laser printers that have been optimized for print quality rather than speed.

B.4 Laser Printers

Laser printers are usually xerographic printing engines similar to office copiers, except that instead of reflecting a bright light off an original to be copied, they use a laser and spinning mirrors to draw the original on the xerographic drum.

Logically (and sometimes physically) there are two parts to a laser printer. First, there is the xerographic marking engine, and second, there is the controller that controls what is put on the page. The marking engine determines the quality of the letters on the page. Buzz terms for describing printers include "How many dots-per-inch" the printer can make and whether it uses "wet" or "dry" toner. The controller determines how the printer looks to your host computer, what commands the printer uses, whether or not it can do graphics, etc. Sometimes, a single marking engine is available with your choice of controllers, or the same controller might be available for a variety of print engines.

Appendix C
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.

C.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. A standard 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.

Various editors have schemes that allow you to type in those 26 extra characters in various ways. Some let you use the CTRL key to generate them as control characters while others have prefix schemes using the ? or ‘ or ^ characters. In general, you should not use those control characters in your manuscript file, because they will not be printed the same way from site to site and device to device. Scribe does not prevent you from putting those characters into your manuscript, even though it probably should.

C.2 Font and Character Considerations for Multiple Font Devices

For use with Scribe, the various character sets have been organized into families, which we call ''FontFamilies''. A Scribe FontFamily 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 FontFamily is stored away in the Scribe Database by name. If you ask for the NewsGothic

FontFamily, for example,[1] by putting the command @Style(FontFamily=<NewsGothic10>) at the front of your manuscript file, then Scribe loads the NewsGothic10 FontFamily definition from its Database. This definition tells Scribe from what character set to get each of the FaceCodes in each of the Fonts. For example, it tells Scribe where to get Italic for BodyFont (running text).

When you type @i[text], what Scribe actually does is to switch to a character set known to contain Italics and prints "text". When you type @g[text], Scribe switches to a character set known to contain Greek and prints "text". Figure C-1 contains a chart of how to print Greek characters. It is *not* necessary to change the print wheel on Diablo-class devices to print Greek characters.

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

Figure C-1: Greek Characters Available with @G

[1] Not every printing device has every FontFamily, of course, but the mechanism by which FontFamilies are changed is the same from one device to the next, even though the FontFamilies aren't identical.

C.3 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 as-key) was proposed in 1963 and approved in 1968 as the official code for the *interchange* of information among computers. DEC computers also use it to *store* 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".

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 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". Many DEC sites have software that uses the Stanford ASCII character code because the folks at Stanford were (and still are) prolific programmers who produced lots of useful programs that were widely distributed. In particular, many editors and text formatters in use at DEC installations use the Stanford ASCII character set or some variant of it.

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 terminology.

The whole problem is exacerbated by the fact that few printer manufacturers supply any fonts that contain the full printing ASCII character set. This means that Scribe has to play tricks on the printer by switching fonts at the right time in order to simulate the appearance of ASCII.

Appendix D
A Few Examples

This appendix contains miscellaneous examples of Scribe usage. The text on each left-hand page is a manuscript file, and the text on the right-hand page is the resulting document file. There is no particular logic to the sequence of these examples.

Manuscript Form:

```
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] geq 10, then 2@+[@i[n]] > @i[n]@+[3].''
Suppose we want @i[to prove that P(n) is true for all positive
integers n]. An important way to do this is:
@Begin(Enumerate)
Give a proof that @i[P(1)] is true; @Tag(BaseStep)

Give a proof that ``if all of @i[P](1),
@i[P](2), . . . , @i[P](@i[n]) are true, then @i[P](@i[n]+1)
is also true''; this proof should be valid
for any positive integer @i[n]. @Tag(InductStep)
@End(Enumerate)

As an example, consider the following series of equations, which
many people have discovered independently since ancient times:
@equation(
1 = 1@+[2], 1 + 3 = 2@+[2],

1 + 3 + 5 = 3@+[2], 1 + 3 + 5 + 7 = 4@+[2],

1 + 3 + 5 + 7 + 9 = 5@+[2].
)
We can formulate the general property as follows:
@equation{
1 + 3 + . . . + (2@i[n] - 1) = @i[n]@+[2] @Tag(Induction)
}
Let us, for the moment, call this equation @i[P(n)]; we wish to
prove that @i[P(n)] is true for all positive @i[n]. Following
the procedure outlined above, we have:
@Begin(Enumerate)
``@i[P](1) is true since 1 = 1@+[2].''

``If all of @i[P](1), . . . , @i[P](@i[n]) are true, then,
in particular, @i[P](@i[n]) is true, so Eq. @Ref(Induction)
holds; adding 2@i[n] + 1 to both sides, we obtain
@begin(equation)
1 + 3 + . . . + (2@i[n] - 1) + (2@i[n] + 1)
  = @i[n]@+[2] + 2@i[n] + 1 = (@i[n] + 1)@+[2]
@End(equation)
which proves that @i[P](@i[n] + 1) is also true.''
@End(Enumerate)

We can regard this method as an @i[algorithmic proof procedure].
In fact, the following algorithm produces a proof of
@i[P](@i[n]) for any positive integer @i[n],
assuming that steps @Ref(BaseStep)
and @Ref(InductStep) above have been worked out.
```

From The Art of Computer Programming Vol. I: Fundamental Algorithms. D. E. Knuth. 1968: Addison-Wesley, Reading MA

Document Result:

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 $P(1), P(2), \ldots, P(n)$ 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:

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

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 $P(1), \ldots, P(n)$ 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.

Manuscript Form:

```
@Heading(Cranberry Bread)
This is a moist, nutty cranberry bread, almost a fruit cake
in terms of texture, that is easy to make and a wonderful
snack.  It's hard to find whole cranberries at any time of
the year other than just before Thanksgiving, so if you
find that you are as addicted to this recipe as we are, you
should stock up on cranberries in mid-November and freeze them.

@SubHeading(Ingredients)
@Begin(Display)
@TabSet(1 inch)
@>2 cups   @\cake flour, measured after sifting
@>1 cup   @\granular white sugar
@>1-1/2 tsp.   @\baking powder
@>1/2 tsp.   @\baking soda
@>1 tsp.   @\salt
@>1   @\Florida orange (juice orange)
@>2 tbsp.   @\Crisco, heated just to melting
@\boiling water
@>1   @\lightly-beaten large egg
@>1 cup   @\chopped walnuts
@>1 cup   @\chopped cranberries, measured after chopping
@End(Display)

@Begin(Itemize)
@i[Sift] together the dry ingredients (flour, sugar, baking
powder, baking soda, salt).  Make sure they are well mixed
before adding the liquids.

@i[Grate] the orange peel off of the orange into a measuring
cup, trying not to get too much of the white rind included.

@i[Squeeze] the orange into the measuring cup on top of the
orange peel.

@i[Add] the 2 Tbsp. of melted Crisco to the rind and juice,
then add boiling water to bring the liquid level to 3/4 cup.
Add the beaten egg and stir lightly.

@i[Mix] the wet ingredients into the dry, immediately, before
the egg starts to cook from the heat of the boiling water,
stirring the resulting mixture as little as possible until it
is uniformly moistened.

@i[Add] the chopped nuts and chopped cranberries.

@i[Pour] into 2 well-greased small loaf pans.

@i[Bake] at 350 for 60 to 70 minutes.
@End(Itemize)
```

From an old Reid family recipe.

Document Result:

Cranberry Bread

This is a moist, nutty cranberry bread, almost a fruit cake in terms of texture, that is easy to make and a wonderful snack. It's hard to find whole cranberries at any time of the year other than just before Thanksgiving, so if you find that you are as addicted to this recipe as we are, you should stock up on cranberries in mid-November and freeze them.

Ingredients

2 cups	cake flour, measured after sifting
1 cup	granular white sugar
1-1/2 tsp.	baking powder
1/2 tsp.	baking soda
1 tsp.	salt
1	Florida orange (juice orange)
2 tbsp.	Crisco, heated just to melting
	boiling water
1	lightly-beaten large egg
1 cup	chopped walnuts
1 cup	chopped cranberries, measured after chopping

- *Sift* together the dry ingredients (flour, sugar, baking powder, baking soda, salt). Make sure they are well mixed before adding the liquids.
- *Grate* the orange peel off of the orange into a measuring cup, trying not to get too much of the white rind included.
- *Squeeze* the orange into the measuring cup on top of the orange peel.
- *Add* the 2 Tbsp. of melted Crisco to the rind and juice, then add boiling water to bring the liquid level to 3/4 cup. Add the beaten egg and stir lightly.
- *Mix* the wet ingredients into the dry, immediately, before the egg starts to cook from the heat of the boiling water, stirring the resulting mixture as little as possible until it is uniformly moistened.
- *Add* the chopped nuts and chopped cranberries.
- *Pour* into 2 well-greased small loaf pans.
- *Bake* at 350 for 60 to 70 minutes

Manuscript Form:

```
@Define(Kind=Center,FaceCode I)
@Define(Choral=Format,AfterEntry "@TabDivide(2)")
@Heading(Christ Before Pilate)
@Kind(Choral)
@Begin(Choral)
Christ, who knew no sin or wrong,@\He who our salvation won,
Like a thief was taken;@\Falsely was convicted
Led before a godless throng,@\Scoffed at, @~
scorned, and spat upon,
By his friends forsaken.@\As the Word predicted.
@end(Choral)

@Kind(Recitative)
Then led away they Jesus, away to the hall of judgment:
and it was early; and they went not themselves therein,
lest there they should be defiled; but that they might eat
the Passover.  Then unto them Pontius Pilate went out, and said:
What accusation bring ye now against this
person?  Then they cried aloud and said unto him:

@Kind(Chorus)
If this man were not a malefactor, we had not brought him
here before thee.

@Kind(Recitative)
Then Pilate said unto them: Now come and take ye him,
and judge ye him according to your law.
The Jews therefore said unto him:

@Kind(Chorus)
By death we may not punish.

@Kind(Recitative)
That so might be fulfilled the word of Jesus which
he had spoken, and had signified by what manner of death
he die.  Then Pilate entered into the hall, and again he
called in Jesus, and said to him: Art thou King of the Jews,
then?  Jesus thus answered him: Sayest thou this thing of
thyself, or did these others tell it thee to say of me?
And Pilate then answered him: Am I a Jew?  Thy people and
thy chief priests have brought thee here for judgment before
me: now what hast thou done?  And Jesus answered him: My kingdom
is not of this world: for were my kingdom of this world, then
my servants all would fight, yea, battle, that I be not
delivered unto the Jews.  Nay then, for not from thence
is my kingdom.
```

From an English translation of Passionmusik nach dem Evangelisten Johannes *(The Passion according to St. John), Johann Sebastian Bach.*

Document Result:

Christ Before Pilate

Choral

Christ, who knew no sin or wrong,
Like a thief was taken;
Led before a godless throng,
By his friends forsaken.

He who our salvation won,
Falsely was convicted
Scoffed at, scorned, and spat upon,
As the Word predicted.

Recitative

Then led away they Jesus, away to the hall of judgment: and it was early; and they went not themselves therein, lest there they should be defiled; but that they might eat the Passover. Then unto them Pontius Pilate went out, and said: What accusation bring ye now against this person? Then they cried aloud and said unto him:

Chorus

If this man were not a malefactor, we had not brought him here before thee.

Recitative

Then Pilate said unto them: Now come and take ye him, and judge ye him according to your law. The Jews therefore said unto him:

Chorus

By death we may not punish.

Recitative

That so might be fulfilled the word of Jesus which he had spoken, and had signified by what manner of death he die. Then Pilate entered into the hall, and again he called in Jesus, and said to him: Art thou King of the Jews, then? Jesus thus answered him: Sayest thou this thing of thyself, or did these others tell it thee to say of me? And Pilate then answered him: Am I a Jew? Thy people and thy chief priests have brought thee here for judgment before me: now what hast thou done? And Jesus answered him: My kingdom is not of this world: for were my kingdom of this world, then my servants all would fight, yea, battle, that I be not delivered unto the Jews. Nay then, for not from thence is my kingdom.

Manuscript Form:

```
@MajorHeading(Blooming Sequence Chart)
@Heading(Plan Carefully for Continuous Spring Bloom)

@Begin(Format)
@TabSet(3 inches,+14,+10)
@b[@=Varieties@\@=Flowering@\
@=in order of appearance@\@=Time@\@=Height]
@Bar()

Snowdrops, Snow Crocus@\@=Early March@\@=3-4"
Windflower@\@=Early March@\@=6"
Spring Snowflakes@\@=Early March@\@=6-9"
Glory of the Snow@\@=March@\@=6"
Crocus@\@=March-April@\@=4-6"
Daffodils (Miniature), Jonquil@\@=March-April@\@=6-12"
Tulips (Kaufmanniana)@\@=March-April@\@=4-6"
Puschkinia@\@=March-April@\@=4-6"
Daffodils (Trumpet, Large-cupped, others)@\@=April@\@=8-20"
Grape Hyacinth, Siberian Squill@\@=April@\@=6-9"
Tulips (Early Single, Fosteriana)@\@=April@\@=10-15"
Hyacinth@\@=April@\@=10"
Fritillaria@\@=April@\@=24-36"
Tulips (Greigii, Mendel, Darwin Hybrid)@\@=April-May@\@=8-30"
Tulips (Darwin, Cottage, others)@\@=May@\@=15-30"
Wood Hyacinth@\@=May@\@=12-15"
Dutch Iris@\@=June@\@=18-24"
Allium@\@=June@\@=6-12"
@End(Format)
```

From a seed and bulb catalog

Document Result:

Blooming Sequence Chart

Plan Carefully for Continuous Spring Bloom

Varieties in order of appearance	Flowering Time	Height
Snowdrops, Snow Crocus	Early March	3-4"
Windflower	Early March	6"
Spring Snowflakes	Early March	6-9"
Glory of the Snow	March	6"
Crocus	March-April	4-6"
Daffodils (Miniature), Jonquil	March-April	6-12"
Tulips (Kaufmanniana)	March-April	4-6"
Puschkinia	March-April	4-6"
Daffodils (Trumpet, Large-cupped, others)	April	8-20"
Grape Hyacinth, Siberian Squill	April	6-9"
Tulips (Early Single, Fosteriana)	April	10-15"
Hyacinth	April	10"
Fritillaria	April	24-36"
Tulips (Greigii, Mendel, Darwin Hybrid)	April-May	8-30"
Tulips (Darwin, Cottage, others)	May	15-30"
Wood Hyacinth	May	12-15"
Dutch Iris	June	18-24"
Allium	June	6-12"

Appendix E
Summaries

A portion of the information in this Appendix also appears in the *Scribe Pocket Reference.*

E.1 Processor Options

Various command-line option switches control some of the things that the Scribe processor does when it runs. You specify an option by including the switch character for your operating system and the desired option. The switch character for the operating systems that run Scribe are listed in Appendix A.

The list of valid command-line options is listed below. Often there is an abbreviated version of an option. All possible options are listed, with choices for the same result separated by commas. Note that only one of the options is necessary and the commas need not be specified.

Option	*Result*
A, Agile	Use output device Agile.
D, Diablo	Use output device Diablo.
Dev:*name*, Device:*name*	Use output device *name*.
Doc:*name*, Document:*name*	Produce output file *name*.
Dover	Use output device Dover.
Draft	Set "Draft" string to "1".
Draft:*value*	Set "Draft" string to "*value*".
F, File	Use output device File.
G, GSI	Use output device GSI.
GG, GIGI	Use output device GIGI.
Hv, HypVocab	Create a lexicon showing the hyphenation points of each word in the document.
Hyd	Create a lexicon showing each hyphenation decision.
Imp, Imprint, Imprint10	Use output device Imprint10.

Keep, KeepFiles	Don't delete Scribe temporary files.
L, LPT	Use output device LPT.
LA36	Use output device LA36.
LGP1	Use output device Lgp1.
PagedFile	Use output device PagedFile.
Q, Quiet	Don't print any error messages on the terminal.
V, Voc, Vocab, Vocabulary	Generate sorted word listing (.LEX file).
W, Words, WordCount	Count the number of words in the document.
X, X9700, 9700	Use output device X9700.

E.2 Document Types

To select one of the document types listed below, put an @Make command in the manuscript file before the first text. For example:

```
@Make(Report)
```

Some document types have variant forms. To use those forms, follow the syntax of this example:

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

Type	*Description*
Article	Simplest sectioned document. Produces numbered Sections, SubSections, Paragraphs, and Appendixes. Figures and Tables are numbered within Sections. There is a Title Page and a Table of Contents, but no Index.
Article, Form 1	Variation of Article with Sections that are not numbered.
Bibliography	Produces a Bibliography only. All other document types are capable of producing a Bibliography, but this document type will provide the Bibliography only, without the accompanying document.
Brochure	Open, informal layout with lots of white space. Provides numbered Chapters, Sections, SubSections, Paragraphs, Appendixes, and AppendixSections. Suitable for producing booklets.
Guide	Similar to Brochure but designed to be printed in a handbook-size format.
Letter	Produces a business letter to be printed on or copied onto letterhead stationary.
LetterHead	Produces a business letter and provides for the letterhead to be drawn. The details of this format vary widely from one site to another. Although any device may be specified when using this document type, specifying this document type for a device that can not draw the logo will result in a message asking you to use Letter.

Manual	Sectioned document providing numbered Chapters, Sections, SubSections, Paragraphs, Appendixes, and AppendixSections. Has a Title Page, Table of Contents, and an Index.
Manual, Form 1	Variation of Manual with numbered Chapters but unnumbered Sections and SubSections.
MilStd837A	Produces a document that conforms to the Military Standard document 837A.
ReferenceCard	Used for printing pocket reference guides and cards. The *Scribe Pocket Reference* was produced using this document type.
Report	Like Manual, but has no Index.
Report, Form 1	Variation of Report with only its Chapters numbered; Sections and SubSections are not numbered.
Slides	Makes overhead projector slides. Font sizes and spacings have been selected to make the slides maximally visible at normal projection distances. Although any device may be specified when using this document type, using a device without font sizes large enough for this type of slide or a device that can not scale will result in a warning message and proof copies only.
Slides, Form 1	Variation of Slides with small- and medium-sized characters in addition to the large-sized characters.
Text	Default document type as Scribe is distributed.[1] Unindented, justified paragraphs on numbered pages. No Table of Contents or Index.
Text, Form 1	Variation of Text with indented paragraphs and double spacing.
Thesis	Satisfies the format requirements for a thesis at the local institution.

E.3 Environments

These environments are available in all document types. Many document types have additional environments defined. For more information and explanation of each, see Section 3.3, page 15.

Name	*Result*
Abstract	Prints text on the title page. Available only for sectioned documents.
B	Requests **boldface** printing.
Black	Produces black colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
Blue	Produces blue colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
C	Requests SMALL CAPITAL printing.

[1] Consult with your DBA to see whether or not this document type is the default document type for your site.

Center	Centers each manuscript line in the body of the environment between the global margins.
Copyright	Produces a copyright statement on the title page. Available only for sectioned documents. Must be followed by a date and name.
CopyrightNotice	Produces a copyright notice on the title page. Available only for sectioned documents. Must be followed by the name of the copyright owner.
Corollary	Simple mathematical environment. Produces appropriately-numbered corollaries.
Cyan	Produces cyan colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
Dark	Produces dark colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
Definition	Simple mathematical environment. Produces appropriately-numbered definitions.
Description	Provides paragraphs with header words in a widened left margin. Use a tab command (@\) to separate the header word(s) from the rest of each paragraph.
Display	Displays each manuscript line that is inside the body of the environment. Output line breaks correspond to manuscript-file line breaks. Left margin widened.
Enumerate	Numbers each paragraph within the body. Sets list off from rest of text with spacing and wider margins.
Equation	Simple mathematical environment. Produces appropriately-numbered equations.
Example	For examples of computer input and output. Uses fixed-width typeface. Breaks lines as in manuscript file. Sets example off with spacing and wider margins.
F0 *through* **F9**	Special fonts. Provides a means for using fonts not defined for the current document. See the @SpecialFont command description.
Figure	Produces a figure with an appropriately-numbered caption.
FileExample	Provides environment for showing examples of computer file contents. Breaks lines as in manuscript file but does not truncate long lines.
Float	Allows the environment that it surrounds to be moved from its place in the .MSS file to a more convenient location in the output. Text continues to be placed and filled on the page even though the floated environment may be located elsewhere. Figures and Tables are floated.
FlushLeft	Aligns the first character in each manuscript line with the global left margin.
FlushRight	Aligns the last character in each manuscript line with the global right margin.

Format	For manual tabular formatting. Uses variable-width font. Breaks lines as in manuscript file. Sets body off with spacing. Does not adjust margins.
FullPageFigure	Produces a figure, with an appropriately-numbered caption, on its own page.
FullPageTable	Produces a table, with an appropriately-numbered caption, on its own page.
G	Requests Greek (ελλεν) printing.
Green	Produces green colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
Group	Delimits text in which page breaks are prohibited.
Heading	Places its body as an unnumbered heading. Breaks lines as in manuscript file.
I	Requests *italic* printing. Is converted to underlining on printing devices which cannot italicize.
InputExample	Used to print examples of computer input. Much like Example, but the text is indented more and long lines are wrapped (as they are in the Verse environment).
Itemize	Flags each paragraph in the margin with special character. Sets list off from the rest of text with spacing and wider margins.
Lemma	Simple mathematical environment. Produces appropriately-numbered lemmas.
Magenta	Produces magenta colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
MajorHeading	Places its body as a top-level unnumbered heading. Breaks lines as in manuscript file.
Math	Produces mathematical formulas in running text.
MathDisplay	Produces mathematical formulas in examples and displays.
Minus	Produces $_{\text{sub}}$script. Synonym: @-.
Multiple	Delimits multiple paragraphs in the .MSS file so that they are treated as a single paragraph by other environments like Enumerate and Description.
O	Requests that text be printed with an overbar. (Not available in most device types.)
OutputExample	Used for printing examples of computer output. Much like the Verbatim environment, but OutputExample is indented a bit more than Verbatim.
P	Requests ***bold italic*** printing.
Plus	Produces $^{\text{super}}$scripts. Synonym: @+.
Proof	Simple mathematical environment. Produces appropriately-numbered proofs.
ProgramExample	For examples of computer programs. Uses an appropriate font; breaks lines as in manuscript file.

Proposition	Simple mathematical environment. Produces appropriately-numbered propositions.
Quotation	Insets quotation as running text with wider margins and space above and below.
R	Requests ordinary roman type style. Intended for use inside @i or @b or @g where a few ordinary characters are needed.
Red	Produces red colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
SubHeading	Places its body as a subordinate heading. Breaks lines as in manuscript file.
T	Requests `typewriter` font.
Text	Provides plain, running text. All formatting is normally inside the environment Text unless specified otherwise.
Table	Produces a table with an appropriately-numbered caption.
Theorem	Simple mathematical environment. Produces appropriately-numbered theorems.
TitleBox	Prints text on the title page in the spot defined as the title box. Available only for sectioned documents.
TitlePage	Produces the title page. Other environments may be used inside this one to print text on the title page. Available only for sectioned documents.
Transparent	Has no attributes and does nothing different by itself. Used to alter the attributes of the current environment.
U	Requests underlined printing. All nonblank characters will be underlined.
UN	Like @u, but underlines only letters and digits.
UX	Like @u, but underlines all characters, including spaces.
Verbatim	Like Format, but uses a fixed-width font. Breaks lines as in manuscript file. Sets off body with spacing. Does not adjust margins.
Verse	Breaks lines as in manuscript file but does not truncate long lines. Sets off body with spacing and wider margins.
W	Treats its body as a "word", that is, as a sequence that cannot be broken across a line.
White	Produces white colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.
Yellow	Produces yellow colored output. The true color printed depends on your output device and the Scribe Database. Ask your ***DBA*** for further information regarding color at your site.

E.4 Commands

(See also the list of environments in Appendix E.3.)

In the following list, **boldface** is used to indicate parameters or command names that you must type exactly as they are listed here. *Italics* are used to indicate text, names, or parameters where you are free to use any value that you want — a variable in other words. ***Bold italics*** are used to indicate that your choice must be a name from a restricted set of names that are permitted in that context. {Braces} around something mean that it is optional. Parentheses are shown as the command delimiter, but any pair of Scribe delimiters may be used.

Some of the commands in this list have been designated as "advanced" to serve as a warning that you should not use the command unless you are fully conversant with the material in the *Scribe User Manual.* Misunderstanding of the intent or scope of a command can lead to hours of needless frustration. Any commands marked with a superscripted asterisk are really forms rather than commands. They are listed in here because their syntax is similar to command's syntax, and at the user-level, they may be thought of as commands.

Command	***Result***
@Bar()*	Draws a horizontal line from the left margin to the right margin.
@Begin(*Environment*, *attribute-value-list*)	Marks the beginning of the specified environment. A list of environments is in Appendix E.3. See also @End.
@Bibliography	Inserts the bibliography at this point in the document. Otherwise, Scribe automatically puts the bibliography at the end. If you use @Bibliography instead of having Scribe place your Bibliography for you, you must put a heading in front of it yourself and you may not get a complete Bibliography.
@BlankPage(*n*)	Inserts *n* pages into the document. The default value for *n* is 1. See also @NewPage.
@BlankSpace(*vertical-distance*)	Inserts blank space for a figure. See Appendix F.8 for a list of valid units of distance.
@Caption(*text-of-caption*)	Specifies the caption for a figure or a table. Any @Tag commands for the figure or table must come after the caption.

@Cite(*parameter{delimited-string}***)**

Generates a bibliographic citation to the reference entry identified by *parameter*, places that citation in the document in place of the @Cite command, and causes that bibliography entry to be included in the document's Bibliography. More than one *parameter* may be given, using commas as separators. If any *parameter* is followed by a delimited string, that string will be included in the citation.

@CiteMark(*parameter***)**

Causes the bibliography entry identified by *parameter* to be included in this document's Bibliography. No actual citation is placed in the text.

@Comment Produces no output. Provides a means to insert comments in the .MSS file without that text being printed in the output. Comment also masquerades as an environment in that it can be used in @Begin(comment)/@End(comment). It is the *only* command having this property.

@Define(*NewName{=OldName},list-of-attributes-and-values***)**

(Advanced command.) Defines *NewName* to be a Scribe environment, with the specified attributes and values. If the "*=OldName*" optional field is present, then the new name is the same as the old one except for the changes specified by the attribute and value list.

@DefineHyphenationDictionaries(*dictionary-name= delimited-word-list***)**

Defines a dictionary to be used with one of the dictionary hyphenation methods discussed in Chapter 10. Synonym: @DefineHyphenationDictionary.

@DefineHyphenationDictionary(*dictionary-name= delimited-word-list***)**

Defines a dictionary to be used with one of the dictionary hyphenation methods discussed in Chapter 10. Synonym: @DefineHyphenationDictionaries.

@Device(*DeviceName*)

Specifies the printing device for the output. See Table 2-1 on page 10 for a list of device names.

@End(*Environment*)

Marks the end of the specified environment. See also @Begin.

@Equate(*NewName=OldName*)

Specifies a synonym *NewName* for an existing Scribe command or environment named ***OldName***. "Equate" is not allowed as *Newname*.

@Foot(*text-of-footnote*)

Places the text in a footnote, numbers it, and inserts an appropriate footnote number in the text.

@GoTo(*horizontal-distance*)

(Advanced command.) Repositions the text cursor to be the given distance from the global left margin. If this new position is to the left of the current cursor position, a new line is started.

@Hinge Marks those positions in a grouped environment where Scribe is permitted to start a new page.

@Hpos(*horizontal-distance*)

(Advanced command.) Causes the text cursor to be repositioned at the given distance from the global left margin, even if this causes the current line to be overprinted. Should normally be used only in Database files.

@Hsp(*horizontal-distance*)

(Advanced command.) Causes a blank space of the requested width to be placed in the document file at that point. Intended for use only within Database files.

@Include(*name-of-a-file*)

Includes the contents of another file in your manuscript file at this point.

@Index(*text-to-be-indexed*)

Makes an entry in the Index, when the command appears in a document that allows indexing; otherwise the @Index command is ignored.

@IndexEntry(Key=<*sort-key*>,Entry=<*text-of-entry*>,{numbered <*number*>})*

(Advanced command.) Makes an entry in the Index that will be alphabetized under *sort-key*, but which will have entry text *text-of-entry*. If the optional Numbered parameter is present, then *number* will be used as the page reference number. @IndexEntry is only available when this command has been included in your document: @LibraryFile(MultiLevelIndex).

@IndexSecondary(Primary=<*primary-index-term*>, Secondary=<*secondary-index-term*>)*
Produces a two-level index entry, with the secondary term indented under the primary term. The secondary term is numbered, but the primary term is not. @IndexSecondary is only available when this command has been included in your document: @LibraryFile(MultiLevelIndex)

@Label(*codeword*) Defines *codeword* as a cross reference label representing the current place in the document. See @Ref and @PageRef.

@LibraryFile(*name-of-library-entry*)
(Advanced command.) Causes the Database to be searched for a library file, which is read in at the point of the command. (No message is printed at the user's terminal at this point.) Use @Include inside manuscript files; for the most part, @LibraryFile is intended for use within Database files only.

@LocalString(*codeword="value"*)]
Defines *Codeword* as a text string with the contents equal to the delimited string *"value"* for the duration of the current environment only. The definition disappears when the environment is exited. See @Value.

@Make(*document-type*)
Specifies the document type definition to use. A list of document types appears in Section E.7.

@Modify(*Name*, *list-of-attributes-and-values*)
(Advanced command.) Redefines or adds attributes to the environment or counter ***Name*** for the duration of the curren environment.

@NewColumn() Breaks the current column and starts at the top of a new one. In single-column text, this command is equivalent to @Newpage.

@NewPage(*n*) Breaks the current line and starts at the top of a new page (unless it is already at the top of a fresh page, in which case nothing further happens), if *n* is zero or not specified. Leaves *n* blank pages and starts at the top of a new page, if *n* is specified. @NewPage is immediate. It does not fill the previous page. See also @BlankPage.

@Note(*text-of-endnote*)

Like @Foot, but produces an endnote if the @Style(Notes Endnotes) command has been specified.

@Ovp(*text*) Outputs *text* to be overprinted and positions the formatting cursor at the beginning of the text.

@Pagefooting(Left=<*text*>, Center=<*text*>, Right=<*text*>, {Immediate,}{Even/Odd}, Line=<*Text line*>)

Specifies a footing to be put at the bottom of each page, beginning with the next page. Immediate specifies that the footing take effect on the current page. Odd or Even specifies the footing for odd- and even-numbered pages in a doublesided document. Line contains second (and subsequent) lines of a multiline footing.

@Pageheading(Left=<*text*>, Center=<*text*>, Right=<*text*>, {Immediate,}{Even/Odd}, Line=<*text-line*>)

Specifies a heading to be put at the top of each page, beginning with the next page. Pageheading permits the optional parameters Immediate, Odd/Even, and Line. See @Pagefooting.

@PageRef(*Codeword*)

Puts into the text the number of the page on which *Codeword* was defined (by @Label(*Codeword*) or @Tag(*Codeword*).)

@Part(*part-name*, Root=<*root-file-spec*>)

Indicates that a manuscript file is part of a multiple part document. This command must be the first command in a part file.

@Picture(Size=*vertical-distance*,GenericDevice=<*filespec*>)

Used inside **Figure**, **FullPageFigure**, and **Equation** to put a digitized picture into the document for device type *GenericDevice*.

@Place(*portion-name*)

(Advanced command.) Causes the text of the named portion that has been assembled so far to be placed in the document at this point.

@Ref(*Codeword*) Retrieves the value of the cross-reference marker *codeword* and places it in the document at that point. To define a cross-reference code word, see @Label and @Tag. See also @PageRef.

@SeeAlso(Primary=*primary-index-term*, Other=*reference-term*)*

Produces a two- or three-level index entry with the term listed as "Other" being prefaced with the phrase "See also". (The number of levels depends on whether of not an @IndexSecondary command has been included for the same primary term.) Neither terms receive a number. @SeeAlso is only available when this command has been included in your document: @LibraryFile(MultiLevelIndex).

@Set(*Counter*=*value*)

Sets the specified counter to the *value* or changes the counter by the designated value if value is signed: @Set(Page=+5) adds 5 to the page counter, but @Set[Page=7] sets it to 7.

@SpecialFont(*n*=<*file-spec*>)

Declares the *n*'th special font. Meaningful only for devices capable of changing fonts.

@String(*Codeword*="*value*")

Defines *Codeword* as a text string with the contents equal to the delimited string "*value*". See @Value.

@Style(*parameter*=*value*)

Sets the @Style parameter named ***parameter*** equal to *value*. @Style parameter names are listed in Appendix E.10.

@TabClear() Clears all tab stops.

@TabDivide(*n*) Sets tabs to divide the text body into *n* columns.

@TabSet(*tab-stop-positions*)

Sets a tab at the horizontal positions indicated. Multiple tabs may be specified in one @TabSet command, with *tab-stop-positions* separated by commas. Distances are computed with respect to the prevailing left margin. Existing tabs are not erased. When the stop value is signed (for example, + 1inch), the new stop is set relative to the preceding stop in the list, (or to the left margin in the case where the first value in the list is signed).

@Tag(*Codeword*) Defines *Codeword* as a cross-reference label representing the position and number of an equation, theorem, figure, or table. For use with @Ref.

@Title(*counter-name*)

Inserts into the text the title currently associated with *counter-name*, which should be a name like Chapter or Section.

@Use(*Component*=<*file-spec*>)

Tells Scribe to look in <*file-spec*> for the desired component. Component names are AuxFile, Bibliography, Database, and HyphenDic. The *file-spec* parameter for Database is actually a directory specification and not a file specification. More than one @Use specification is permitted for Bibliographies consisting of multiple files.

@Value(*name*) Inserts the value currently associated with the string *name*. Names are defined with the @String command; some strings are predefined by Scribe. These predefined strings appear in Appendix E.6.

E.5 Punctuation-Character Commands

A "punctuation-character" command is one that consists of an @-sign followed by a single punctuation character. Except for @+ and @-, these commands are complete in themselves and take no arguments.

Char	*Result*
@@	The command character followed by itself produces a single, literal "@" character in the document.
@␣	The command character followed by a space requests a literal space. That is, it treats the space as a character (part of a word) rather than as a word separator.
@!	Sets the return marker to the current horizontal position.
@$	Sets the left margin for the current environment to the current horizontal position.
@*	Forces Scribe to start a new line without justifying the old one.
@+(*text*)	Prints the *text* as a superscript at the current cursor position.
@-(*text*)	Prints the *text* as a subscript at the current cursor position.
@.	Generates a period that does not ever serve as the end of a sentence. For ending abbreviations. If @Style(Dotmode=Old), then @. sets a tab stop in the current cursor position instead.
@:	Forces a sentence break, even if the previous punctuation character was not a period or exclamation point or question mark.
@/	Moves the cursor to the return marker position. (See @!)
@=	Marks the left end of text to be centered. Do not use in a filled environment. See also @\.
@>	Marks the left end of text to be flushed right. Do not use in a filled environment. See also @\.

@\	Tab command. Moves the cursor to the next tab stop or marks the end of text being centered or flushed right. Do not use in a filled environment.
@|	Specifies a position within a word where a line break is permitted.
@^	Sets a tab at the current cursor position.
@&	Repeats the characters between & and the next command from the current position until the next tab setting.
@)	Like @&, but the replicated patterns are synchronized in fixed columns from one line to the next.
@;	No-operation. Scribe completely ignores @; in the input file.
@~	Causes Scribe to ignore everything in the manuscript file between it and the next printing character. Used for putting non-significant line breaks in environments where end-of-line normally matters.
@#	Leaves a quad space for a special character and creates an entry in the Error file so you can return to the proper place and draw in the character.
@]	Positions the cursor at the prevailing left margin.
@_	Conditional hyphen. Allows Scribe to break a word across a line boundary at the indicated point. If the word is broken, a hyphen will be inserted. If the word is not broken, no hyphen will appear. This command functions only when Hyphenation is turned on. If Hyphenation is off, @_ is ignored.

E.6 Predefined String Names

These strings are predefined for use with the @Value command (see Section 4.2, page 32).

Name	*Result*
Date	Day, month, and year of the current date, for example, 15 July 1985. The format is controlled by @Style(Date).
Day	Day of the month, for example, 15.
Device	The output device for this run. Taken from the @Device command in the .MSS file or the DeviceName parameter of the .DEV file, the "Device" command-line option, or supplied by default from the Site file.
DeviceName	The name of the output device for this run, for example, PostScript Page Description Language. Taken from the DeviceTitle parameter of the .DEV file.
FileDate	The date and time when the manuscript file was created, for example, 30 May 1985 at 14:24. The format is controlled by @Style(FileDate).
FullManuscript	The full file specification of the manuscript (root) file being processed.
GenericDevice	Name of the device class to which the selected output device belongs.

Manuscript	The name of the manuscript (root) file being processed, for example, USER1.MSS.
Month	The name of the current month, for example, July.
Page	The current page number in the document, for example, 217.
RootFileDate	If multiple files are in use (via @Include), RootFileDate is the date and time of last update of the root file, for example, 19 June 1985 at 12:02. The format is controlled by @Style(FileDate).
ScribeVersion	The version of Scribe currently processing the file, for example, 4(1405).
SectionNumber	The section number from the last sectioning command (null in an unnumbered document).
SectionTitle	The section title specified by the last sectioning command (null in an unsectioned document).
Site	The abbreviation for the site name. Taken from the site code field in SCRIBE.SIT.
SiteName	The name of the site. Taken from the site name field in SCRIBE.SIT.
SourceFile	The name and line number in the manuscript (included) file currently being processed, for example, VALUE.RFT, 09600/1.
Time	The time when the current Scribe run began, for example, 15:07. The format is controlled by @Style(Time).
Timestamp	The date and time when the current Scribe run began, for example, 15 July 85 15:07. The format is controlled by @Style(TimeStamp).
Username	The name of the user running the program, according to data provided by the operating system.
Weekday	The name of the current day of the week, for example, Monday.
Year	The current year, for example, 1985.

E.7 @Style Parameters

The @Style command specifies parameter-value pairs that control the appearance of the document. @Style parameters that affect the overall document definition are restricted to the beginning of the file (that is, prior to any output text). Other @Style parameters can appear anywhere in the manuscript and take effect when they are processed. The @Style command has the following form:

@Style(*parameter*$_1$ *value*$_1$, *parameter*$_2$ *value*$_2$, ...)

Some @Style parameters expect numeric values, for example 1.3 inches. Others expect word values, e.g. Yes or No. Others expect delimited string values, for example <8 March 1952>. Do not use delimiters on word or numeric values; that is, don't put quotes around Yes.

Parameter	*Values*
BackgroundColor	A color name specifying the color of the background on which this document is to be printed.

BibSelect Keyword {Cited or Complete} specifies if the document's bibliography is to contain all references from the .BIB file or just those that were cited with @Cite. Default value: Cited.

BibSequence A parameter from the set {Alphabetic, Numeric} specifying the sort sequence to be used for sorting the bibliography. If Numeric, the bibliography will appear in the order in which entries were cited in the text. If Alphabetic, the sort will be by Key field and year. Default value: Alphabetic.

BindingMargin Horizontal distance for binding doublesided documents. Its value should be the amount of paper that is expected to be covered by the staple or binding.

BottomMargin Vertical distance between last line of text and bottom of page. (Beginning only)

Citation An integer from 1 to 5 that specifies the citation format to be used.

1 = numeric citation, the first work cited is numbered 1, etc.

2 = the key field and the last two digits of the year, separated by a space.

3 = first *CitationLength* letters of the key, followed immediately by the last two digits of the year.

4 = your own key is used for citation.

5 = full key field and full year, separated by a comma.

CitationLength Used when @Style(Citations 3) (portion of author's last name plus last two digits of year) is in effect. This parameter specifies how many letters of the author's name are to be used. Default value: 3.

Citations Synonym for Citation.

CitationSeparator A string to be used to separate multiple citations. Default value: Comma.

CitationType A parameter from the set {Brackets, Parentheses, Plain, Superscripts} specifying how citations are to appear in the text. Default value: Brackets.

Color A color name specifying the color of the text in this document.

ColumnMargin Specifies the distance between columns of text.

Columns Specifies the number of columns that the text of the document is to be printed in.

Date A template that specifies the style for printing dates. This template must be some representation of the date Saturday, March 8, 1952. Month names may be in English, Spanish, French, or German; numbers may be ordinal, cardinal, roman, or English. For example:

```
@Style(Date=''8 March 1952'')
@Style(Date=''08/03/52'')
@Style(Date=''8 de marzo de 1952'').
@Style(Date=''Eighth of March, Fifty-two'')
```

DeviceName A string that overrides the value of the predefined string Device without actually changing the device for which output is being produced.

DeviceTitle A string that overrides the value of the predefined string DeviceName.

DotMode Controls the interpretation of the @. command. In old versions of Scribe, @. was used to set a tab stop. It is now used to indicate a period that does not end a sentence. Takes a value from the set {New, Old}. Default value: New.

DoubleSided Boolean value. Yes allows the insertion of extra blank pages to force major headings onto odd pages if this effect is specified in the document type.

Endnotes Boolean value. Yes means to place footnotes generated by @Foot at the end of the document. (Beginning only)

ExceptionDictionaries Specifies the dictionaries to be used to supplement the hyphenation algorithm. Synonym: ExceptionDictionary.

ExceptionDictionary Specifies the dictionary to be used to supplement the hyphenation algorithm. Synonym: ExceptionDictionaries.

FileDate A date template that specifies the style for printing file dates. See @Value(FileDate) and also Date, above.

FontFamily On devices that can change fonts, the name of the FontFamily to use for this document. (Beginning only)

FontScale The basic body type size in which this document will appear. Meaningful only for devices that can change type sizes automatically. (Beginning only)

Footnotes Counter Template for controlling style of footnote numbering. (Beginning only)

Hyphenation Specifies what method of hyphenation is to used for the document. Takes a value from the list {AutomaticExact, AutomaticFolded, DictionaryExact, DictionaryFolded, Off, False, No, On, Yes, True, Old, OldExact, OldFolded, Warn}.

HyphenationDictionaries Specifies the dictionaries to be used to supplement a dictionary hyphenation method. One or more dictionary names may be specified. Synonym: HyphenationDictionary.

HyphenationDictionary Specifies the dictionary to be used to supplement a dictionary hyphenation method. One or more dictionary names may be specified. Synonym: HyphenationDictionaries.

HyphenBreak A boolean that specifies whether a hyphen in the input text can be treated as a line break if necessary. This parameter is independent of the value of Hyphenation.

Indent Horizontal distance indicating amount of indenting for each paragraph, relative to its left margin. (Beginning only)

Indentation Same as Indent.

IndexCap Boolean that specifies whether all indexing terms are to be sorted as if they were capitalized.

Justification Boolean value. Yes means permit those environments that normally justify their right margins to do so. No means never justify a right margin. (Beginning only)

LeftMargin Horizontal distance of the global left margin from the physical left margin of the page. (Beginning only)

LineWidth Horizontal distance from the global left margin to the end of the line. (Beginning only)

LongestHyphenatable
Specifies the length of the longest word you want Scribe to hyphenate. Default value: 99.

MultipleBibliography
A boolean specifying whether more than one bibliography file needs to be processed.

Notes Specifies where to place footnotes. Takes a value from the set {Footnote, Endnote, Inline}. (Beginning only)

Outline A boolean that determines whether or not an outline will be produced for a sectioned document. Default value: True.

PageNumber A counter template specifying the style of page numbering.

PageNumbers Synonym for PageNumber.

RawFontDirectory
Specifies which RawFontDirectory is to be used for this document. Only certain devices use this parameter. Check with your *DBA* to see if you need to know more about it.

ReferenceFormat Synonym for References.

References Name of entry in the bibliography data base specifying which reference style and citation style to use. For example,

```
@Style(References=CACM)
```

RightMargin Horizontal distance between the end of the text line and the global right margin. (Beginning only)

ScriptPush Boolean value. Determines whether (Yes) or not (No) to add extra vertical spacing for subscripts and superscripts.

ShortestHyphenatable
Specifies the length of the shortest word you want Scribe to hyphenate. Default value: 5.

SingleSided No value. Turns off DoubleSided.

Spaces Sets the value of the Spaces attribute for the document. Takes a value from the set {Kept, Compact, Ignore, Tab, Null, Ignored, Normalize, Normalized, NoBreak}.

Spacing Vertical distance from base of one line of text to base of the next. (Beginning only)

Spread Vertical distance added to Spacing to specify the vertical spacing between paragraphs. (Beginning only)

StringMax Maximum number of characters that can appear in a delimited string. Default value: about 2000 characters.

Time A template specifying the format in which @Value(Time) will be printed. Similar to Date above; you must specify the time 4:30 p.m.:

```
@Style<Time=(1630hrs)>
```

TimeStamp Like Time, but it specifies the format in which @Value(TimeStamp) is printed.

TopMargin Vertical distance from top of paper to the baseline of the first text line on the page. (Beginning only)

WidestBlank A horizontal distance giving the maximum amount of white space that will be tolerated in a line before hyphenation will be attempted.

WidowAction Controls the processing of widow lines. Takes a value from the set {Force, Warn, ForceWarn, Ignore}.

E.8 Bibliography Formats

Bibliography format definitions in the Database are used to control the style and sequencing of the list of references and the citations. Select one with the **References** @Style parameter:

```
@Style(References=STDalphabetic)
```

As of the time of this manual (April 1984), the bibliography formats have not been made entirely consistent with one another. They were all written at different sites by different people, and not all of them implement all forms of all of the reference types. For definitive documentation, look at the Database files with file type .REF. Some of the .REF files call library files. For those files, look in the file FILENA.LIB, where FILENA is the first six characters of the parameter inside the @LibraryFile command.

Name	*Description*
1APA	Similiar to the APA format except that it contains an Annote field that is treated as a Comment.
1APADraft	Similiar to the 1APA format except that it is double-spaced.
AnnAPA	Similiar to the 1APA format except that the Annote field is treated as text.
AnnAPADraft	Similiar to the 1APADraft format except that the Annote field is treated as text.
AnnotedSTDAlphabetic	Same as StdAlphabetic, but includes annotations and has filled lines.

AnnotedSTDIdentifier
: Similiar to the STDIdentifier format except it includes annotations and has unfilled lines.

AnnotedSTDNumeric
: Same as STDNumeric, but includes annotations (i.e. the contents of the Annote field) in the Bibliography and has filled lines.

AnnSTDAlphabetic
: Similiar to the STDAlphabetic format except it includes annotations and has unfilled lines.

AnnSTDNumeric
: Similiar to the STDNumeric format except it includes annotations and has unfilled lines.

APA
: (American Psychological Association). Spelled-out citations (Knuth, 1978), outdented closed reference list, alphabetical ordering of references.

APADraft
: Draft version of APA format. Same as regular version, but triple-spaces the Bibliography.

CACM
: Numeric citations [5], closed format, alphabetical ordering of references.

ClosedAlphabetic
: Similiar to the STDAlphabetic format.

ClosedNumeric
: Similiar to the STDNumeric format.

IEEE
: Superscripted numeric citations[5], closed format, citation sequence ordering of references.

IPL
: (Information Processing Letters). The format required by IPL. This format is incomplete; it does not have all standard Scribe types yet (April 1984) and is being included for convenience only.

SIAM
: (Society for Industrial and Applied Mathematics). The format required by SIAM journals. This format is incomplete; it does not have all standard Scribe types yet (April 1984) and is being included for convenience only.

STDAlphabetic
: Alphabetic citations [Knuth 78], open format, alphabetical ordering of references.

STDIdentifier
: Open format, reference identifier for citations rather that a generated label.

STDNumeric
: Numeric citations [5], open format, alphabetical ordering of references.

E.9 Bibliography Entry Types

The entry types available in the standard Bibliography formats differ with each format. The optional and required fields for each Bibliography format are shown here. If two fields are flagged with the same superscript, then one or the other of them can be used, but not both.

E.9.1 The 1APA and AnnAPA Bibliography Reference Formats

The required and optional fields listed below are valid for the 1APA, 1APADraft, AnnAPA, and AnnAPADraft Bibliography reference formats.

Type	*Required Fields*	*Optional Fields*
Article	Journal Title Year*	Author Date* Key Month Note Number Pages Volume
Book	Publisher Title Date*	Address Author Edition Editor** Editors** HowPublished Key Note Series Volume Year*
Booklet	Title Date*	Address Author HowPublished Key Note Publisher Year*
Conference	Author Organization* Title Year**	Date** Meeting Month Society*

InBook	Author Date* Publisher Title	Address Booktitle Chapter Edition Editor** Editors** HowPublished Key Note Pages Series Volume Year*
InCollection	Author Booktitle Publisher Title Year	Address Editor* Editors* Key Note Series Volume
InProceedings	Author Booktitle Publisher* Title Year	Address Editor** Editors** Key Note Organization*
MastersThesis	Author School Title Year*	Date* Key Month Note
Manual	Title Year*	Author Address Date* Edition Month Note Organization
Misc	*None*	Author HowPublished Key Note Title
PhDThesis	Author School Title Year*	Date* Key Month Note

Proceedings	Organization** Title Year*	Address Date* Editor** Editors** Key Note Publisher
TechReport	Author Organization* Title Year**	Address Date** Institution* Key Month Note Number Type
Unpublished	Author	Date* Title Key Month Note Year*

E.9.2 The Annoted Bibliography Reference Formats

The required and optional fields listed below are valid for the AnnotedSTDAlphabetic, AnnotedSTDIdentifier, AnnotedSTDNumeric, AnnSTDAlphabetic, and AnnSTDNumeric Bibliography reference formats.

Type	*Required Fields*	*Optional Fields*
Article	Author Journal Title Year	Key Month Note Number Pages Volume
Book	Author Publisher Title Volume	Address Date* Note Series Year*
Booklet	Author Title	Address HowPublished Year Key Note

InBook	Author Booktitle Publisher Title Year*	Address Chapter Date* Editor** Editors** Key Note Pages Series Volume
InCollection	Author Booktitle Publisher Title Year	Address Chapter Editor Key Note Pages Series Volume
InProceedings	Author Booktitle Title Year	Address Editor Key Month Note Organization* Pages Publisher*
MastersThesis	Author School Title Year	Key Month Note
Manual	Title Year	Address Author Edition Key Note Organization
Misc	*None*	Author HowPublished Key Note Title
PhDThesis	Author School Title Year	Key Month Note
Proceedings	Organization* Publisher Title Year**	Address Date** Editor* Key Note

TechReport	Institution Title Year	Address Author Key Month Note Number Type
Unpublished	Author Title	Key Note

E.9.3 The APA Bibliography Reference Formats

The required and optional fields listed below are valid for the APA and APADraft Bibliography reference formats.

Type	*Required Fields*	*Optional Fields*
Article	Author Journal Pages Title Year*	Date* Key Month Note Number Volume
Book	Author Publisher Date* Title	Address HowPublished Key Note Series Volume Year*
Booklet	Author Publisher Title Year	Address HowPublished Key Note
Conference	Author Title Year*	Date* Meeting Month Note Society
InBook	Author Booktitle Publisher Title Year	Address Editor* Editors* Key Note Series Volume
InProceedings	Author Organization Title Year	Address Key Month Note Pages

MastersThesis	Author School Title Year	Key Month Note
Misc	*None*	Author HowPublished Key Note Title
PhDThesis	Author School Title Year	Key Month Note
Proceedings	Organization Title Year*	Address Date* Key Note
TechReport	Author Institution Title Year*	Date* Key Month Note Number Type
Unpublished	Author Title	Key Note

E.9.4 The CACM Bibliography Reference Format

The required and optional fields listed below are valid for the CACM Bibliography reference format.

Type	*Required Fields*	*Optional Fields*
Article	Author Journal Title Volume Year	Key Month Note Number Pages
Book	Publisher Title Editor* Year	Address Author* HowPublished Key Note Series Volume
Booklet	Author Title Year	Address HowPublished Key Month Note

InBook	Author Booktitle Publisher Title Year	Address Chapter Editor Key Note Pages Series Volume
InCollection	Author Booktitle Publisher Title Year	Address Chapter Editor* Editors* Key Note Pages Series Volume
InProceedings	Author Booktitle Title Year	Address Key Month Note Organization Pages
MastersThesis	Author School Title Year	Address Key Month Note
Manual	Title Year	Address Author Edition Key Note Organization
Misc	Author HowPublished Title	Key Note
PhDThesis	Author Key School Title Year	Month Note
Proceedings	Title Year	Address Key Month Note Organization

TechReport	Author Institution Title Year	Address Key Month Note Number Type
Unpublished	Author Note Title	Key

E.9.5 The Closed and STD Bibliography Reference Formats

The required and optional fields listed below are valid for the ClosedAlphabetic, ClosedNumeric, STDAlphabetic, STDIdentifier, and STDNumeric Bibliography reference formats.

Type	*Required Fields*	*Optional Fields*
Article	Author Journal Title Year	Key Month Note Number Pages Volume
Book	Author* Publisher Title Year	Address Editor* Editors* Key Note Number Series Volume
Booklet	Title	Address Author HowPublished Key Note Year
InBook	Author BookTitle Publisher Title Year	Address Chapter Key Note Number Pages Series Volume

InCollection	Author Booktitle Publisher Title Year	Address Chapter Editor* Editors* Key Note Number Pages Series Volume
InProceedings	Author Booktitle Title Year	Address Editor** Editors** Key Month Note Organization* Pages Publisher*
MastersThesis	Author School Title Year	Key Month Note
Manual	Title Year	Address Author Edition Key Note Organization
Misc	*None*	Author HowPublished Key Month Note Title Year
PhDThesis	Author School Title Year	Key Month Note
Proceedings	Organization* Publisher Title Year	Address Editor* Editors* Key Note

TechReport	Author Institution Title Year	Address Key Month Note Number Type
Unpublished	Author Title	Key Month Note Year

E.9.6 The IEEE Bibliography Reference Format

The required and optional fields listed below are valid for the IEEE Bibliography reference format.

Type	*Required Fields*	*Optional Fields*
Article	Author Journal Title Year	Key Month Note Number Pages Volume
Book	Author* Publisher Title Year	Address Editor* Editors* Key Note Number Series Volume
Booklet	Address Author Title Year	HowPublished Key Note
InBook	Author Publisher Title Year	Address Chapter Key Note Number Pages Series Volume

InCollection	Author Booktitle Publisher Title Year	Address Chapter Editor* Editors* Key Note Number Pages Series Volume
InProceedings	Author Booktitle Title Year	Address Editor** Editors** Key Month Note Organization* Pages Publisher*
MastersThesis	Author School Title Year	Key Month Note
Manual	Title Year	Address Author Edition FullOrganization Key Note Organization
Misc	*None*	Author HowPublished Key Note Title
PhDThesis	Author School Title Year	Key Month Note
Proceedings	Booktitle Year	Address Key Month Note Organization Publisher
TechReport	Author Institution Title Year	Key Month Note Number Type

Unpublished	Author Title	Key Note

E.9.7 The IPL Bibliography Reference Format

The required and optional fields listed below are valid for the IPL Bibliography reference format.

Type	*Required Fields*	*Optional Fields*
Article	Author* Journal Pages Title Volume Year	FullAuthor* Key Month Note Number
Book	Author* Publisher Title Year	Address FullAuthor* HowPublished Key Note Series Volume
Booklet	Author* Publisher Title Year	Address FullAuthor* HowPublished Key Month Note
InBook	Author* Booktitle Editor Pages Publisher Title Year	Address FullAuthor* Key Note Series Volume
InProceedings	Author* Booktitle Title Year	Address FullAuthor* Key Month Note Organization Pages
MastersThesis	Author* School Title Year	Address FullAuthor* Key Month Note
Misc	Author* HowPublished Title	FullAuthor* Key Note

PhDThesis	Author*	Address
	School	FullAuthor*
	Title	Key
	Year	Month
		Note
Proceedings	Organization	Address
	Title	Key
	Year	Note
		Pages
TechReport	Author*	Address
	Institution	FullAuthor*
	Number	Key
	Title	Month
	Year	Note
		Type
Unpublished	Author*	FullAuthor*
		Key
		Note
		Title

E.9.8 The SIAM Bibliography Reference Format

The required and optional fields listed below are valid for the SIAM Bibliography reference format.

Type	*Required Fields*	*Optional Fields*
Article	Author*	FullAuthor*
	Journal	Key
	Pages	Note
	Title	
	Volume	
	Year	
Book	Address	FullAuthor*
	Author*	HowPublished
	Publisher	Key
	Title	Note
	Volume	Series
	Year	
Booklet	Author*	Address
	Publisher	FullAuthor*
	Title	HowPublished
	Year	Key
		Month
		Note

InCollection	Address	FullAuthor*
	Author*	Key
	Booktitle	Note
	Editor	Series
	Pages	Volume
	Publisher	
	Title	
	Year	
InProceedings	Address	FullAuthor*
	Author*	Key
	Booktitle	Month
	Organization	Note
	Pages	
	Title	
	Year	
MastersThesis	Address	FullAuthor*
	Author*	Key
	School	Month
	Title	Note
	Year	
Misc	Author*	FullAuthor*
	HowPublished	Key
	Title	Note
PhDThesis	Address	FullAuthor*
	Author*	Key
	School	Month
	Title	Note
	Year	
Proceedings	Address	Key
	Booktitle	Note
	Organization	
	Year	
TechReport	Address	FullAuthor*
	Author*	Key
	Institution	Month
	Number	Note
	Title	Type
	Year	
Unpublished	Author*	FullAuthor*
		Key
		Note
		Title

E.10 Bibliography Field Parameters

These parameters are used in defining bibliography database entries. All take a delimited string or an abbreviation code as an value.

Name	*Description*
Address	The address of the publisher or printer or organization.
Author	The name(s) of the author or authors, in the format that they should be printed.
Annote	Any annotation text. Not actually printed in most bibliography formats.
BookTitle	The title of a book or proceedings of which this reference is a chapter or paper or article. Do not italicize or underline.
Chapter	If a reference is being made to part of a book and not the entire book, you can specify either chapter or pages.
Edition	Manuals often have an edition name or number that is not part of the actual title of the manual.
Editor	The name of the editor. If more than one, use Editors.
Editors	The name of the editors. If only one, use Editor.
FullAuthor	The full name of the author or authors, written out without commas, as "John Q. Citizen."
HowPublished	For unusual manuscripts, how it came into your possession ("personal note", etc.).
Institution	The organization or institution backing or publishing a technical report or a proceedings.
Journal	The title of the journal. Do not italicize or underline.
Key	The sort key. This field is used for alphabetization.
Month	January, February, etc.
Note	Any comment. Differs from Annote in that Note will always be printed, but Annote will be printed only in those bibliography types that specify annotation.
Number	Issue number of a journal or series number in a book series or serial number of a technical report.
Organization	The name of the organization holding a conference that published a proceedings.
Pages	The page numbers within a journal, proceedings, or book that contain the material actually cited.
Publisher	The name of the publishing company.
School	For theses, the name of the school granting the degree.
Series	When books are published in a series, the series has a name.
Title	The title of the book, article, thesis, or other document that is being cited. Do not italicize or underline; that detail will be handled by the reference format you select.
Type	Some technical reports are called by other names. If this is not a "Technical report" then put its true name in this field.

Volume	The volume number of a journal or a series book. Do not italicize or boldface.
Year	The year of publication; four digits: 1979.

Appendix F
Syntax Summary

By *syntax* we mean the rules for what characters are allowed in what positions. Can a cross-reference name contain a hyphen? Can you put spaces after the commas in a @PageHeading command? This Appendix covers the general rules that apply to all the commands and environments in Scribe.

Capitalizing

Capitalizing only matters in the numbering templates listed in Figure 15-2. For every other Scribe term, command, environment, or attribute, it does not matter. For example, @BEGIN is as good as @begin or @BeGiN. We usually use mixed case because we find it easier to read. As you can see from the examples in the text, anything goes.

Delimiters

Scribe recognizes seven pairs of delimiters. Any of the pairs is valid anytime. The only restriction is that you use the closing delimiter that matches the opening one.

```
{ }    [ ]    < >    ( )    " "    ‘ ’    ’’
```

F.1 Environments and Delimiters

Environments can appear in either a long form (with @Begin/@End) or a short form (with just the name). For environments, @Begin and @End used with the environment name act themselves as delimiters. For example, **@Begin(Description)** would be equivalent to **@Description[** (or to @Description with any opening delimiter); **@End(Description)** would be equivalent to] (or to whichever closing delimiter matches the opening delimiter).

F.2 Commands and Delimiters

For commands, you can never use @Begin and @End as delimiters; that is, you must always use the short form with one of the delimiter pairs. For example

```
@PageHeading<left "@Value(Page)",
     right "@value(Sitename)">
```

Another way of saying this is that commands have only a short form.

F.3 Spaces

Where can a space go in a Scribe command or environment? A command starts with an @ character. The command or environment name must immediately follow the @ sign. (This restriction is necessary because @ followed by a space is actually a command for a significant blank.) You can put a space between a command or environment name and its opening delimiter, but that practice is very poor stylistically. For example,

```
@Begin [Display]
@Heading  [How I Spent My Summer Vacation]
```

F.4 Carriage Returns

Whether a carriage return is part of a command or part of the manuscript text depends on where the @ sign for the command is. When the @ sign is the first character on a line, then the end-of-line following it is part of the command, not part of the text. When the @ sign is anywhere else on the line, then an end-of-line following it is part of the manuscript text.

F.5 Code Names for the @Tag and @Label Commands

Code names that you define with @Tag and @Label can be as long or short as you want. They can contain any number or letter and any of the following special characters.

#	pound sign
.	period
-	hyphen
&	ampersand
%	percent sign

The following special characters are not allowed in code names because they have special meanings in Scribe:

/	slash
,	comma
;	semicolon
_	underscore
=	equal sign
	any of the delimiter characters

F.6 String and Environment Names

Names for strings and environments (defined with @String and @Define) can be any length you want. They too can contain any number or letter, but the set of special characters not permitted in strings is more restricted than for labels. Only three special characters are permitted:

#	pound sign
&	ampersand
%	percent sign

F.7 Parameter Commands

Some commands, such as @Style and @Pageheading, contain a list of parameters and values. Sometimes the values are delimited strings. It is good practice to always put delimiters around parameter values, but as the general rule is that if the value contains a Scribe delimiter, spaces, or a comma, it should be delimited.

```
@Style(parameter1=value1,parameter2 "value2",
                    parameter3 [value3])
```

As you can see, you need some way of separating a parameter from its value. Three characters are valid separators:

=	equal sign
/	slash
	a space

Parameters and values come in pairs. You need to separate pairs from each other with commas. Both the separators and the commas can have any number of spaces surrounding them. That is, the following command means exactly the same as the one above.

```
@Style(
        Parameter1      value1,
        Parameter2   =    "value2"
        ,Parameter3=    [Value3]
)
```

Delimited values cannot contain any instances of the closing delimiter, even if it is paired with its opening delimiter. For example, the characters in "value2" above could not contain a double-quote, and the characters in "value3" could not contain a closing square bracket `]`.

F.8 Numbers and Distances

Numbers representing distances can be carried to 4 significant figures. For example, @Style(Spacing=0.884cm) actually attempts to set the final printing device to the requested line spacing of 0.884 centimeters. Of course, if the printing device is not capable of the resolution you request, Scribe will give you the positioning closest to the requested value.

Numbers whose meaning depends on the size of the current font (values for lines, characters, spacing, and so on) are carried only to the nearest tenth of a line or tenth of a character, meaning that @Style(Spacing=1.634) is treated as @Style(Spacing=1.6).

In specifying distance, you can use any of the following units. All units in the same row are equivalent.

Table of Distance Units

in, inch, inches, "
cm, centimeters
mm, millimeters
pt, pts, point, points
pica, picas
em, ems, quad, quads
char, chars, character, characters, en, ens
line, lines

Appendix G
Mathematical Formulas

A table of special characters is included in this Appendix for the Dover Laser Printer (Table G-1), the Quality Micro Systems Lasergrafix 1200 using Talaris fonts, the Santec S700 Terminal (Table G-3), and the Imagen Imprint-10 Laser Printer (Table G-4). The table for special characters available on the Apple LaserWriter (POSTSCRIPT). is included in Chapter 11 (Table 11-2). If you desire mathematical output on other printers, contact your ***DBA***.

Several examples of both the manuscript form and document result for formulas produced using the Scribe mathematical facility, which is described in detail in Chapter 11, are shown in this Appendix. The examples illustrate both the syntax of the facilities individual commands and some of the possible combinations of them. All the examples except the footnote example were produced using the MathDisplay environment. That one example was produced with the Math environment.

The input (manuscript form) for many of the multi-line mathematical forms do not show delimiters, even though they are documented as taking delimited strings as values in Section . The rule for when the mathematical forms need delimiters is the same as the rule for when parameter commands take delimiters. (See Section F.7 for a discussion of delimiters and parameter commands.) If the value to the mathematical form contains a Scribe delimiter, a space, or a comma, then delimiters are required. If none of those characters are included as part of the value, then the delimiters may be removed. We do, however, suggest that you always use delimiters to avoid problems.

Examples in the MathDisplay Environment

Manuscript Form:

```
@Int(From a, To b)f#(x)dx#=#@~
@Limit(As "@Delta x@~
@RightArrow 0")@Sum(From i=1,
To n) f#(x@down[i]) @Delta x
```

Document Result:

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{i=1}^{n} f(x_i)\Delta x$$

```
@Sr(erf)(x)#=#@Over(Num 2,
Denom "@Sqrt(@g(p))")@~
@Int(From 0, To x)@~
e@up[-t@up[2]] dt
```

$$\operatorname{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$$

```
@Limit(As n @RightArrow @Infty)@~
@g(f)@down[n](x)#=#@~
@Brace(Bot "1, @Quad x #=#1.",
Top "0, @Quad 0 @LtE x#<#1,")
```

$$\lim_{n\to\infty}\phi_n(x)=\begin{cases}0, & 0\le x<1,\\ 1, & x=1.\end{cases}$$

```
H(p,r) @Eqv @Sum(From i=1, To N)@~
@Over(Num p@down[i]@up[2],
Denom 2m)#+#@Omega(R@down[1],@~
#@LDots#,R@down[N])
```

$$H(p,r)\equiv\sum_{i=1}^{N}\frac{p_i^2}{2m}+\Omega(R_1,\ \ldots\ ,R_N)$$

```
(@Delta S)@down[@r[gas]]#=#@~
@Int() @Over(Num dQ,Denom T)#=#@~
@Over(Num "@Delta Q",Denom T)#=#@~
R#@Log@Over(Num V@down[2],
Denom V@down[1])
```

$$(\Delta S)_{\text{gas}}=\int\frac{dQ}{T}=\frac{\Delta Q}{T}=R\log\frac{V_2}{V_1}$$

```
y@down[1](x)#=#@Sum(From n=0,
To @Infty)@Over(Num "(-1)@~
@up[n]x@up[n]",
Denom "n!(n#+#1)#!#")
```

$$y_1(x)=\sum_{n=0}^{\infty}\frac{(-1)^n x^n}{n!(n+1)!}$$

```
@Sum(From "0 @LtE k @LtE n")@~
@Choose(Chosen m, From k)#=#@~
@Choose(Chosen m, From 0)#+#
@Choose(Chosen m,From 1)#+#@~
@CDots#+#@Choose(Chosen m,
From n)#+#@Choose(Chosen m#+#1,
From n#+1)
```

$$\sum_{0\le k\le n}\binom{k}{m}=\binom{0}{m}+\binom{1}{m}+\cdots+\binom{n}{m}+\binom{n+1}{m+1}$$

```
@Sin t#=#@Sum(From n=0,
To @Infty) @Over(Num "(-1)@~
@up[n]t@up[2n+1]",
Denom "(2n#+#1)!")
```

$$\sin t=\sum_{n=0}^{\infty}\frac{(-1)^n t^{2n+1}}{(2n+1)!}$$

```
@r(See footnote.)@Foot<The
difference between
@Math[@Int(From a, To x)@~
e@up[-t@up[2]]dt]@~
and @Math[@Int(From b, To x)@~
e@up[-t@up[2]]dt@~
is merely a constant which
can be added to @Math[C],
making the choice of the lower
limit of integration actually
immaterial.
```

See footnote.[1]

```
@Sr(if)@0#<#@Abs(x#-#b)#<#@~
@g(d),@@Sr(then)@@~
@Abs[f#(x)#-#L]#<#@g(e).
```

if $0<|x-b|<\delta$, then $|f(x)-L|<\varepsilon$.

```
@Sum(From k=1, To n)@g(g)@~
@down[2k-1] @RightArrow@~
@Over(Num n, Denom "2@g(p)")@~
#@Int(From 0, To "2@g(p)")@~
d@g(n)#@g(g)(@g(n))
```

$$\sum_{k=1}^{n}\gamma_{2k-1}\rightarrow\frac{n}{2\pi}\int_{0}^{2\pi}d\nu\,\gamma(\nu)$$

```
@Sum(From i=1, To 3)@~
@g(L)@down[ii]#=#m@~
@Sum(From i=1, To 3)@~
@Over(Num "@Partial u@~
@down[i]",
Denom "@Partial x@~
@down[i]")#=#m@~
@Nabla:@b(u)
```

$$\sum_{i=1}^{3}\Lambda_{ii}=m\sum_{i=1}^{3}\frac{\partial u_i}{\partial x_i}=m\nabla\cdot\mathbf{u}$$

```
f#=#{(x,y)@VBar#y#=#@~
(x@up[2]#+#x#-#6)/(x@up[2]@~
#+#4),#x@In@r(Re)}
```

$$f=\{(x,y)|y=(x^2+x-6)/(x^2+4),x\in\mathrm{Re}\}$$

---

[1] The difference between $\int_a^x e^{-t^2}dt$ and $\int_b^x e^{-t^2}dt$ is merely a constant which can be added to C, making the choice of the lower limit of integration actually immaterial.

```
@g(f)(x@down[n+1])#-#@g(f)@~
(x@down[n-3])#=#@Int(From x@~
@down[n-3], To x@down[n+1])@~
@g(f)'(x)dx
```

$$\phi(x_{n+1})-\phi(x_{n-3})=\int_{x_{n-3}}^{x_{n+1}}\phi'(x)dx$$

```
F#(s)G(s)#=#@Int(From 0,
To @Infty) g(@g(h))d@g(h)@~
@Int(From 0, To @Infty)@~
e@up[-s(@g(x)+@g(h))]@~
f#(@g(x))d@g(x)
```

$$F(s)G(s)=\int_0^\infty g(\eta)d\eta\int_0^\infty e^{-s(\xi+\eta)}f(\xi)d\xi$$

```
J@down[1](x)#=#@Over(Num x,
Denom 2)# @Sum(From n=0,
To @Infty) @Over(Num "(-1)@~
@up[n]x@up[2n]",
Denom "(n#+#1)!n!2@up[2n]")
```

$$J_1(x)=\frac{x}{2}\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(n+1)!n!2^{2n}}$$

```
@Nabla@up[2]y@Ovp(')@~
@down[n]#=#@Nabla(@Nabla y@~
@Ovp(')@down[n])#=#@~
@Nabla(y@Ovp(')
@down[n]#-#@~
y@Ovp(')@down[n-1])
```

$$\nabla^2 y'_n=\nabla(\nabla y'_n)=\nabla(y'_n-y'_{n-1})$$

```
x@up[-1/2] @Sum(From m=0,
To @Infty) @Over(Num "(-1)@~
@up[m]x@up[2m+1]",
Denom "(2m#+#1)!"), @Quad x#>#0
```

$$x^{-1/2}\sum_{m=0}^{\infty}\frac{(-1)^m x^{2m+1}}{(2m+1)!},\quad x>0$$

```
z@g(l)#=#@Over(Num "@g(ra)@~
q@up[2]", Denom R@down[0]@up[2])@~
e@up[R@down[0]/@g(r)] #@~
@SimEq # 3.8 * 10@up[-8]@~
@Sr(erg)
```

$$z\lambda=\frac{\rho\alpha q^2}{{R_0}^2}e^{R_0/\rho}\approx 3.8\times10^{-8}\,\text{erg}$$

```
@Ln @Over(Num K@down[2],
Denom K@down[1])#=#2.303#@Log@~
@Over(Num K@down[2],
Denom K@down[1])#=#@~
-#@Over(Num "@Delta H@up[@i(0)]",
Denom RT) @VBar@~
@Ss(Super T@down[2],
Sub T@down[1])
```

$$\ln\frac{K_2}{K_1}=2.303\log\frac{K_2}{K_1}=-\frac{\Delta H^0}{RT}\Big|_{T_1}^{T_2}$$

```
@g(f)@down[n](x) #=#@~
x@up[2]#+#@Over(Num x@up[4],
Denom 2!)#+#@Over(Num x@up[6],
Denom 4!)#+#@CDots#+#@~
@Over(Num x@up[2n], Denom n!)
```

$$\phi_n(x)=x^2+\frac{x^4}{2!}+\frac{x^6}{4!}+\cdots+\frac{x^{2n}}{n!}$$

```
P#(x)#=#b@down[0]x@up[n]#+#@~
b@down[1]x@up[n-1]#+#@~
@CDots#+#b@down[n-3]@~
x@up[3]#+#b@down[n-2]@~
x@up[2]#+#b@down[n-1]x@~
#+#b@down[n]
```

$$P(x)=b_0x^n+b_1x^{n-1}+\cdots+$$
$$b_{n-3}x^3+b_{n-2}x^2+b_{n-1}x+b_n$$

```
@g(y)@down[y](x,y)@^#=#@~
@Over(Num "@Partial",
Denom "@Partial y")@~
@Int(From 0, To x) M(t,y)@~
dt#+#h′(y)
```

$$\psi_y(x,y)=\frac{\partial}{\partial y}\int_0^x M(t,y)dt+h'(y)$$

```
A#=#@Over(Num "(@g(e)/@g(s))@~
A@down[0]e@up[@g(e)t]",
Denom "A@down[0]e@up[@g(e)t]#+#@~
(@g(e)/@g(s)#-#A@down[0])")
```

$$A=\frac{(\varepsilon/\sigma)A_0e^{\varepsilon t}}{A_0e^{\varepsilon t}+(\varepsilon/\sigma-A_0)}$$

```
@Over(Num "@LAngle m@down[k]@~
@RAngle", Denom M)#=#@~
@Over(Num "e@up[-@g(b)E@down[k]]",
Denom "@Sum(From k=0, To @Infty)@~
e@up[-@g(b)E@down[k]]")
```

$$\frac{\langle m_k\rangle}{M}=\frac{e^{-\beta E_k}}{\sum_{k=0}^{\infty}e^{-\beta E_k}}$$

```
@g(Y)(x)#=#@Int(From -1, To 1)#@~
@Over(Num "@g(b)x@~
@up[@g(g)/@g(d)]",
Denom "(x+1)(x+2)")@~
e@up[-2@g(d)x] dx
```

$$\Psi(x)=\int_{-1}^{1}\frac{\beta x^{\gamma/\delta}}{(x+1)(x+2)}e^{-2\delta x}dx$$

```
@Int(From 0, To "@r(T)")@~
g[t,x(t),u(t)]dt#+#@g(e) @GtE
@Int(From 0, To "@r(T)")@~
g[t,x@Ast(t),u@Ast(t)]dt
```

$$\int_0^T g[t,x(t),u(t)]dt+\varepsilon\geq \int_0^T g[t,x^*(t),u^*(t)]dt$$

```
@g(y)@down[@r(m,M)]@~
(0,0)#=#@Inf<@g(f)@~
@down[@r(m,M)](p)@~
{@VBar p @In@~
@r(A)@down[n]@up[1]}
```

$$\psi_{m,M}(0,0)=\inf\{\phi_{m,M}(p)|p\in A_n^{\,1}\}$$

```
(u @OTimes v) @OTimes w #=#@~
((uv)(1#+#@g(d)@down[1]))@~
@OTimes w
```

$$(u\otimes v)\otimes w=((uv)(1+\delta_1))\otimes w$$

```
@g(Y)@down[0]'(@b(r)@down[1],@~
#@LDots#,@b(r)@down[N]) @Eqv@~
@Over(Num Z, Denom V@up[N/2])#@~
@Prod(From i<j) [1#+#f#(ij)]
```

$$\Psi_0'(\mathrm{r}_1,\ldots,\mathrm{r}_N)\equiv\frac{Z}{V^{N/2}}\prod_{i<j}[1+f(ij)]$$

```
@g(y)@down[@Infty](@b(r))#=#@~
@Sum(From l=0, To @Infty)@~
@Sum(From m=-l, To +l)@~
Y@down[lm](@g(q),@g(f))@~
@g(y)@down[lm](kr)
```

$$\psi_\infty(\mathrm{r})=\sum_{l=0}^{\infty}\sum_{m=-l}^{+l}Y_{lm}(\theta,\phi)\psi_{lm}(kr)$$

Table G-1: Special Characters for the Dover

Informal name	Scribe name	Example	Availability
Aleph	@Aleph	ℵ	Normal
And	*See "Intersection (logical)"*		
Angle	@Angle	∠	Normal
Approximate equality	@Approx	≈	Normal
Asterisk	@Ast	*	Normal
Back arrow	*See "Left arrow"*		
Bottom	@Bot	⊥	Normal
Bullet (hollow)	@Circ	∘	Normal
Bullet (solid)	@Bullet	•	Normal
C-set	@CSet	ℂ	Normal
Circle (small)	*See "Bullet (hollow)" and "Degrees"*		
Degrees	@Degr		Blank
Delta	@Delta	Δ	Normal
Divided-by	@Div	/	Normal
Dot-in-circle	@ODot		Blank
Down arrow	@DownArrow	↓	Normal
Empty set	@EmptySet	∅	Normal
Equality	@Eq	=	Normal
Equivalence	@Eqv	≡	Normal
Existential quantifier	@Exists	∃	Normal
For-all	*See "Universal quantifier"*		
Greater	@Gt	>	Normal
Greater-or-equal	@GtE	≥	Normal
Greater-or-less	@GtLt		Blank
H-bar	@HBar	ℏ	Normal
In	*See "Membership"*		
Inequality	@Neq	≠	Normal
Infinity	@Infty	∞	Normal
Integers	@ZSet	ℤ	Normal
Intersection (logical)	@And	∧	Normal
Intersection (set)	@Inter	∩	Normal
Intersection (square)	@SqInter	⊓	Normal
Left angle bracket	@LAngle	⟨	Normal
Left arrow	@LeftArrow	←	Normal
Less	@Lt	<	Normal
Less-or-equal	@LtE	≤	Normal
Less-or-greater	@LtGt		Blank
Membership	@In	∈	Normal
Minus	@Sub	−	Normal
Minus-in-circle	@Ominus	⊖	Normal
Minus-or-plus	@Mp	∓	Normal
Much-greater	@MuchGt	≫	Normal
Much-less	@MuchLt	≪	Normal
Nabla	@Nabla	∇	Normal

Natural numbers	@NSet	$\mathbb{N}$	Normal
Negation (logical)	@Not	$\neg$	Normal
Non-equivalence	@NEqv	$\not\equiv$	Substitute
Non-membership	@NotIn	$\notin$	Normal
Not	*See "Negation (logical)"*		
Not-in	*See "Non-membership"*		
Operator (generic)	@Op		Blank
Or	*See "Union (logical)"*		
Partial derivative	@Partial	∂	Normal
Planck's constant	*See "H-bar"*		
Plus	@Add	$+$	Normal
Plus-in-circle	@OPlus	$\oplus$	Normal
Plus-in-u	@UPlus	$\uplus$	Normal
Plus-or-minus	@Pm	$\pm$	Normal
Q.E.D.	@Qed	$\Box$	Normal
Q-set	@QSet	$\mathbb{Q}$	Normal
Right angle bracket	@RAngle	$\rangle$	Normal
Rational numbers	@RSet	$\mathbb{R}$	Normal
Right arrow	@RightArrow	$\rightarrow$	Normal
Similarity	@Similar	$\sim$	Normal
Similar or equal	@SimEq	$\approx$	Normal
Slash	*See "Divided-by"*		
Slash-in-circle	@ODiv	$\oslash$	Normal
Square	*See "Q.E.D."*		
Star	*See "Asterisk"*		
Subset	@Subset	$\subseteq$	Normal
Subset (proper)	@PrSubset	$\subset$	Normal
Superset	@Supset	$\supseteq$	Normal
Superset (proper)	@PrSupset	$\supset$	Normal
Times	@Mult	$\times$	Normal
Times-in-circle	@OTimes	$\otimes$	Normal
Top	@Top	$\top$	Normal
Triangle	*See "Nabla" and "Delta"*		
Union (logical)	@Or	$\vee$	Normal
Union (set)	@Union	$\cup$	Normal
Union (square)	@SqUnion	$\sqcup$	Normal
Universal quantifier	@ForAll	$\forall$	Normal
Up arrow	@UpArrow	$\uparrow$	Normal
Vertical bar	@VBar	$\mid$	Normal
Vertical bar (double)	@DVBar	$\parallel$	Normal

Table G-2: Special Characters for the QMS using Talaris Fonts

Informal name	Scribe name	Example	Availability
Aleph	@Aleph	ℵ	Normal
And	*See "Intersection (logical)"*		
Angle	@Angle	∠	Substitute
Approximate equality	@Approx	≈	Normal
Asterisk	@Ast	*	Normal
Back arrow	*See "Left arrow"*		
Bottom	@Bot	⊥	Normal
Bullet (hollow)	@Circ	∘	Normal
Bullet (solid)	@Bullet	•	Normal
C-set	@CSet	**C**	Fake
Circle (small)	*See "Bullet (hollow)" and "Degrees"*		
Degrees	@Degr	°	Normal
Delta	@Delta	Δ	Normal
Divided-by	@Div	/	Normal
Dot-in-circle	@ODot	⊙	Normal
Down arrow	@DownArrow	↓	Normal
Empty set	@EmptySet	∅	Normal
Equality	@Eq	=	Normal
Equivalence	@Eqv	≡	Normal
Existential quantifier	@Exists	∃	Normal
For-all	*See "Universal quantifier"*		
Greater	@Gt	>	Normal
Greater-or-equal	@GtE	≥	Normal
Greater-or-less	@GtLt		Blank
H-bar	@HBar		Blank
In	*See "Membership"*		
Inequality	@Neq	≠	Substitute
Infinity	@Infty	∞	Normal
Integers	@ZSet	**Z**	Fake
Intersection (logical)	@And	∧	Normal
Intersection (set)	@Inter	∩	Normal
Intersection (square)	@SqInter	⊓	Normal
Left angle bracket	@LAngle	⟨	Normal
Left arrow	@LeftArrow	←	Normal
Less	@Lt	<	Normal
Less-or-equal	@LtE	≤	Normal
Less-or-greater	@LtGt		Blank
Membership	@In	∈	Normal
Minus	@Sub	−	Normal
Minus-in-circle	@Ominus	⊖	Normal
Minus-or-plus	@Mp	∓	Normal
Much-greater	@MuchGt	≫	Normal
Much-less	@MuchLt	≪	Normal
Nabla	@Nabla	∇	Normal
Natural numbers	@NSet	**N**	Fake
Negation (logical)	@Not	¬	Normal
Non-equivalence	@NEqv	≢	Substitute
Non-membership	@NotIn	∉	Substitute
Not	*See "Negation (logical)"*		

Not-in	*See "Non-membership"*		
Operator (generic)	@Op	**OP**	Fake
Or	*See "Union (logical)"*		
Partial derivative	@Partial	∂	Normal
Planck's constant	*See "H-bar"*		
Plus	@Add	$+$	Normal
Plus-in-circle	@OPlus	$\oplus$	Normal
Plus-in-u	@UPlus	$\uplus$	Normal
Plus-or-minus	@Pm	$\pm$	Normal
Q.E.D.	@Qed		Blank
Q-set	@QSet	**Q**	Fake
Right angle bracket	@RAngle	$\rangle$	Normal
Rational numbers	@RSet	**R**	Fake
Right arrow	@RightArrow	$\rightarrow$	Normal
Similarity	@Similar	$\sim$	Normal
Similar or equal	@SimEq	$\simeq$	Normal
Slash	*See "Divided-by"*		
Slash-in-circle	@ODiv	$\oslash$	Normal
Square	*See "Q.E.D."*		
Star	*See "Asterisk"*		
Subset	@Subset	$\subseteq$	Normal
Subset (proper)	@PrSubset	$\subset$	Normal
Superset	@Supset	$\supseteq$	Normal
Superset (proper)	@PrSupset	$\supset$	Normal
Times	@Mult	$\times$	Normal
Times-in-circle	@OTimes	$\otimes$	Normal
Top	@Top	$\top$	Normal
Triangle	*See "Nabla" and "Delta"*		
Union (logical)	@Or	$\vee$	Normal
Union (set)	@Union	$\cup$	Normal
Union (square)	@SqUnion	$\sqcup$	Normal
Universal quantifier	@ForAll	$\forall$	Normal
Up arrow	@UpArrow	$\uparrow$	Normal
Vertical bar	@VBar	$\mid$	Normal
Vertical bar (double)	@DVBar	$\parallel$	Normal

Table G-3: Special Characters for the Santec

Informal name	Scribe name	Example	Availability
Aleph	@Aleph		Blank
And	*See ''Intersection (logical)''*		
Angle	@Angle	∠	Normal
Approximate equality	@Approx	≈	Normal
Asterisk	@Ast	*	Normal
Back arrow	*See ''Left arrow''*		
Bottom	@Bot	⊥	Normal
Bullet (hollow)	@Circ	∘	Normal
Bullet (solid)	@Bullet	·	Normal
C-set	@CSet	C	Fake
Circle (small)	*See ''Bullet (hollow)'' and ''Degrees''*		
Degrees	@Degr		Blank
Delta	@Delta	Δ	Normal
Divided-by	@Div	/	Normal
Dot-in-circle	@ODot		Blank
Down arrow	@DownArrow		Blank
Empty set	@EmptySet	Φ	Substitute
Equality	@Eq	=	Normal
Equivalence	@Eqv	≡	Normal
Existential quantifier	@Exists		Blank
For-all	*See ''Universal quantifier''*		
Greater	@Gt	>	Normal
Greater-or-equal	@GtE	⩾	Normal
Greater-or-less	@GtLt	≷	Normal
H-bar	@HBar	ℏ	Normal
In	*See ''Membership''*		
Inequality	@Neq	≠	Normal
Infinity	@Infty	∞	Normal
Integers	@ZSet	Z	Fake
Intersection (logical)	@And	∧	Substitute
Intersection (set)	@Inter	∩	Normal
Intersection (square)	@Sqinter		Blank
Left angle bracket	@LAngle	⟨	Normal
Left arrow	@LeftArrow	←	Normal
Less	@Lt	<	Normal
Less-or-equal	@LtE	⩽	Normal
Less-or-greater	@LtGt	≶	Normal
Membership	@In	∈	Substitute
Minus	@Sub	−	Normal
Minus-in-circle	@Ominus		Blank
Minus-or-plus	@Mp	∓	Normal
Much-greater	@MuchGt	≫	Normal
Much-less	@MuchLt	≪	Normal
Nabla	@Nabla	∇	Normal
Natural numbers	@NSet	N	Fake
Negation (logical)	@Not		Blank
Non-equivalence	@NEqv	≢	Substitute
Non-membership	@NotIn	∉	Substitute
Not	*See ''Negation (logical)''*		

Not-in	*See ''Non-membership''*		
Operator (generic)	@Op	OP	Fake
Or	*See ''Union (logical)''*		
Partial derivative	@Partial	∂	Normal
Planck's constant	*See ''H-bar''*		
Plus	@Add	+	Normal
Plus-in-circle	@OPlus		Blank
Plus-in-u	@UPlus	⊎	Substitute
Plus-or-minus	@Pm	±	Normal
Q.E.D.	@Qed	□	Normal
Q-set	@QSet	Q	Fake
Right angle bracket	@RAngle	⟩	Normal
Rational numbers	@RSet	R	Fake
Right arrow	@RightArrow	→	Normal
Similarity	@Similar	~	Substitute
Similar or equal	@SimEq	≃	Normal
Slash	*See ''Divided-by''*		
Slash-in-circle	@ODiv		Blank
Square	*See ''Q.E.D.''*		
Star	*See ''Asterisk''*		
Subset	@Subset	⊆	Substitute
Subset (proper)	@PrSubset	⊂	Normal
Superset	@Supset	⊇	Substitute
Superset (proper)	@PrSupset	⊃	Normal
Times	@Mult	×	Normal
Times-in-circle	@OTimes		Blank
Top	@Top		Blank
Triangle	*See ''Nabla'' and ''Delta''*		
Union (logical)	@Or	V	Substitute
Union (set)	@Union	∪	Normal
Union (square)	@SqUnion		Blank
Universal quantifier	@ForAll		Blank
Up arrow	@UpArrow	↑	Normal
Vertical bar	@VBar	|	Normal
Vertical bar (double)	@DVBar	‖	Normal

Table G-4: Special Characters for the Imprint-10

Informal name	Scribe name	Example	Availability
Aleph	@Aleph	ℵ	Normal
And	*See "Intersection (logical)"*		
Angle	@Angle	∠	Normal
Approximate equality	@Approx	≈	Normal
Asterisk	@Ast	*	Normal
Back arrow	*See "Left arrow"*		
Bottom	@Bot	⊥	Normal
Bullet (hollow)	@Circ	∘	Normal
Bullet (solid)	@Bullet	•	Normal
C-set	@CSet	**C**	Fake
Circle (small)	*See "Bullet (hollow)" and "Degrees"*		
Degrees	@Degr	°	Normal
Delta	@Delta	Δ	Normal
Divided-by	@Div	/	Normal
Dot-in-circle	@ODot	⊙	Normal
Down arrow	@DownArrow	↓	Normal
Empty set	@EmptySet	∅	Normal
Equality	@Eq	=	Normal
Equivalence	@Eqv	≡	Normal
Existential quantifier	@Exists	∃	Normal
For-all	*See "Universal quantifier"*		
Greater	@Gt	>	Normal
Greater-or-equal	@GtE	≥	Normal
Greater-or-less	@GtLt		Blank
H-bar	@HBar		Blank
In	*See "Membership"*		
Inequality	@Neq	≠	Normal
Infinity	@Infty	∞	Normal
Integers	@ZSet	**Z**	Fake
Intersection (logical)	@And	∧	Normal
Intersection (set)	@Inter	∩	Normal
Intersection (square)	@SqInter	⊓	Normal
Left angle bracket	@LAngle	⟨	Normal
Left arrow	@LeftArrow	←	Normal
Less	@Lt	<	Normal
Less-or-equal	@LtE	≤	Normal
Less-or-greater	@LtGt		Blank
Membership	@In	∈	Normal
Minus	@Sub	−	Normal
Minus-in-circle	@Ominus	⊖	Normal
Minus-or-plus	@Mp	∓	Normal
Much-greater	@MuchGt	≫	Normal
Much-less	@MuchLt	≪	Normal
Nabla	@Nabla	∇	Normal
Natural numbers	@NSet	**N**	Fake
Negation (logical)	@Not	¬	Normal

Non-equivalence	@NEqv	≢	Substitute
Non-membership	@NotIn	∉	Normal
Not	*See "Negation (logical)"*		
Not-in	*See "Non-membership"*		
Operator (generic)	@Op	**OP**	Fake
Or	*See "Union (logical)"*		
Partial derivative	@Partial	∂	Normal
Planck's constant	*See "H-bar"*		
Plus	@Add	+	Normal
Plus-in-circle	@OPlus	⊕	Normal
Plus-in-u	@UPlus	⊎	Normal
Plus-or-minus	@Pm	±	Normal
Q.E.D.	@Qed		Blank
Q-set	@QSet	**Q**	Fake
Right angle bracket	@RAngle	⟩	Normal
Rational numbers	@RSet	**R**	Fake
Right arrow	@RightArrow	→	Normal
Similarity	@Similar	∼	Normal
Similar or equal	@SimEq	≃	Normal
Slash	*See "Divided-by"*		
Slash-in-circle	@ODiv	⊘	Normal
Square	*See "Q.E.D."*		
Star	*See "Asterisk"*		
Subset	@Subset	⊆	Normal
Subset (proper)	@PrSubset	⊂	Normal
Superset	@Supset	⊇	Normal
Superset (proper)	@PrSupset	⊃	Normal
Times	@Mult	×	Normal
Times-in-circle	@OTimes	⊗	Normal
Top	@Top	⊤	Normal
Triangle	*See "Nabla" and "Delta"*		
Union (logical)	@Or	∨	Normal
Union (set)	@Union	∪	Normal
Union (square)	@SqUnion	⊔	Normal
Universal quantifier	@ForAll	∀	Normal
Up arrow	@UpArrow	↑	Normal
Vertical bar	@VBar	|	Normal
Vertical bar (double)	@DVBar	‖	Normal

Index

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 424 PITTSBURGH, PA

POSTAGE WILL BE PAID BY ADDRESSEE

UNILOGIC, Ltd.
Suite 240, Commerce Court
Four Station Square
Pittsburgh, PA 15219-1119

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 424 PITTSBURGH, PA

POSTAGE WILL BE PAID BY ADDRESSEE

UNILOGIC, Ltd.
Suite 240, Commerce Court
Four Station Square
Pittsburgh, PA 15219-1119

INFORMATION REQUEST

Scribe fascinates me. Please send more information to:

_____

_____

_____

Scribe program for the following computers:

_____

_____

Scribe printer support for the following printer types:

__ Letter-quality Printer __ Dot-matrix Printer __ Laser Printer
__ Photocomposer __ Other (specify): _____

Have salesperson call me. Phone: _____

READER COMMENT FORM

UNILOGIC would appreciate any comment you may have on Scribe or its documentation.

From (optional): _____

_____

_____

_____