SH20-1857-0

Installed User Program

SCRIPT/370 Version 3 User's Guide

Program Number: 5796-PHL

This guide contains usage and reference information for SCRIPT/370 Version 3, a text processing program that executes under the Conversational Monitor System (CMS) component of IBM Virtual Machine Facility/370 (VM/370).

The SCRIPT text processing program formats input files created by the CMS Editor. SCRIPT control words interspersed within textual data describe how SCRIPT is to format the output file. Functions performed by SCRIPT control words include:

- Single- or multiple-column formatting
- Automatic concatentation and justification of output lines
- Pagination control, including sizing, page numbering, and specification of up to six top and bottom titles to appear on every output page
- Indention or offset of blocks of text
- Automatic heading formatting and table of contents generation
- Extensive symbolic capabilities, including conditional processing control, communication via terminal input/output, and a macro capability.

IBM

PROGRAMMING SERVICES PERIOD

During a specified number of months immediately following initial availability of each licensed program, designated as the PROGRAMMING SERVICES PERIOD, the customer may submit documentation to a designated IBM location when he encounters a problem which his diagnosis indicates is caused by a licensed program error. During this period only, IBM through the program sponsor(s), will, without additional charge, respond to an error in the current unaltered release of the licensed program by issuing known error correction information to the customer reporting the problem and/or issuing corrected or notice of availability of corrected code. However, IBM does not guarantee service results or represent or warrant that all errors will be corrected. Any onsite programming services or assistance will be provided at a charge.

WARRANTY

EACH LICENSED PROGRAM IS DISTRIBUTED ON AN 'AS IS' BASIS WITHOUT WARRANTY OF ANY KIND EITHER EXPRESS OR IMPLIED.

First Edition (September 1976)

A form for readers' comments has been provided at the back of this publication. If this form has been removed, address comments to: IBM Corporation, 1049 Asylum Avenue, Hartford, Connecticut 06032. Attention: Mr. G. M. Saltman.

SCRIPT/370 Version 3 is an IBM Installed User Program (IUP) that runs under the Conversational Monitor System (CMS) component of IBM Virtual Machine Facility/370 (VM/370).

This user's guide contains "how to" and reference information concerning SCRIPT. Knowledge of VM/370, CMS, and the CMS Editor is required. Information about VM/370 and CMS is contained in the publications:

IBM Virtual Machine Facility/370:

Introduction, Order No. GC20-1800

Terminal User's Guide, Order No. GC20-1810

CMS User's Guide, Order No. GC20-1819

CMS Command and Macro Reference, Order No. GC20-1818

CP Command Reference for General Users, Order No. GC20-1820

How To Use This Book

Users who have had no experience with SCRIPT/370 will find basic information on its operation and control word processing in "Section 1. How To Use SCRIPT." The topics discussed in Section 1 are:

- Simple Text Formatting
- Formatting Pages and Titles
- Multiple Column Processing
- Head Levels and Tables of Contents
- Combining SCRIPT Files
- Automatic Formatting and Page Composition
- Symbol Processing
- Interactive SCRIPT Processing
- Writing SCRIPT Macros
- EasySCRIPT
- Automatic Hyphenation

"Section 2. The SCRIPT Command" describes the options available with the SCRIPT command of CMS.

"Section 3. Control Words" provides an alphabetical list of SCRIPT control words. Each control word description may contain:

- The format of the control word and a discussion of its operands and parameters (if any)
- Usage notes
- Examples

"Section 4. Error Messages" lists the error messages generated by the SCRIPT processor, and their associated return codes.

"Section 5. Installing SCRIPT/370" describes the procedure for installing the SCRIPT program on your system, and the procedure for verifying its operation.

There are two appendixes:

- "Appendix A: Summary of SCRIPT Control Words" lists all the SCRIPT control words, operands, and default values.
- "Appendix B: Compatibility of Version 3 with Earlier Versions" summarizes new SCRIPT control words and command options. Changes to control words and command options available in earlier versions of SCRIPT are described, as a conversion aid.

This book was entirely formatted by SCRIPT/370 Version 3.

Notational Conventions

In this book, whenever a control word or command format is shown, the following conventions are used:

- Stacked items represent choices: you can (or must) choose one of the items.
- If an item is shown within brackets, such as

  [n]

  it indicates that the operand or parameter is optional. You do not have to enter it.

- If one or more items are shown within braces, such as

  ```
  {ON }
  {OFF}
  ```

  it indicates that you must choose one or the other.

- If an operand or parameter is shown with neither brackets or braces, there is no choice: you must enter it.

- Keyword operands are shown in all uppercase letters, for example, NORM is a keyword operand of the .PN (PAGE-NUMBERING-MODE) control word. You must enter keyword operands exactly as shown.

- Variable parameters are shown in lowercase letters. For example, the letter

  ```
  n
  ```

  is always shown when a numeric parameter is required.

- Ellipses in examples, such as

  ```
  .
  .
  .
  ```

  indicate that text lines not pertinent to the example are not shown.

FIGURES

WHAT IS SCRIPT?

SCRIPT/370 Version 3 is a text processing program that runs under CMS, the Conversational Monitor System. CMS is the interactive time-sharing component of IBM Virtual Machine Facility/370 (VM/370). That's quite a mouthful: all those words assume that you know 1) what a text processing program is and 2) that you know about CMS and VM/370.

First, let's talk about CMS and VM/370: you have to know how to use CMS if you want to use SCRIPT. You have to know how to create and edit files using the CMS Editor, how to manage CMS disks and how to enter CMS commands. That kind of information is not provided in this publication: to learn about VM/370, how to log onto the system, and how to use CMS, see the VM/370 Terminal User's Guide, and VM/370 CMS User's Guide.

Now, text processing programs: what are they? Essentially, a text processing program (or simply, a text processor) is a program that reads input data, formats it into manuscript pages, and prints the results, which is the output. The input data consists of text that has been entered at a terminal that is attached to a computer. The terminal is just like a typewriter, except that the text is saved on a direct access storage device, or disk.

Since the data is stored, it can be retrieved and modified at any time, using an editing program. Then a text processing program, such as SCRIPT, can be executed to process the modified data and produce an output file that incorporates the changes. Additions or deletions of text are reflected in the output: SCRIPT knows how wide each page must be, and how many lines to print on each page. It fills up a page with text, then begins printing a new page automatically. It continues processing until it reaches the end of the input data, that is, the end-of-file.

All text processing programs can do these simple things. SCRIPT, however, is more flexible than other text processors. This flexibility takes several forms:

- SCRIPT is exceedingly simple and fantastically complex. How can this be? SCRIPT can format an entire document automatically, with no controls from you at all. All you have to do is enter the input text. However, there are an unlimited number of instructions (or combinations of instructions) that you can give to SCRIPT, to tell it how you want the text formatted. Thus, you can choose how simple you want SCRIPT to be.
- SCRIPT disk files are independently maintained. There is no special editor or editing system that you must use to modify or update your SCRIPT files. If you are using CMS, you use the CMS Editor. In addition, you can use many other CMS file system commands to modify, copy, or rearrange your SCRIPT input files.
- The imbed capability of SCRIPT makes it possible to combine many SCRIPT files to produce a single, integrated output file. These imbedded files can be arranged in any sequence, and while they are being processed, SCRIPT treats them all as if they were contained in a single, continuous input file.
- The most useful and exciting features of SCRIPT are its macro and symbolic capabilities, which make SCRIPT very much like a high-level programming language. You can define your own control words, conditionally process text, perform variable symbol substitutions, and even do simple arithmetic functions in a SCRIPT file.

So, you can see that while SCRIPT is a text processing program, it is much, much, more.

WHO USES SCRIPT?

SCRIPT can be used by anyone who wants to simplify the work of typing, correcting, printing, and examining letters, manuscripts, or memos. SCRIPT is used by:

- Secretaries
- Programmers
- Publications specialists
- Administration personnel

Any written document that is subject to changes or updates is a good candidate for SCRIPT processing. Once you have learned how to use SCRIPT, you will find that you can use it for just about all of your typing and writing needs. In fact, you will probably never want to use anything else!

HOW SCRIPT WORKS

SCRIPT reads input lines from CMS disk files. As it reads the lines, it formats them, and scans the text for "control words" which are recognized by SCRIPT as instructions to perform particular tasks.

CONTROL WORDS

Control words are normally identified by a period (.) in column 1. For example,

```
.sp
```

is a control word that tells SCRIPT to leave a blank space between text lines on output, and the control word

```
.pa
```

tells SCRIPT to start a new page of output.

Each control word has a short, 2-character name and a long, descriptive name. Thus, the .SP control word has the long name, SPACE-LINES, and the .PA control word has the long name, PAGE-EJECT. When you enter SCRIPT control words with input text into a SCRIPT file, you can use either the short or the long form, in any combination of uppercase and lowercase letters, but you must always precede it with a period. The control words are named so as to be as descriptive as possible, to make it easier for you to learn and to remember them.

The .SP control word, for example, may be entered in a variety of ways, among them:

```
.sp             .SPACE
.SPACE-LINES    .spa
.space-lines    .SPACE-L
```

The long names for the control words are provided for clarity; the short forms are the real control word names. You should be aware that if you use the long names of the control words, SCRIPT must perform extra steps to translate the long form to its short, two-letter equivalent.

In this book, when control words are used in examples, the short name is used; in text, either the short name or the long name may be used. When the long name is used, it is shown without the period (.).

CONTROL WORD OPERANDS

Most SCRIPT control words have operands or parameters, that you use to specify how you want a control word processed. Many control words accept the keyword operands ON and OFF, so you can inhibit or restore a SCRIPT function or setting. For example, CENTER is a control word that tells SCRIPT to begin centering output on the page, and

```
.ce on
```

and

```
.ce off
```

are control lines that start and stop centering (.CE is the short name for the CENTER control word).

Many control words accept numeric parameters, that you use to specify an extent or a value for a SCRIPT operation, for example

```
.sp 10
```

indicates that you want 10 lines of space

```
.sp 2
```

indicates that you want 2 lines of space, and so on. Control words that accept numeric parameters may also accept keyword operands. The CENTER control word, shown above, allows you to specify a particular number of lines to center, for example:

```
.ce 10
```

Each of the control word descriptions in Section 3 lists the operands and parameters accepted by each control word.

INVOKING THE SCRIPT PROCESSOR

You invoke the SCRIPT processor by issuing the SCRIPT command from the CMS environment of VM/370. When you issue the SCRIPT command, you must specify the filename of the CMS file that contains your SCRIPT input. The filetype of the file must be SCRIPT. For example, if you create a file with the CMS file identifier TEST SCRIPT, then to invoke SCRIPT to process the file, you issue the command

```
script test
```

SCRIPT responds with the line:

```
SCRIPT/370 VERSION 3, LEVEL n - mm/dd/yy
```

where n is the version level of SCRIPT/370, and mm/dd/yy is the date of that version level. If you are using a typewriter terminal, you next receive the message:

ADJUST PAPER, THEN PRESS RETURN

Then, SCRIPT processes the text and control words in the file, and the results are typed at your terminal.

If you are using a display terminal, the display screen goes into a MORE... status, and you must press the Cancel key (or equivalent) to see the first screenful of SCRIPT output.

Options of the SCRIPT command allow you to specify whether you want your output printed on the real system printer or written into a CMS disk file. For example, if you want the output from SCRIPT processing to be printed on the printer in the computer room, use the PRINT option:

```
script test (print
```

The SCRIPT command format, and a detailed description of each of the options, are shown in "Section 2. The SCRIPT Command."

WHEN DO YOU HAVE TO USE SCRIPT CONTROL WORDS?

A SCRIPT file may contain no control words at all. In this case, the output format of the SCRIPT file is determined by a set of default characteristics. By using SCRIPT control words, you can override these defaults, as well as provide additional kinds of controls over SCRIPT processing. The number of SCRIPT control words you need to learn is directly proportional to the complexity of the documents you want to produce.

When you create a SCRIPT file, some of the things you must consider are:

- How is the text formatted? Do you want to add spaces between lines or paragraphs? Indent lines? Create numbered or bulleted lists?
- What size paper are you using for SCRIPT output? How many lines of text should be on the page? How wide is it? Do you want special titles on the top or bottom of each page? Where, and in what format, do you want the page number to appear?
- Are you going to use multiple-column processing?
- Do you want to generate a table of contents listing major headings, and the page numbers on which they occur?
- How long is the final document going to be? Can you organize it into several input files and let SCRIPT combine them?
- Are you going to have illustrations? Are you going to create tables using SCRIPT? Do you need to leave blank pages or blank space so that artwork can be included later? How are you going to number the illustrations?
- Are you using variable information? Can you use symbolic names throughout a document to represent information that changes frequently?
- Do you want the SCRIPT processing to be interactive? Are there types of information you may want to enter during SCRIPT processing?
- Are you using the same sequences of control words frequently? Can you write a SCRIPT macro so that you do not have to rekey all the control words in the sequence each time you want to use it?

The remainder of Section 1 contains information describing how you can use SCRIPT control words to perform these functions in a SCRIPT file.

HOW DO YOU DECIDE WHICH CONTROL WORDS TO USE?

Experience and personal preference. This book describes many formatting techniques, and shows lots of examples. No one example or technique is necessarily the best way to format something: there are usually several ways to do the same thing. As you become more experienced in using SCRIPT, certain standard ways of doing things will evolve and may be accepted as installation standards where you work.

SCRIPT OPTIONAL FEATURES

In addition to the capabilities listed above, there are two SCRIPT features that are available as options. These are:

- Automatic hyphenation
- EasySCRIPT

The first of these is exactly what is sounds like: SCRIPT can decide when and where to hyphenate words that fall at the

end of an output line, thus making your output look more professional, and eliminating wasted space.

EasySCRIPT is another optional feature: it provides a number of high level GML (Generalized Markup Language) "tags" which you can use instead of SCRIPT control words to format a document. EasySCRIPT is a very useful tool for people who do not have a lot of time to learn about SCRIPT, but who must produce consistently-formatted documents in a short period of time. Since SCRIPT control words can be mixed with EasySCRIPT tags in an input file, you can learn EasySCRIPT first, then learn about other SCRIPT control words later, when you have time.

One of the things that SCRIPT does when it processes your input text is to format it, so that regardless of how you enter the input lines in the SCRIPT file, the output text is consistently justified in columns of a regular width. This formatting consists of two processes, which SCRIPT performs simultaneously:

- Concatenation - the process of spilling words back and forth from one line to another to fill out a column to its prescribed width, and
- Justification - the process of adding extra blanks between words in an output line so that the right edges are evenly aligned (right-justified).

Thus, lines that are entered in a SCRIPT file as:

```
The quick brown
fox
came over to greet the lazy poodle.
```

may be formatted as:

```
The  quick  brown  fox
came over to greet the
lazy poodle.
```

As SCRIPT reads input, it "saves" words until it accumulates enough words to fill an entire line. When the next word in the input would make the line too long, SCRIPT justifies and prints the line, then begins formatting the next line. When two input lines are concatenated, SCRIPT inserts one blank between the last word from one line and the first word from the next.

If you enter text in a SCRIPT file with no control words, all of the text is formatted as in the above example.[1]

Most writing that you do, however, requires some kind of formatting: paragraphing, for example. You can do this kind of formatting very easily in SCRIPT, using no control words at all, or you can use control words to provide the formatting. In either case, you must be familiar with the concept of "breaks."

[1]All of the examples of SCRIPT formatting in this book are shown, for convenience, with very short lines.

BREAKS

When you want an input line to begin a new line of output, and do not want it concatenated with the line above it, you must cause a break, so that SCRIPT prints the partial line that is being saved before processing the new line.

One of the simplest ways to cause a break in a SCRIPT file is to begin a line with one or more blank characters (by using the space bar on your terminal keyboard). When SCRIPT reads an input line that begins with a blank character, the formatting process is interrupted, all of the text that has accumulated for the current line is printed as is, even if more words would have fit on the line, and the next input line begins a new output line.

To create paragraphs in text, then, all you have to do is to enter spaces before each line that you want to mark a new paragraph. For example, the lines:

```
The quick brown
fox
came over to greet the lazy poodle.
   But the poodle was frightened
and ran away.
```

may appear on SCRIPT output as:

```
The  quick  brown  fox
came over to greet the
lazy poodle.
   But the  poodle was
frightened   and   ran
away.
```

You can cause a similar break using the SCRIPT control word, .BR (BREAK). The lines

```
The quick brown
.br
fox
```

result in the output:

```
The quick brown
fox
```

If you want text to have leading blanks, as for paragraphing, you must supply the blanks on the text line. For an alternate method of paragraphing, you may want to use the .PP (PARAGRAPH-START) control word, which provides a blank line space, and 3-character indention on the first line.

The paragraphs in this book were created with the .PP control word.

SPACES

If you want to leave spaces between lines of text, you can enter a line of blanks into the text file by pressing the space bar at least once on a line that has no other text, then pressing the Return or Enter key.

Instead of entering a blank line, you can use the .SP (SPACE-LINES) control word. Thus, the lines

```
The quick brown fox came over to
greet the lazy poodle.
.sp
But the poodle was frightened
and ran away.
```

are formatted as follows by SCRIPT:

> The quick brown fox came over to greet the lazy poodle.
>
> But the poodle was frightened and ran away.

The .SP control word allows you to enter a numeric parameter, indicating how many spaces you want to leave on the text output. For example,

```
.sp 5
```

indicates that you want to leave 5 lines of space in the text output. You can use multiple spaces when you want a heading or a title to stand out, for example the lines:

```
A Love Story
.sp 3
The quick brown fox
was eager
to meet the pretty poodle.
```

may result in:

> A Love Story
>
> The quick brown fox was eager to meet the pretty poodle.

The .SP control word also causes a break.

CONDITIONAL SPACES

When you enter input for a SCRIPT file, you cannot always be sure where a new page may begin. If a page eject happens to occur at a place in your file where you had provided for extra spaces, you may not want those spaces to appear at the top of the new page. For these situations, SCRIPT provides the control word, .SK (SKIP-LINES). As with the .SP control word, you can specify the number of spaces that you want to skip:

```
.sk 3
```

Skips are ignored if they happen to occur at the top of an output page. (If you are using multiple-column processing, the lines are ignored if they occur at the top of a column.) In any other situation, the .SK control word is equivalent to the .SP control word. You may find it convenient, therefore, to use the .SK control word whenever you want spaces, and the .SP control word only when you need to provide space at the top of a page (for example, if you need to leave room for an illustration at the top of the page).

MULTIPLE SPACES

If you want to produce output that is double-spaced or triple-spaced, there are control words that you can use to indicate to SCRIPT that while formatting is to continue for each text line, extra spaces should be inserted between each line on output. For double-spacing, use the .DS (DOUBLE-SPACE-MODE) control word:

```
.ds
```

After this control word is processed, all output text lines have an additional space between them. Any spaces or skips that you have placed in the file are doubled as well; if you have a .SP 2, then your output has four spaces.

An additional SCRIPT control word, .LS (LINE-SPACING), allows you to specify some other increment for output spacing. You can, for example, specify

```
.ls 4
```

so that there are four lines of space between each text line of output, and each .SP or .SK control word value is multiplied by four.

The .DS and .LS control words can both be canceled by the control word .SS (SIN-

GLE-SPACE-MODE), which returns SCRIPT output to normal single-spacing.

PAGE EJECTS

Default SCRIPT formatting is designed for paper that is 8 1/2 by 11 inches, that is, standard typewriter-size paper. As SCRIPT formats text, it keeps track of how many lines it has filled on a page; when it reaches the bottom of the page, it performs a "page eject" and continues output on a new page, pausing at the top of the page to print the top titles, if any. The default title, printed at the top of the page, indicates the page number. SCRIPT always keeps track of the current page number as it is processing.

If you are directing your SCRIPT output to the terminal, you may want to use the STOP option on the SCRIPT command line. The STOP option tells SCRIPT to pause at the end of each page of output, so that you have time to insert a new sheet of paper.

When you are entering SCRIPT input, you may want to force SCRIPT to begin a new page of output before printing the next text. To do this, use the .PA (PAGE-EJECT) control word:

```
.pa
```

This control word causes a break, so SCRIPT prints the last output line, then leaves the remainder of the current page blank.

The .PA control word also allows you to specify a numeric parameter, so that you can assign a page number to the new page. For example, if you are creating a SCRIPT file with a title page and then you want the second output page to be numbered page one, you can use the control word:

```
.pa 1
```

For an alternate method of suppressing the numbering of introductory pages, see the discussion of the .PN (PAGE-NUMBERING-MODE) control word in Section 3.

When you specify a page number with the .PA control word, the numbering sequence is reset and continues sequentially.

ODD AND EVEN PAGES

Page ejects also occur when you use the control words .OP (ODD-PAGE-EJECT) and .EP (EVEN-PAGE-EJECT). When these control words are encountered, SCRIPT performs either one or two page ejects, such that the next page to contain text is even-numbered (in the case of the .EP control word) or odd-numbered (in the case of the .OP control word). For example, if SCRIPT is currently processing output page numbered three and then reads the control word

```
.op
```

it does a page eject, prints only top and bottom titles on page four, and prints the next text on the page numbered five.

These control words are convenient when you are formatting a document that is to be published, and certain pages must begin on even- or odd-numbered pages.

OVERRIDING SCRIPT DEFAULT FORMATTING

When you are creating SCRIPT documents, there may be occasions when you do not want SCRIPT to concatenate and justify your input lines. You may want, for example, to present a simple list, such as:

```
Boston
Chicago
New York
Providence
```

If these lines are processed when SCRIPT formatting is in effect, these four names may be concatenated as follows:

```
Boston Chicago New
York Providence
```

To prevent this, you may use the .BR control word between each entry to force a break, or you can use the .FO (FORMAT-MODE) control word to suspend SCRIPT formatting:

```
.fo off
Boston
Chicago
New York
Providence
```

To restore normal formatting after suspending it, use the control word:

```
.fo on
```

Since ON is the default operand for the .FO control word, you can use:

```
.fo
```

You should use the .FO OFF control word when you create tables or charts in SCRIPT; but remember to turn formatting back on when you resume entering text.

SUSPENDING CONCATENATION AND JUSTIFICATION

The .FO control word suspends all formatting, both concatenation and justification. There are times, however, when you may want to suspend either one of these processes, but not both. SCRIPT, therefore, provides the .JU (JUSTIFY-MODE) and the .CO (CONCATENATE-MODE) control words, which can be used to suspend and restore justification and concatenation, respectively.

For example, you may want to produce SCRIPT output that resembles normal typewriter output, that is, "ragged right" output. In this case, you do not want to suspend concatenation, since you still want each line to contain as many words as can fit on it, but you do not want extra blanks inserted between the words to pad the line to a specific length. To achieve this format, use the .JU control word, specifying the OFF operand:

```
.ju off
```

When the .JU OFF control word is in effect, output is formatted as in the preceding paragraph.

To resume justification of input lines, use the ON operand of the .JU control word:

```
.ju on
```

Since ON is the default mode of operation for the .JU control word, you do not need to specify ON.

Concatenation can be suspended in a similar way. To restore concatenation, use the control word

```
.co on
```

Since ON is the default mode of operation for the .CO control word, you do not need to specify ON.

Since full format mode is made up of justification and concatenation, you can restore them both with .FO ON, even if they have been turned off individually with .CO OFF and .JU OFF.

INDENTING TEXT

When you are creating documents, you may want to set off paragraphs or portions of text by indenting them. This often improves the readability of a manuscript by emphasizing certain text. Using SCRIPT, you can conveniently cause paragraphs to be indented using the the .IN (INDENT) control word. For example, the lines

```
This line is not indented.
.in 5
This line is indented.
```

result in

```
This line is not indented.
     This line is indented.
```

The .IN control word acts as a break, so that text accumulated before the .IN control word is processed and printed, then the next text is processed.

The .IN control words effectively sets a new left margin for output text, so that when you want text indented, you do not have to enter blanks in front of the input lines (as you would for normal typing). When default SCRIPT formatting is in effect, SCRIPT continues to concatenate and justify input text lines that begin in column 1, but prints the output indented the number of character spaces you specify.

Here's another example:

```
These few lines of text
are formatted
with enough words
.in 5
so that you can
see how SCRIPT's formatting
process
.in +3
continues and may
.in -6
even be reversed, by using a
negative value.
```

These lines may result in:

```
These few lines of
text are formatted
with enough words
     so that you can
     see how SCRIPT's
     formatting
     process
        continues and
        may
  even be reversed,
  by using a negative
  value.
```

In this example, the first .IN control word shifts output to the right 5 spaces so that text begins in column 6. The second .IN control word requests that the current indention be incremented by 3 spaces, so the left margin is now in column 9. When you supply a negative value with the .IN control word, the margin is shifted to the left.

Input	Output
.in 5 .un 5 .up This is a Heading .sk Topics may frequently require elaboration, resulting in: .sk .of 5 1. Lists that enumerate exceptions. .sk .of 5 2. Subdivisions of the topic, that result in discussions of varying degrees of length. .of .sk .un 5 .up This is a Second Heading .sk There is a hierarchy involved here; some items are clearly more important than others. .sk We are illustrating the control words described below. .sk .of 8 INDENT moves the left margin of all subsequent output. .of 8 OFFSET moves the left margin of all but the next line. .of 8 UNDENT moves the left margin of only the next line. .in 0 .sk The end: the .IN control word clears everything.	THIS IS A HEADING Topics may frequently require elaboration, resulting in: 1. Lists that enumerate exceptions. 2. Subdivisions of the topic, that result in discussions of varying degrees of length. THIS IS A SECOND HEADING There is a hierarchy involved here; some items are clearly more important than others. We are illustrating the control words described below. INDENT moves the left margin of all subsequent output. OFFSET moves the left margin of all but the next line. UNDENT moves the left margin of only the next line. The end: the .IN control word clears everything.

Figure 1. Example of Indents, Offsets and Undents

INDENTS AND OFFSETS

Any INDENT control word cancels a current offset and resets the left margin. If you specify a positive or negative increment with the INDENT control word when an offset is in effect, the offset is canceled, and the new left margin is computed from the current indent value.

Both the .IL (INDENT-LINE) and .UN (UNDENT) control words use the current margin (the indent value plus the offset value) when computing the margin for the next line.

To achieve a format that has several levels of offsetting, you can combine the .IN and .OF control words. See Figure 1 for an example of one technique to use. If you frequently need to create documents using similar formats, you may want to use EasySCRIPT. EasySCRIPT is described later in this section.

When you create lists using indents and offsets, you should use a tab character to provide the space required between the item indicator (that is, the number or special character), and the first word of text. When SCRIPT justifies a line, it inserts extra blanks where blanks already appear on the line. If you use blank spaces following the item indicator, SCRIPT may add extra blanks when it justifies the line; if so, the first line may not be aligned with the remainder of the offset item.

SOME USES FOR TABS

Tab characters at the beginning of input lines cause breaks. Thus, you can conveniently use tab characters to begin paragraphs or to create tables or charts. For example, the lines

```
.tb 10
We are planting:
(TAB)Marigolds
(TAB)Peonies
(TAB)Cucumbers
.br
this year.
```

are formatted as:

```
We are planting:
          Marigolds
          Peonies
          Cucumbers
this year.
```

You should remember, however, to use the INDENT control word when you want to indent blocks of formatted text. If each line of text begins with a tab character, SCRIPT does not format the text.

Note, in the above example, the use of the .BR control word following the tabbed items. If SCRIPT formatting is in effect when these lines are processed, and there is no break after the last tabbed item, SCRIPT concatenates the next line with the last item on the list.

Tab Fill Characters

When you specify tab settings with the .TB control word, you can specify a character to be used as a "fill" character when SCRIPT formats the line. Ordinarily, SCRIPT uses a blank to pad a line through a tab position. However, if you specify a tab setting such as

```
.tb ./5
```

then enter a line that begins with a tab character, the positions normally filled with blanks are filled with periods instead. Thus, the line

```
(TAB)This line begins with a tab.
```

is formatted as:

```
.....This line begins with a tab.
```

You can specify a different fill character for each tab setting position you specify with the TAB-SETTING control word. If you do not specify a fill character, SCRIPT always uses a blank.

LET SCRIPT DO IT

There are several SCRIPT control words you can use to simplify the composition of pages and text or titles. Each of these control words accepts a text line as a parameter, and formats the line for you. Using these control words you can automatically:

- Center lines on a page
- Right-justify a text line or title
- Underscore, capitalize, or both underscore and capitalize a line

CENTERING AND RIGHT-JUSTIFYING LINES

The CENTER control word (.CE) adjusts a line on the page so that there are an equal number of spaces in each margin. The line:

```
.ce Chapter 1
```

may appear:

```
                Chapter 1
```

The RIGHT-ADJUST (.RI) control word adjusts a text line or title so that it is flush with the right margin. Thus,

```
.ri Chapter 1
```

appears as:

```
                                        Chapter 1
```

Both the .CE and .RI control words allow you to specify a numeric parameter, indicating how many input lines should be right-adjusted, as in:

```
.ce 4
```

```
After this control word is processed,
the next four lines from the input file
are centered within the current
margins.
```

The lines are not concatenated or justified.

You can also use, with the .CE and .RI control words, the operands ON and OFF:

```
.ri on
  .
  .
  .
.ri off
```

All the text lines between the .RI ON and

REMEMBER CONTROL WORDS THAT DO NOT CAUSE BREAKS: Many control words that provide format functions do not cause breaks, so you can type individual words or phrases of a sentence on different input lines, as you require. The underscoring and capitalizing control words are a good example of this.

The input lines

```
This
.up sentence
.us has several control
.uc words in
.up it.
```

result in

This SENTENCE has several control WORDS IN IT.

STACKING SCRIPT CONTROL WORDS

Whether you are using a few or many control words in text formatting, you can take advantage of SCRIPT's stacking facility. This allows you to enter more than one control word on a single input line, or to enter a control word and text on the same input line. To separate the control words, or the control word and text line, you can use a semicolon (;), for example:

```
.sk;.ce on
```

is equivalent to the two lines:

```
.sk
.ce on
```

Stacking SCRIPT control words is convenient whenever you need to use more than one control word between text lines. When you return to edit the file later on, you can see all the control words together, and this can save time, especially if you are using a typewriter terminal.

Redefining the Control Word Separator:

You must be careful when you use semicolons on text lines that are processed as control word lines. For example, the line

```
.us Be careful; semicolons end lines.
```

results, on output, in:

Be careful
semicolons end lines.

To avoid this problem, there is a special control word, .CW (CONTROL-WORD-SEPARATOR), that allows you to indicate a character other than a semicolon as the separator for stacking control words. For example, you can enter the line above as follows:

```
.cw
.us Be careful; semicolons end lines.
.cw ;
```

The .CW ; line restores the control word separator.

The .CW control word is useful when you are defining symbolic tags that are made up of more than one control word. For information on defining symbolic tags, see "Symbol Processing" later in this section.

WHEN YOU NEED A PERIOD IN COLUMN ONE

When SCRIPT processes input, every line that contains a period in column one is treated as a control word. If there is a period, and what follows is not a valid control word, SCRIPT issues an error message. What happens when you need to enter a period as text in column one?

You can use the .LI (LITERAL) control word, which tells SCRIPT that you do not want a line interpreted as a control word, for example:

```
.li ...and so it goes.
```

prints as

```
...and so it goes.
```

You can specify parameters with the .LI control word, if there are many lines that begin with a period, for example the sequence:

```
Study the following control words:
.li on
.DS
.PA
.IM.
.li off
This assignment is due on Monday.
```

results in:

Study the following control words: .DS .PA .IM. This assignment is due on Monday.

FORMATTING PAGES AND TITLES

The output pages that SCRIPT formats are designed to fit on standard typewriter paper, 8 1/2 by 11 inches. This default format is also suitable for standard 11-inch computer forms. SCRIPT pages have:

- 66 lines per page
- 60 characters of text on each line

The control words that determine these values are .PL (PAGE-LENGTH), which has a default value of 66 and .LL (LINE-LENGTH), which has a default value of 60. In addition, SCRIPT allows 6 lines at the top of the page and 6 lines at the bottom of the page for margins. These 12 lines are included in the overall page length; thus, the total number of text lines on a page is 54.

There are control words that determine the size of the margins. They are:

```
.TM (TOP-MARGIN)
.BM (BOTTOM-MARGIN)
```

By changing the values of these control words, you can adjust the dimensions of a page of output. Two immediate considerations are:

- The physical size of the paper on which you are printing SCRIPT output.
- The number of lines printed or typed per page on the output device.

The latter is important because SCRIPT assumes, in addition to 11-inch paper, that output is printed at 6 lines per inch (thus, the page length of 66). You can see, then, that if you are creating output that prints at 8 lines per inch, you may want to use the PAGE-LENGTH control word to reset this value:

```
.pl 88
```

Similarly, if you are printing 6 lines per inch on a different size of paper, for example 14-inch forms, you would also need to change the page length. In other words, the page length value should be the same as the number of physical print lines on the paper.

When you change the page length, the top and bottom margins do not change automatically. If you need to format an output page, for example if you need exactly 68 lines of text, you can use the .TM (TOP-MARGIN) and .BM (BOTTOM-MARGIN) control words to adjust the number of text lines that SCRIPT prints on each page. To print output with 68 lines of text on 14-inch paper printed at 6 lines per inch, you could use the control words:

```
.pl 84
.tm 8
.bm 8
```

to achieve an output format of 68 lines per page.

Usually, once you have set the page formats for a document you do not need to change them; the values for these control words remain in effect until you explicitly reset them.

Occasionally, however, you may need to adjust a page dimension during SCRIPT processing. Using the formatting example above, if you need to print a page using only 65 text lines, you can recalculate values for the top or bottom margin, or you can let SCRIPT do it:

```
.bm +3
```

increases the bottom margin by 3 characters, effectively decreasing the number of text lines on the page. To restore the original page depth, use the control word

```
.bm -3
```

The control words that are used to lay out the format of a page are summarized in Figure 2.

CHANGING THE LINE LENGTH

When you are changing the default dimensions of SCRIPT output, you must consider the width of pages as well as the length. The SCRIPT default, 60 characters, is designed for typewriters or printers that print 10 characters per inch. You can calculate how wide you want your text lines to be and use the .LL control word to set the page width, that is, to control the right-hand margin of your output.

```
.ll 89
```

The above line tells SCRIPT that you want 89 characters on a line. SCRIPT uses this amount to format output lines until you explicitly reset the line length. As with

the .PL, .BM, and .TM control words, you can adjust the line length by a positive or negative specification. The line

```
.ll -20
```

resets the line length to 20 less than the current value. If you use this control word in conjunction with an indent control word:

```
.in 20
```

you can effectively center formatted output on a page.

WHEN IS LINE LENGTH IMPORTANT?

When SCRIPT is concatenating text, the line length represents the maximum number of characters that SCRIPT prints on a line. If SCRIPT is not concatenating text (as when .FO OFF is in effect), then any lines that are longer than the current line print as they appear in the input file, and extend into the right margin.

The .LL control word also controls the width of top and bottom titles that print on output pages.

TOP AND BOTTOM TITLES

When you use SCRIPT to create documents that are more than a single page, SCRIPT provides, by default, a top title that appears right justified on the top of each output page:

```
                                        PAGE 26
```

SCRIPT provides control words for you to set titles that are printed on the bottom of each page as well as the top. You can use titles to indicate page numbers, chapter or section headings, document titles or form numbers, or almost anything you want. The control words to set titles are:

```
.ET (EVEN-PAGE-TOP-TITLE)
.OT (ODD-PAGE-TOP-TITLE)
.EB (EVEN-PAGE-BOTTOM-TITLE)
.OB (ODD-PAGE-BOTTOM-TITLE)
```

If you want the same titles on both even- and odd-numbered pages, you can use the control words:

```
.TT (TOP-TITLE)
.BT (BOTTOM-TITLE)
```

HOW TO SPECIFY TITLES

Each title consists of three parts, which are printed in the following positions according to the current line length:

```
part 1 is left justified
part 2 is centered
part 3 is right justified
```

These parts are specified in the title control word line by using four delimiters, usually a diagonal (/). When you set a title, you must specify what information you want to provide in each section of the title. To center the words "First Draft" at the top of every output page, you specify:

```
.tt //First Draft//
```

If you want this title to appear on the left side of every output page, you specify

```
.tt /First Draft///
```

To cancel a title (so that it no longer prints on output pages), you make all three parts null:

```
.bt ////
```

The above control word cancels any bottom title control words that may be in effect.

If you need to create a title that contains the diagonal character, you can choose any character that does not appear in the title as the delimiter:

```
.bt <<SCRIPT/370 is Fun!<<
```

If you use even or odd title control words (.ET, .OT, .EB, .OB) then the titles you specify appear only on the even- or odd-numbered pages. For example, the titles:

```
.eb /First Draft///
.ob ///June 1776/
```

result in the words "First Draft" appearing on the lower left of each even-numbered page of output and the words "June 1776" appearing on the lower right of each odd-numbered page of output.

Even and odd SCRIPT titles are convenient for formatting manuscripts that require folios or page numbers to be printed flush with the outside margins on left and right pages of books.

```
.hm 3
```

then SCRIPT allows 3 blank lines between the top title (if any) and the first line of text on the page. The top margin value, however, does not change. Instead, the position of the top title changes.

Increasing a top or bottom margin does not automatically change the heading or footing margins, either. If you try to decrease a top or bottom margin to a value too small to accommodate the titles plus the heading or footing margin, an error results.

PAGE NUMBERS

In addition to using top and bottom title control words to indicate and change how page numbers appear on an output file, there are a few fancy things you can do with the .PN (PAGE-NUMBERING-MODE) control word. Operands of the .PN control word allow you to specify:

PAGE NUMBERING: If you do not want SCRIPT to print the page number on output, but want SCRIPT to continue incrementing the numbers, use

```
.pn off
```

If you do not want SCRIPT to increment the page number counter, you can use the control word:

```
.pn offno
```

To restore either of the above, use:

```
.pn on
```

FRACTIONAL PAGINATION: You can also specify that you want fractional pagination to begin with the next even-numbered page:

```
.pn frac
```

If this control word is encountered while SCRIPT is processing page 46, then subsequent pages are numbered 46.1, 46.2, 46.3, and so on until the control word

```
.pn norm
```

is processed. Then, SCRIPT does a page eject and numbers the next page 47.

ROMAN NUMERALS FOR PAGE NUMBERS: When you want page numbers to be printed in lowercase roman numerals use:

```
.pn roman
```

This operand is useful for printing front matter, such as prefaces, forewords, and so on. To restore arabic numbering, use the control word:

```
.pn arabic
```

PREFIXES FOR PAGE NUMBERS: If your document uses special numbering schemes, such that each chapter or section requires a prefix in front of the page number, you can use the PREF operand of the .PN control word, for example:

```
.pn pref 1-
```

After this control word is encountered, a bottom title that is specified as:

```
.bt /&///
```

prints as

1-28

This operand is useful for texts that begin a new numbering scheme with each section.

All of the above operands affect titles that have a page number symbol (normally the &) in them.

In addition, the automatic table of contents generated by the heading level (.Hn) control words contain the page numbers in the format specified by the .PN control word. For information on creating tables of contents in SCRIPT, see "Head Levels and Tables of Contents" later in this section.

MULTIPLE TITLES

You can actually set up to 12 titles at one time in a SCRIPT file. These 12 possible titles are:

- Six lines of titles for even-numbered pages
- Six lines of titles for odd-numbered pages

When you want to use multiple titles, there are two things you must consider:

1. Specifying the text of the titles.
2. Allocating space for the titles to print.

All of the title control words accept numeric parameters of 1 through 6, which indicate the number of the title line. For top titles (.TT, .ET, .OT), title lines 1

MULTIPLE-COLUMN PROCESSING

SCRIPT can format output in up to nine columns. The control words that you might use for multiple-column formatting are:

```
.CD (COLUMN-DEFINITION)
.CL (COLUMN-LENGTH)
.SC (SINGLE-COLUMN-MODE)
.MC (MULTICOLUMN-MODE)
.CB (COLUMN-BEGIN)
.BC (BALANCE-COLUMNS)
```

When you want SCRIPT to process multiple-column output, use the following procedure to determine how to specify the control words.

1. Decide how many columns you want and how wide you want each column to be.
2. Decide how much space you want to leave between each formatted column. This space is called the "gutter."
3. Determine where you want each column to be placed on the page, based on the column length plus the gutter length.
4. Calculate the overall line length, which is the sum of the lengths of the columns plus the length(s) of the gutter(s).

For example, if you want to process text in three columns on paper that is 11 inches wide, you might first decide that each column should have 28 characters, and that you want to leave a 3-character gutter between the columns. You could use the control words:

```
.cl 28
.cd 3 0 31 62
.ll 90
```

The .CL (COLUMN-LENGTH) control word specifies the width of each column. The first parameter of the .CD (COLUMN-DEFINITION) control word indicates the number of columns, and the remaining parameters indicate how far each column is shifted from the left margin (its displacement).

Using the above example, if you want the first column to be printed in column 1, then specify 0 as the displacement. Then the second column can begin in position 31 (28 + 3), and the third in position 62 (31 + 28 + 3). The overall width is the total width of the columns (3 x 28) plus the width of the gutters (2 x 3), or 90, which you can use for the line length (.LL).

The LINE-LENGTH control word controls the width of top and bottom titles; therefore, you only have to include it if you have specified any top and/or bottom title control words, and if you want the titles to be right- and left-justified with the outside margins of the 3-column format. Let's see what happens when we use the three control words shown above:

****

Now we are having SCRIPT process in three column format, using the control words .CL 28, .CD 3 0 31 62, and .LL 90. We had to do a little more, however, to provide the nice spacing between the double and triple column formats.

To get line spaces across the entire width of the page, we restored single-column formatting. We did this with the control word:

```
.cd 1
```

before using the .CD 3 line.

While temporarily in single-column mode, we add spaces and some asterisks, to make the page look better. Similarly, we have to be sure and leave spaces after this short threecolumn example.

****

THE EFFECT OF COLUMN DEFINITIONS

Notice that the COLUMN-DEFINITION control word provides two functions: it defines the number of columns, then the displacements of each. If you enter the control word

```
.cd 1
```

after having previously entered .CD 3 0 31 62, SCRIPT output is formatted in one column, with a displacement of 0. You do not need to respecify this value; if you then later want to begin formatting in 3 columns again, you only need to specify

```
.cd 3
```

and the displacements of 0, 31, and 62 that

```
This text is shown in SCRIPT default format, with 60 characters per line of text.
No control words have been entered so far.
.sp
.cd 4 5 18 31 44
.fo off
Apple
Banana
Cantelope
Cherry
Fig
Grapefruit
Pear
Plum
Watermelon
.cd 1 0
.sp
.fo
Now the  list is ended. You can see that when you let SCRIPT do the formatting for
you, you can much more easily update the alphabetical list, when necessary.


When SCRIPT processes these lines, the result is:

    This  text  is  shown  in SCRIPT  default  format,  with  60
    characters  per line  of text.  No control  words have  been
    entered so far.

         Apple        Cherry       Grapefruit   Plum
         Banana       Fig          Pear         Watermelon
         Cantelope

    Now the list is ended. You can  see that when you let SCRIPT
    do the formatting  for you, you can much  more easily update
    the alphabetical list, when necessary.
```

Figure 3. Example of SCRIPT Multiple-Column Formatting

.bc on

Columns may be automatically made "ineligible" for balancing by SCRIPT. If so, then SCRIPT does not attempt to balance the columns when any of the conditions mentioned above occurs.

A column is made ineligible for balancing:

1. If it was started with a .CB (COLUMN-BEGIN) control word.
2. If it was started as the result of a .KP (KEEP) control word or a .CC (CONDITIONAL-COLUMN-BEGIN) control word. Keeps are also caused by some heading level control words.
3. If a keep has been processed in this column.

If, however, the new column that is started after any of the above results in a page eject, then the column is considered available for balancing.

HEAD LEVEL 3 (.H3): generates a table of contents entry and a topic heading. The table of contents entry is indented two characters; there are no skips before it. The topic heading is automatically capitalized, but not underscored in the text, with three skips before and two line spaces after it.

HEAD LEVEL 4 (.H4): generates a topic heading, but no table of contents entry. The topic heading is automatically underscored, but not capitalized, with three skips before and two line spaces after it.

HEAD LEVEL 5 (.H5): generates a topic heading, but no table of contents entry. The topic heading is automatically underscored and capitalized, with one skip before it. There are no spaces after it; a level 5 heading is concatenated with the text that follows.

The heading introducing each of the heading levels in this description are generated with the .H5 control word.

HEAD LEVEL 6 (.H6): generates a topic heading, but no table of contents entry. The topic heading is automatically underscored, but not capitalized, with one skip before it. There are no spaces after it; a level 6 heading is concatenated with the text that follows.

The function of a keep is provided for the topic headings generated with these control words. The current column must have enough room available to contain the topic heading, the spaces after it, and two additional lines. If there is not enough room, a column eject is performed.

SPACING AND PAGE EJECTS

SCRIPT control of heading levels takes advantage of conditional spacing capabilities. Thus, while a .H2 control word is generally followed by two spaces, these spaces are not generated if a .H2 is immediately followed by spaces or skips (such as might be generated by another .Hn control word, for example, .H3). This eliminates multiple spacing when headings are used without intervening text.

The .H1 control word causes a page eject only if SCRIPT is not already at the top of a page. There are some instances when you will want to precede a .H1 with a page eject control word:

- When you want to assign the page a specific page number (for example, with a .PA 1 control word).
- When you want an odd or even page eject (.OP or .EP).

CHANGING HEADING LEVEL DEFINITIONS

If any of the default characteristics for particular heading levels do not satisfy your requirements, you can change them with the .DH (DEFINE-HEAD-LEVEL) control word. Or, you can use the .DH control word to change the definition of a head level only temporarily. The .DH control word accepts keyword parameters that describe head level characteristics. For example, SPAF is a keyword that you use when you want to change the number of line spaces automatically generated following a heading; SKBF indicates how many lines to skip before the heading. The line

```
.dh 3 spaf 1 skbf 2
```

changes the number of line spaces generated after each .H3 control word from 2 to 1 and the number of skips generated in front of the heading from 3 to 2. To restore the default settings, enter

```
.dh 3
```

Characteristics you set or change with the .DH control word remain in effect until you reset them.

Heading level characteristics are also changed with you use the control word

```
.ez on
```

to initialize EasySCRIPT GML tags.

THE TABLE OF CONTENTS

When SCRIPT processes a heading level control word that requires a table of contents entry, it writes a line into a file called IKSUT2 SCRIPT. The information that is written into this file includes:

- The text of the topic heading.
- The page number of the page on which the topic heading appears.
- The revision code character that was in effect when the topic heading was processed.
- The head level definition that was in effect when the topic heading was generated. Even if that head level is redefined later, this table of contents

- Specify a formatting control for a text line passed with the control word (for example ".US line" or ".CE line").

Since the table of contents is in a special format, the results of any control words that attempt to modify the format are unpredictable. You may want, however, to provide additional spaces between headings, or to force column or page ejects where appropriate.

masterfile. If each unit of information is in a separate file, then when you want to move or remove information, you need only to change the position of the .IM control word in the masterfile, or to delete it.

In addition, small files may be easily shared by several masterfiles. Each masterfile may imbed the small file where appropriate. Thus, you do not need to keep duplicate copies of similar information in different places.

Third, you should keep in mind that while there may be a limit to the number of records that can be contained in a single disk file, there is no restriction on the number of files that SCRIPT can process. Also, using the CMS Editor, many different people can work on pieces of the same document simultaneously.

For convenience in updating and tracking SCRIPT files, it is recommended that you use one file as the masterfile for a SCRIPT document. This file can contain the formatting controls for page size, depth, column definitions, and so on, that are to be in effect for the entire document. The remainder of the masterfile may contain only the .IM control words that imbed the rest of the files.

When you are proofreading SCRIPT output files that contain many imbed files, you can use the NUMBER option of the SCRIPT command. When you use this option, SCRIPT prints, next to each output line, the filename of the file that is currently being processed, and the relative record number of the last input line SCRIPT had read when the output line was formatted. This feature also makes it very easy to update and correct your SCRIPT files.

If your document is in multiple-column format, you may need to specify a numeric parameter with the NUMBER option, such as

```
script mymaster (print number 2)
```

If you do not specify a numeric parameter, SCRIPT leaves 16 blanks between the filename and the text; in multiple-column formats, there may not be enough room on the paper to accommodate this many blanks.

The UNFORMAT option of the SCRIPT command recognizes the IMBED and APPEND control words. When these control words are processed and SCRIPT is not formatting output, the contents of the imbedded or appended file are written into the formatted output. That is, the IMBED and APPEND control words are actually processed, whereas all other SCRIPT control words are not.

DELAYED IMBEDS

There is a special kind of imbed file that you can create with the .DI (DELAY-IMBED) control word. This control word allows you to enter input lines into a special SCRIPT file named IKSUT1, which is the delayed imbed file. When you are finished entering lines into the IKSUT1 file, SCRIPT continues processing text. As soon as a page eject occurs (either naturally or because of a control word that causes a page eject), the IKSUT1 file is imbedded. For example, the following sequence shows how you might delay the inclusion of a few lines of text until the next page eject occurred:

```
.di on;.ce on
**********
*READ CAREFULLY *
**********
.ce off
.di off
```

When SCRIPT encounters a .DI ON control word, it begins accumulating text in the IKSUT1 file; it does not process the lines at this time, except to look for ".DI OFF" beginning in column 1. After the .DI OFF control word is found, SCRIPT continues processing output with the next line in the file. As soon as a page eject occurs, the delayed text is imbedded and processed before SCRIPT continues with the next input line in the file.

See Figure 4 for an example of the .DI control word. Notice that a numeric parameter is used with the .DI control word, instead of the ON and OFF operands. Figure 4 itself has been formatted using the .DI control word.

SAVING FILE STATUS

The .SA (SAVE-STATUS) and .RE (RESTORE-STATUS) control words are useful with delayed imbeds. These control words save SCRIPT settings for such things as indents, formatting, line lengths, and so on, and then restore them. Why would you want to do this? Delayed imbeds, like any imbed files, are processed as if they are a part of the original input file. With a delayed imbed, though, you cannot be sure what values are going to be in effect when the delayed file is actually imbedded.

For example, the left margin may be at column one when the .DI control word is first read, but a .IM control word may have been processed before the end of the page was reached. If you do not want the

entirely, regardless of whether the current file is an imbed file or not. When you use the QUIT control word, processing terminates after SCRIPT prints the remainder of the current page (and thus any bottom titles in effect) and after SCRIPT closes all open files. In contrast, QUICK-QUIT causes immediate termination of processing, without a final page eject. Thus, some formatted text on the last page might never have been printed.

A PROFILE SCRIPT

SCRIPT provides a profile capability (like a CMS PROFILE EXEC) via the imbed function. If you have a file named PROFILE SCRIPT on any of your accessed disks, then it is automatically imbedded before any other input text in the primary input file when you issue the SCRIPT command. SCRIPT uses the standard CMS disk search order to locate the PROFILE SCRIPT. If there is no PROFILE SCRIPT, no error condition results; SCRIPT just begins processing the primary input file.

You can use the PROFILE SCRIPT to contain:

- Frequently used set symbols or macro definitions
- Standard formatting controls or titling control words that you use for all of your files

If you do not want the PROFILE SCRIPT imbedded when you SCRIPT a file, use the NOPROF option of the SCRIPT command:

```
script test (noprof
```

This command option suppresses SCRIPT's search for a PROFILE SCRIPT.

on the terminal. You can circumvent this proofreading problem in some cases by making the translation conditional, using the .IF control word.

```
.if SYSOUT eq PRINT .tr * af
```

This control word line results in output translation of asterisks (*) only if output is going to the printer. For details on how this works, see the discussion of the .IF control word in Section 3.

This technique is also useful for hyphens, since the special hyphen (X'BF') that aligns with corner symbols is non-printable on a terminal, so you cannot see them. Thus, if you use

```
.if SYSOUT eq PRINT .tr - bf
```

you won't have this problem. (You should try to remember to translate the hyphen back to its normal state for text, since the X'BF' hyphen does not align properly when used for inter-word hyphenation.)

DRAWING BOXES

At last! SCRIPT has automated the process of drawing boxes around illustrations or text, and of formatting charts with horizontal and vertical lines. The control word that does it all is .BX (BOX), which you use in three steps:

1. Define the columns that you want to contain vertical rules, for example,

   ```
   .bx 1 10 20 30
   ```

   This control word initializes an overlay structure for subsequent text and formats and prints a box top:

   ```
   ┌────────┬─────────┬─────────┐
   ```

2. Each time you want a horizontal line across a box, use the control word

   ```
   .bx
   ```

   with no operands. The lines are drawn with intersections at the vertical rules:

   ```
   ├────────┼─────────┼─────────┤
   ```

3. When you want to finish a box, use the control word

   ```
   .bx off
   ```

   This terminates the overlay structure and draws a bottom line, again with the correct intersections:

   ```
   └────────┴─────────┴─────────┘
   ```

That's how the structure is provided. What happens to your text? After a box is started, SCRIPT processes and formats output lines as usual. Then, after each line is completely formatted and ready to print, SCRIPT overlays it with the current box structure.

Note, however, that if a column that has been defined as a vertical column already has a data character in it, it is not overlaid with a vertical bar.

Let's look at some examples of drawing boxes.

```
.bx 1 10 43
.in 2;.cl 41
.tb 10
.of 8
Item1(TAB)This is the first
item in a normal-looking
offset list.
.bx
.of 8
Second(TAB)This is the
second item
of a
normal-looking offset list.
.bx off
.of 0
```

When these input lines are processed, the result is:

```
┌────────┬──────────────────────────────────┐
│ Item1  │This is the first item of a       │
│        │normal-looking offset list.       │
├────────┼──────────────────────────────────┤
│ Second │This is the second item of a      │
│        │normal-looking offset list.       │
└────────┴──────────────────────────────────┘
```

Notice how the overlay structure complements the formatted text. By using the .IN and .CL control words, the left and right text margins are shifted two characters each way.

Notice, also, that the columns defined on the .BX control line are the exact column positions. Contrast this with the setting of the .TB control word (.TB 10, remember, results in spaces through column 10, with text beginning in column 11). This means you can use the same numbers for the .BX control word as for the .TB control word, and tab over to the column where the vertical bar will be.

The special corners and intersections that SCRIPT uses for format boxes are compound symbols made of corner characters. Since these characters are not available on

3. Delayed keeps are always printed in the next column, even if there is enough room for it in the current column.

When you want SCRIPT to do a keep, use the .KP (KEEP) control word to delimit the input that you want to keep together, for example:

```
.kp on
These text lines are all
part of a keep.
.kp off
```

You should always mark the end of a keep with the .KP OFF control word. If you do not, SCRIPT considers the keep ended when a full column of text has accumulated, or when certain control words that are not allowed in a keep are encountered, for example .PA or .CB.

You can also end a keep by beginning another keep, for example:

```
.kp on
These two lines
must be kept together
.kp on
These two lines
must be kept together.
.kp on
   .
   .
   .
```

Floating and delayed keeps are especially convenient when you have SCRIPT illustrations or tables in text, or when you want to leave blocks of text open for other kinds of artwork. These lines are used to create the floating keep that appears at the top of the next column:

```
.kp delay
.ce on
-------------------------
|  Sample Delayed Keep  |
-------------------------
.sk
.ce off
.kp off
```

If the control word .KP DELAY in this example is changed to .KP FLOAT, then the figure is printed as it is processed in the input file, if there are at least four lines remaining in the column when the keep is processed.

Using Keeps

Keeps are very useful for formatting text and illustrations, but you must be aware of two things if you are to use keeps properly.

```
-------------------------
|  Sample Delayed Keep  |
-------------------------
```

First, keeps do not cause breaks. If SCRIPT formats part of a line before encountering a .KP control word, it saves the line until the end of the keep is reached, then continues processing. This is convenient for many kinds of keeps, footnotes, for example. However, it can result in the last line of a paragraph that precedes a keep getting printed following the text that is within the keep. To avoid this, be sure that you cause a break before beginning the keep:

```
end of paragraph.
.br;.kp on
This line begins the new paragraph.
   .
   .
   .
.kp off
```

Many control words cause breaks, remember, so that if you have used a SPACE or SKIP before the keep, or an INDENT or OFFSET control word, you do not need to use the BREAK control word.

A second important thing to remember when you use keeps is that there are control words that are disallowed in keeps, among them the .CD (COLUMN-DEFINITION) control word (and the .SC and .MC control words, which also change column definitions). This means that you cannot, while processing multiple columns, use a keep to process a single-column illustration.

If SCRIPT encounters a disallowed control word while processing a keep, it terminates the keep. If you want to perform the function of a keep with an illustration that changes the column definition, you can use the .DI (DELAY-IMBED) control word, which is similar to a delayed keep. When you use a delayed imbed, the text of the "imbed" is always printed at the top of the next page. The DELAY-IMBED control word is described under "Combining SCRIPT Files" earlier in this section. For complete details, see the .DI control word format in Section 3.

FOOTNOTES AND HEADNOTES

Footnotes and headnotes are two special kinds of keep. Footnotes provide an automatic way of formatting footnotes to appear at the bottom of a column. SCRIPT does the calculation necessary to decide how many lines of text it can print in this column, so you do not have to decide where

When SCRIPT is formatting double columns, the revision character is placed within the gutter provided between columns, so that the second column is not shifted over. However, you must provide for the two characters used by .RC when you define more than two columns using the .CD control word.

If you have many .RC control words in a file, but you do not want them to print, you can change the control word that initializes them to a null (blank) character:

```
.rc 1
```

After this control line is processed, revision code 1 is considered a blank.

considers "&name," an undefined symbol, and performs no substitution, even if there had been a previous set symbol in the form

```
.se name = something
```

COMPOUND SYMBOLS

When substituting symbol names, SCRIPT performs as many levels of substitution as necessary. Each input line is scanned repeatedly until all symbols are substituted. This allows you to enter compound symbol expressions. For example, if the following symbols have been set:

```
.se x = 1
.se type1 = first
.se type2 = second
```

then the line

```
This is the &type&x try.
```

results in:

```
This is the first try.
```

For an additional example of compound symbol expressions, see "Symbols for the System Date and Time" below.

UNDEFINED SYMBOLS

When SCRIPT does not perform substitution on symbols beginning with "&" as in the case of any undefined symbol, or if "substitute mode" (controlled by the .SU control word) is set to OFF, the unresolved symbol remains in the file and is printed on output as it appears in the input file.

If you are familiar with the CMS EXEC processor, you might expect undefined symbols to be removed from a line. This is not true of SCRIPT symbols. The undefined symbols appear in your output, so you can see them.

In some cases, symbols may not be set until after the appearance of the substitution form, as in references to page numbers that occur later in the file. In this case, SCRIPT considers the symbol to be undefined and cannot perform substitution. There is, however, a SCRIPT command option that causes SCRIPT to read through an entire file twice: once to read the set symbols, and the second time to perform substitution, format, and print the document. This is the TWOPASS option:

```
script myfile (twopass
```

When SCRIPT processes the input file MYFILE SCRIPT, it reads the input twice; on the first pass, it reads and sets symbols, and on the second pass it performs substitution while it formats output lines. The TWOPASS option is also required if you want to generate an automatic table of contents to appear at the front of a document. Automatic tables of contents were described under "Head Levels and Tables of Contents" earlier in this section.

UNSETTING SYMBOLS

If you do not want to use a symbol in a file anymore, you can cancel the symbol using the .SE line:

```
.se symbolname off
```

Thereafter, the symbol name is treated by SCRIPT as an undefined symbol.

INHIBITING SUBSTITUTION

When you are creating an input file that uses expressions that contain literal ampersands, you may, as a precaution, want to turn substitution off while SCRIPT processes the input. To do this, use the .SU (SUBSTITUTE-MODE) control word:

```
.su off
&0 through &9 are special symbols.
```

Now, when SCRIPT processes this line, it does not attempt to substitute values, if any, for the symbols &0 and &9. To restore symbol substitution, use the control word:

```
.su on
```

RESERVED SET SYMBOLS

There are several groups of reserved symbol names that are automatically initialized and recognized by SCRIPT. The first group consists of eight symbols that you can use to obtain the current system values for the date and time. The second group contain values of SCRIPT control word settings. A third group of symbols, &0 through &9, are used in a special way by SCRIPT macros and by the IMBED and APPEND control words.

These symbols allow you to conditionally change some SCRIPT processing values. Consider the following example:

```
.se in = &$in
.if &in = 0 &in = 1
.bx &in &$cl
.in +2
.cl -2
text...
text...
.bx off
.in -2;.cl +2
```

The .BX control word begins a box structure using the current margins, whatever they are. The .IN and .CL control words shift the margins so that text is centered within the box. After the text is processed, the original values are restored.

As another example, consider the situation in which you want to leave a blank page with only a figure caption at the bottom. The file may be printed within different masterfiles requiring different page lengths:

```
.pa;.cm leave page for a figure
.se lines = &$lc - 1
.sp &lines a
Figure x. Sample Output
```

You will find many convenient uses for these special symbols, which are especially useful in writing SCRIPT macros.

THE &$RET SPECIAL SYMBOL

Another special symbol, &$RET, contains the return code from the last CMS command that was executed with the SYSTEM-COMMAND (.SY) control word. Initially, &$RET is 0.

&0 THROUGH &9

The special symbols &0 through &9 are set whenever a .IM (IMBED) or .AP (APPEND) control word is processed, or whenever a SCRIPT macro is invoked. Thus, during the processing of a long input file, these symbols may be set and reset many times.

The symbols &0 through &9 are considered undefined until they are explicitly set, or used. Since the symbol &0 contains the number of tokens passed on a .IM or .AP control word, it is reset whenever you use either of these control words.

ATTRIBUTE SYMBOL PREFIXES

SCRIPT provides, in addition to the reserved symbols discussed above, two attribute prefixes that you can use to test other symbols. These are E' and L'.

E' verifies the existence of a symbol. That is, it tells you whether or not there is a definition active for a symbol name. When you use the E' prefix, a string may be substituted with either 1 or 0, depending on whether or not the symbol has been defined. For example, if the following symbol has been set:

```
.se test = on
```

the line

```
The result is &E'&test..
```

is substituted as:

```
The result is 1.
```

If the symbol named "test" had not been set, the value of &E'&test would be 0. If a string is not a defined symbol name, as in

```
&E'czechoslovakia
```

then the result is also 0.

L' can be used to test the length of a symbol (or any token, for that matter). For example, after the lines:

```
.se test = 'This is a test.'
.se length = &L'&test
```

the value of "length" is 15. If the symbol named "test" had not been set, then "length" would have a value of 5.

Neither &E' or &L' can be used to test compound symbols. The result of

```
&L'&A&B
```

is always 4, regardless of whether &A and &B are defined symbols, and regardless of their lengths.

SOME THINGS YOU CAN DO WITH SET SYMBOLS

There is virtually no limit to the uses for set symbols, or symbolic names, in SCRIPT files. As you become familiar with SCRIPT, and are at ease using it, you will find many applications for symbol processing. Some techniques which you might want to use are shown below.

- To rearrange illustrations, update the .SE numbering scheme. You should not have to manually change the figure captions or references in the body of your text. These numbers are automatically substituted with their new values by SCRIPT. (You must, of course, remove references to figures that you have deleted.)

CONDITIONAL PROCESSING WITH IF AND GOTO

The symbolic capability of SCRIPT allows you to assign values to symbolic names. The high-level IF/GOTO capability provides a way of testing symbol values during SCRIPT processing, and to indicate how processing should continue, based on the results of the test.

The SCRIPT control words that provide this function are .IF, .GO (GOTO), and "..." (SET-LABEL). A sample sequence might look like the following:

```
.if &type = 1 .go bypass
   .
   .
   .
...bypass
```

In the above example, all the control words and text between the .IF and the ... control word (which sets the label "bypass") are skipped if the symbol "&type" has a value of 1. The conditions that you can test for and the codes you can use are:

Code	Meaning
eq or =	equals
ne or ¬=	is not equal to
gt or >	is greater than
lt or <	is less than
ge or >=	is greater than or equal to
le or <=	is less than or equal to

The target of a .IF control word does not have to be a .GO control word, for example:

```
.if &SYSHOUR >= 12 .im pm
.if &SYSHOUR < 12 .im am
```

In this example, a different file is imbedded (either PM SCRIPT or AM SCRIPT) based on whether the time is before or after 12 noon.

Conditional processing with the .IF control word can be especially convenient when one file is included as an imbed in several different masterfiles. You can provide for slight differences among the files by setting the same symbol to a different value in each masterfile, and using that symbol to determine how processing is done in the imbed file.

USING SYMBOLS TO INVOKE SCRIPT CONTROL WORDS

You can use the .SE control word to define high-level macro-like "tags." These tags may contain sequences of control words that you frequently use together.

For example, if you are creating a document that has several different formatting requirements, and you need to switch from a two-column format to a single-column format, to a three-column format, and so on, the existing control words for multiple-column formatting do not provide enough function to allow this kind of switching back and forth. Each of the required formats may require a column definition (.CD control word) and a column length (.CL control word) specification.

Since each symbol consists of at least two control words, then you must use the control word separator within the symbol, so that when the symbol is invoked, both control words are processed:

```
.se 1col = '.cd 1 0;.cl 72'
.se 2col = '.cd 2 0 46;.cl 43'
.se 3col = '.cd 3 5 25 45;.cl 15'
```

Notice what happens, however, when these .SE control words are processed: since the semicolon is the default control word separator, the first .SE control word is broken up as:

```
.se 1col = '.cd 1 0
.cl 72'
```

The second of these lines causes an error in SCRIPT processing. In order to prevent this, you must change the control word separator symbol.

Changing the Control Word Separator

Each time a control word line is processed, SCRIPT divides it into two pieces, the part before the first control word separator, and the remainder, which is saved for later. Thus, a line like:

```
.cw ?;.se B = ';.BR;'?.cw ;
```

is processed as follows:

word allows a symbol value to be delimited with apostrophes. In a COPY file, it's done with a nonblank ending character.

If this COPY file is in the macro library named SCRIPT MACLIB, then you can issue the following SCRIPT command to process a file that uses the symbol &B:

```
script test (lib (script
```

If you do not use the LIB option, then the symbol B is an undefined symbol.

The default filename for the symbol library is GML. If your MACLIB file has this name, then you can omit the filename on the SCRIPT command line (but you must still specify LIB, so that SCRIPT knows that it has to search a library).

ARRAY SET SYMBOLS

An array symbol is a special form of the set symbol that allows you to assign many values to the same symbol name and, on output, refer to all the values by referring only to the symbol name. An array symbol may be set with a control word in the form:

```
.se name() = symbol-value
```

where the parentheses indicate that this is an element of an array and 'symbol-value' is any expression that may legally appear on a SET-SYMBOL control word line.

When SCRIPT encounters the array symbol value in the form:

```
&name(*)
```

"&name(*)" is substituted with the values of all the currently defined array elements, in the order in which they were indexed, and formatted with commas and blanks separating the individual elements.

For example, if you are creating an index for a document, you could use array symbols on the pages you want to reference. If the line

```
.se list()=&
```

appears in a number of places in a document, then the expression

```
best sellers &list(*)
```

may result in the line

```
best sellers 10, 12, 20, 42
```

If, on expansion of an array symbol, the output line becomes too long, the first piece is used on one line, and the overflow is printed on the next.

CONTROLLING ARRAY ELEMENTS

Each element in an array has a number associated with it. In the example used above, 10 is the first element because it is the first one encountered; 12 is the second element; and so on.

You can explicitly name the array element you wish to set, by including a number within the parentheses:

```
.se list(1)=&
```

The above line sets element number 1 of the array that will be substituted with the symbol &list(*). When this symbol is used, this particular entry will be listed first, even if it is not the first one set.

This may be convenient for setting primary index entries, that you want listed in front of secondary entries. Here's another example:

```
.se name(1) = 1
.se name(47) = 2
.se name(25) = 3
.se name(2) = 4
.se name(3) = 5
```

then the expression

```
&name(*)
```

would result in "&name(*)" being substituted as follows:

```
1, 4, 5, 3, 2
```

In other words, SCRIPT inserts the array element values in ascending element index order, not in the order in which they were defined. Note that in this example there were several available but undefined element numbers in between those that were defined. On final array substitution, any undefined elements in an array are ignored on output.

Element Zero

Every array symbol has an element zero, represented by the expression

```
&name(0)
```

Element zero has two functions.

INTERACTIVE SCRIPT PROCESSING

When you use SCRIPT, you do not have to have all of your input text in final form when you issue the SCRIPT command. The previous discussion, "Symbol Processing", explained how SCRIPT can test and manipulate variable symbols. There are several methods you can use to communicate with SCRIPT to change or test variable symbols just prior to or during SCRIPT command processing.

There are three control words that cause SCRIPT to interrupt processing, issue a virtual machine read to your terminal, and allow you to enter a line or lines of input or data.

- The .RD (READ-TERMINAL) control word accepts data lines at a typewriter terminal during SCRIPT output. This control word is useful if you are creating form letters and want to enter names, addresses, or other kinds of variable information directly at the terminal.
- The .TE (TERMINAL-INPUT) control word accepts input lines of data or control words and processes them as they are entered.
- The .RV (READ-VARIABLE) control word allows you to assign a value to a symbol name during SCRIPT processing by reading it from the terminal.

The last two of these, .TE and .RV, are enhanced by the control word .TY (TYPE-ON-TERMINAL), which displays a line at the terminal during SCRIPT processing. (The line is not included in the printer or disk output file.) You can use the .TY control word to display prompting messages for the .TE and .RV control words.

The .TE control word accepts several operands. You can specify

```
.te on
```

so that SCRIPT continues reading input from the terminal until you type in

```
.te off
```

Then, SCRIPT processing continues with the next line in the file. You can enter SCRIPT control words or text (which SCRIPT formats according to the formatting controls in effect).

If you specify a numeric parameter with the .TE control word, such as

```
.te 4
```

then SCRIPT reads that number of lines before it continues processing.

You can also terminate terminal input with the control words .QU (QUIT), .QQ (QUICK-QUIT), or .EF (END-OF-FILE). When .QU or .QQ is encountered, SCRIPT terminates processing; the .EF control word indicates the end of the current file. (The .TE control word is essentially an imbed, where the "file" imbedded is your keyboard!)

The following example uses all three of these control words to process and format the same file an indefinite number of times.

```
...start
.im heading
.ty Enter NAME and ADDRESS (3 lines)
.te 3
.im letter
.ty Any more?
.rv answer
.if .&answer eq .yes .go start
```

The lines between the label ...start and the .IF control word are processed an indefinite number of times. As long as you continue to enter "yes" when prompted with the message "Any more?", SCRIPT loops back to the beginning of the file, prompts you for another name and address, and continues.

Notice how the .RV control word results in the setting of the symbol name. After a word is entered in response to the .RV control word, it is as if you had used the .SE control word:

```
.se answer = value
```

where "answer" was the name specified on the .RV control line, and "value" is the word (or expression) you enter from the terminal.

Once you have set a symbol in this manner, you can use the symbol "&answer" as you would any other set symbol. In the above example, a period is concatenated with the symbol name, so that if a null line is entered in response to the .RV control word, the .IF line does not cause an error message.

WRITING SCRIPT MACROS

Like many other high level languages, SCRIPT offers a macro capability, which allows you to define your own control words that are combinations of existing control words.

You can define macros with the .DM (DEFINE-MACRO) control word. In order for your SCRIPT macros to be processed by SCRIPT, you must use the .MS (MACRO-SUBSTITUTION) control word:

```
.ms on
```

Since SCRIPT macros are invoked as if they were control words, if you try to invoke a SCRIPT macro when you have not used the above control line, the macro is treated as an invalid control word.

HOW MACROS ARE DEFINED

When you define a SCRIPT macro, you must assign a name for it and then specify the input lines that are to be substituted whenever the macro is called. Some SCRIPT control words can be duplicated by macros. For example, the .PP control word could be defined as a macro as follows:

```
.dm pp /.sk/.il 3/&*
```

The control word components of the macro definition must be separated by any unique delimiter; a diagonal (/) is used most frequently. The final delimiter may be omitted.

The special symbol &* represents "the line," which is the line passed to the macro for formatting. Thus, when the line

```
.pp On second thought,
```

is processed, &* has a value of "On second thought,".

WHAT TO NAME YOUR MACRO

The macro name does not have to be two characters in length. It can be up to 10 characters, and may contain special characters. The name may be the same as the two letter name of a control word, but it may not be the same as the long name of a control word or any equivalent. In other words, a macro called "BR" will be invoked instead of the BREAK control word, but a macro called "BREAK" will not. This is because SCRIPT tries to find a two-letter equivalent for any long control word before looking it up as a macro or a control word. If the input line says ".BREAK," SCRIPT recognizes the word and changes it to ".BR". Thus, the thing called "BR", not the thing called "BREAK", is performed whenever either .BR or .BREAK is encountered. If you name your macro BREAK, it can never be invoked because SCRIPT will think you want the .BR control word when you try to invoke it.

SPECIAL SYMBOLS FOR MACROS

The .DM control word also recognizes the special symbols &0 and &1 through &9, which are assigned values when a macro is processed. &0 represents the number of tokens (words) that are entered in the line (&*). &1 is the first token, &2 the second, and so on. Thus, if a macro is called with the control line

```
.process fileb 10 filea no
```

then the following values are set:

Symbol	Value
&*	fileb 10 filea no
&0	4
&1	fileb
&2	10
&3	filea
&4	no
&5-&9	&5-&9 (undefined)

Note: The APPEND and IMBED control words accept tokens that are passed to imbedded files using the symbols &1 through &9; &0 is set to the number of tokens passed. These are the same symbols as those used in macro calls; therefore symbols set by the IMBED or APPEND control words are canceled by macro calls, and vice versa.

The symbols &1 through &9, if not set, are considered undefined by SCRIPT when it scans the line. Therefore if you create a macro as follows:

```
.dm ofs /.sk/.of &1
```

EASYSCRIPT

EasySCRIPT is a system generation option of SCRIPT[1]. The EasySCRIPT GML processor provides five SCRIPT formatting shortcuts that take advantage of SCRIPT to offer an elegantly simple way to format most documents. EasySCRIPT tags can be freely intermixed with standard SCRIPT control words. Using these shortcuts, you can:

1. Produce numbered or bulleted lists automatically.

2. Automatically format headings and a table of contents. And, if you want, you can have EasySCRIPT number your headings using the Dewey decimal numbering system. Then, when you add or delete information, the numbering is changed for you.

3. Format text in paragraphs aligned with the current indent level of a list or heading section.

EasySCRIPT functions may be invoked using the .EZ control word. The control word

```
.ez on
```

enables the EasySCRIPT GML tags.

EASYSCRIPT TAGS

There are five EasySCRIPT tags. Each tag provides two different sets of functions depending upon whether it is capitalized or not. The rule is that the capitalized version provides MORE function.

The five basic tags are:

1. &Hx -- Inserts a Dewey-decimal numbered heading of level x where x is 1, 2, 3, 4, 5, or 6.

   To create documents without the Dewey-decimal heading numbers, type the 'h' in the heading command in lowercase.

2. &P -- Start a new major paragraph. A major paragraph resets the indentation to zero and produces the necessary spacing.

   To maintain the current indentation for a minor paragraph, type the paragraph command with a lowercase 'p'.

3. &Nx -- Inserts a numbered item of level x where x is 1, 2, 3, or 4.

   If you do not want items numbered, enter the tag with a lowercase 'n'. A list is itemized at the level of indentation desired.

4. &B -- Inserts a bulleted item (one that begins with a •) under the current paragraph or numbered item.

   Sub-bullets (items that are introduced with hyphens) may be entered under bulleted items by typing the bullet tag with a lowercase 'b'.

5. &toc -- Generates a table of contents.

As you can see, all five EasySCRIPT tags begin with an ampersand (&). A tag may be connected to the line that follows with a period or with one or more blanks:

```
&TAG.line
```

is the same as:

```
&TAG line
```

These tags are enabled by the .EZ ON control word, but even if .EZ ON has not been processed, all of these functions can be invoked as operands of the .EZ control word. For example,

```
.ez H2 line
```

is valid if only EasySCRIPT was selected when the SCRIPT program was generated. If the .EZ ON control word has been processed, the same function can be invoked with the expression:

```
&H2 line
```

---

[1] For this reason, if your SCRIPT module was generated without EasySCRIPT, the .EZ control word is treated as an invalid control word.

1.0 THIS IS A LEVEL ONE HEADING

A level one heading, when printed:

1. Begins a new page and is right justified if it falls on an odd-numbered page.
2. Is typed all uppercase and underscored.
3. Is followed by five spaces and the text or a triple space and another heading.
4. The heading above is entered as follows:

   &H1 THIS IS A LEVEL ONE HEADING

1.1 THIS IS A LEVEL TWO HEADING

A level two heading, when printed:

1. Has a triple space above it.
2. Is typed all uppercase and underscored.
3. Is followed by a double space and the text or a triple space and another heading.
4. The heading above is entered as follows:

   &H2 This Is a Level Two Heading

1.1.1 THIS IS A LEVEL THREE HEADING

A level three heading, when printed:

1. Has a triple space above it.
2. Is printed in uppercase letters.
3. Is followed by a double space and the text or a triple space and another heading.
4. The heading above is entered as follows:

   &H3 This Is a Level Three Heading

1.1.1.1 This Is A Level Four Headin

A level four heading, when printed:

1. Has a triple space above it.
2. Is underscored.
3. Is followed by a double space and the text or a triple space and another heading.
4. The heading above is entered as follows:

   &H4 This Is a Level Four Heading

THIS IS A LEVEL FIVE HEADING: A level five heading, when printed, has a single space above it and is typed in uppercase without underscoring fcllowed by the text on the same line as is shown here.

The heading above is entered as follows:

```
&H5 This Is a Level Five Heading: A level five heading, when...
followed by text on the...
```

This Is a Level Six Heading: A level six heading has a single space above it and is typed in uppercase and lowercase and underscored followed by the text on the same line as is shown here.

The heading above is entered as follows:

```
&H6 This Is a Level Six Heading: A level six heading...
by the text on the same...
```

Figure 5. Sample EasySCRIPT Heading Formats

b. This is item two of a second level numbered item. enough to show format.

3. This is item three of a first level numbered item.

   This is a minor paragraph placed underneath the item and it shows how the indent is maintained underneath the item.

   a. This is another item one of a second level numbered item.

      This is a minor paragraph placed underneath a level two numbered item to illustrate how the indention is maintained.

      - We can put bulleted items under any level of indention.

        The item above was entered as follows:

        &B We can put bulleted items...

      - Here is another bullet.

        - And, of course there is always a sub-bullet!

          The item above was entered as follows:

          &b And, of course...

      1) This is item one of a third level numbered item.

         The item above was entered as follows:

         &N3 This is item one of a third...

      2) This is item two of a third level numbered item. This added sentence shows how longer items are formatted.

         a) This is item one of a fourth level numbered item.

            The item above was entered as follows:

            &N4 This is item one of a fourth...

            - Here is a bulleted item under the fourth level of numbered item.

              Minor paragraphs here? Certainly! And we manage to maintain the indentions and not foul up any counting!

              - As many bullets as necessary to do the job.

                - Sub-bullets, too.

                - There is so much indented now that little space is left for text.

         b) This is item two of a fourth level numbered item.

      3) And we can go back to level three.

   b. Or level two.

      A minor paragraph tag maintains any existing indention as shown here.

4. Or level one.

UN-NUMBERED LISTS

Un-numbered lists can be formatted using the '&Nx' tag with a lowercase 'n'. Following are some examples of un-numbered lists:

This is item one of a level one indention.
This is item two of a level one indention.
This is item three.
This is item four.

   This is item one of a level two indention.
   This is item two of a level two indention.
   This is item three.
   This is item four.

      This is item one of a level three indention.
      This is item two of a level three indention.

AUTOMATIC HYPHENATION

The SCRIPT hyphenation facility is a system generation option of SCRIPT[1]. It consists of an algorithmic hyphenation program, an exception dictionary, and a special editor for the dictionary.

When you want SCRIPT to use hyphenation when formatting output lines, use the .HY (HYPHENATE) control word:

```
.hy on
```

To cancel hyphenation, use the control word

```
.hy off
```

To suppress hyphenation at the end of a paragraph, use the control word:

```
.hy sup
```

This causes hyphenation to be interrupted until the next break occurs; then hyphenation is automatically resumed.

HOW THE HYPHENATOR WORKS

When SCRIPT is formatting a line and the next word does not fit in the current line, SCRIPT tries to hyphenate the word if at least four spaces remain on the current line.

To find out where the word should be hyphenated, SCRIPT:

- Searches the exception dictionary, if there is one, to see if there is an entry for the word. The hyphenation exception dictionary is discussed below.
- If the word is not in an exception dictionary, SCRIPT uses a special algorithm to determine the best hyphenation point (such that the longest piece of the word remains on the current line).

---

[1] For this reason, if your SCRIPT module is generated without hyphenation, the .HY control word is treated as an invalid control word.

ALTERING THE HYPHENATOR'S CHARACTERISTICS

If you want more or less frequent hyphenation, you can change the default characteristics of the hyphenator. There are two internal values kept by the hyphenator: THRESH, which is the hyphenation threshold (the minimum number of blank characters that must remain on the line before SCRIPT attempts to hyphenate the next word); and MINPT, which is the minimum hyphenation point (the smallest acceptable hyphenation point for the next word).

The default values are 7 for THRESH and 4 for MINPT. To change them, use the SET operand of the .HY control word:

```
.hy set thresh 6
.hy on
```

After these control words are encountered, then there must be at least six blanks left on the line before SCRIPT tries to hyphenate the next word.

You should keep in mind that using hyphenation with SCRIPT requires overhead in processing time; the larger the threshold value that you use, the less hyphenation is done, so processing time may be improved.

HYPHENATING SINGLE WORDS

Regardless of whether SCRIPT is using automatic hyphenation or not, there may be occasions when you would like a word to be hyphenated if it occurs at the end of a line. The .HW (HYPHENATE-WORD) control word allows you to specify that you want a word hyphenated, if necessary, and to specify how it should be hyphenated.

This may be convenient for long words that are normally hyphenated, or for words that may occasionally need hyphenation, for example:

```
Guinevere's
.hw lighter--than--air
laughter was heard
.hw through-out
the kingdom.
```

When this line is processed, SCRIPT uses the hyphens supplied as hyphenation points

to jump all over the place. If you insert a word that is already in the dictionary, the new one will be put on the line next to the old one. You may then go up or down one and delete one of the two. The FIND subcommand ignores hyphens, so you can find any word no matter how it is hyphenated.

HYPEDIT does not have as many subcommands as the regular editor. If you try to use one that is not allowed, you will get a ?HYPEDIT: message. About the only one you will miss is the CHANGE subcommand. Because of the encoding that must be done on each word, no changes are permitted once the word has been entered. To change an entry, you have to delete the current one and insert the new one. The basic subcommands, INPUT, QUIT, SAVE, UP, DOWN, DELETE, and FILE, work just the same way as in the standard CMS EDIT command.

The following are examples of valid SCRIPT command lines:

```
script test (print page(10) adjust(4))
script resume nowait quiet
```

You may truncate the name of any of the options to the minimum length that is not ambiguous. If you use a truncation that is ambiguous, SCRIPT uses the first option it finds that fits. The options are listed below in the order in which they are scanned, so you can tell from this list which option would be used if you gave an ambiguous name, such as "N".

The options may be given in any order; if two conflicting options are entered, the last one processed takes effect.

ADJUST

The format of the ADJUST option is:

```
ADJUST [ (nn[) ]]
```

This option causes output to be shifted over "nn" spaces. The number may be one or two characters long; if no number is given, 30 is assumed. (If the defaults are used for page dimensions and the output is printed on standard 14-inch wide paper, shifting the output 30 spaces to the right causes it to be centered.) The right parenthesis is required only if more options follow.

CENTER

The format of the CENTER option is

```
CENTER [ (nn[) ]]
```

CENTER is another name for ADJUST. The ADJUST and CENTER options are identical.

CONTINUE

The CONTINUE option causes processing to continue after an error condition has occurred, and an error message has been displayed. Severe errors and terminal errors cause SCRIPT to terminate even if CONTINUE is specified.

FILE

The FILE option causes formatted SCRIPT output to be written into a disk file. The output file is named "$filename SCRIPT", where filename consists of the first seven characters of the filename given on the command line. If $filename SCRIPT already exists, it is replaced; no message is issued to tell you that the old one is being erased.

The filemode of this file is An, where n is the same as the mode number of the primary SCRIPT input file. If there is no read/write A-disk when the FILE option is used, SCRIPT terminates processing.

The output may be in either printer or typewriter format, depending upon whether the PRINT option was specified. This means that if you do not specify the PRINT option when you specify the FILE option, then the file will be in typewriter format by default.

PRINT

The PRINT option specifies that output is to be formatted in printer format, one of the two basic SCRIPT output formats. If the FILE option is not specified, the PRINT option also causes the output to be directed to the offline printer. (If FILE is specified, PRINT merely controls the output format; the destination is a disk file.)

TERM

The TERM option specifies that output is to be formatted in typewriter format, one of the two basic SCRIPT output formats. If the FILE option is not specified, the TERM option also causes the output to be directed to the terminal. (If FILE is specified, TERM merely controls the output format; the destination of the output is a disk file.) TERM is the default mode of operation.

PAGE

The PAGE option allows you to selectively print pages of formatted SCRIPT output. The PAGE option has several formats; these are:

- SCRIPT fn PAGE (frpage topage

  "frpage" indicates the page number of the first page you want printed and "topage" indicates the page number of the last page you want printed.

- SCRIPT fn PAGE (frpage ONLY

  prints only the page specified as frpage.

- SCRIPT fn PAGE (frpage
  SCRIPT fn PAGE (frpage *

  either of the above commands print SCRIPT output from the page specified to the end of the file. frpage may also be specified as an asterisk (*). This means that printing should begin with the current page, whatever the page number is.

- SCRIPT fn PAGE (PROMPT

  tells SCRIPT that you want to specify several pages or page ranges. SCRIPT prompts you to enter a page range, and you can enter any of the frpage/topage combinations shown above.

The page ranges must be entered in the order in which the pages appear in SCRIPT output, for example,

```
script myfile (page 6 1
```

is valid if, following the page numbered 6, there is a control word

```
.pa 1
```

that numbers the next page 1.

When you have specified the PAGE option, SCRIPT continues prompting you until the end of the file is reached or until you enter a null line in response to a prompting message. If there is no page with the number you have given, or SCRIPT has already passed the page with that number, SCRIPT may reach the end of processing without printing anything. There is no error condition.

SYSVAR

The format of the SYSVAR option is:

```
SYSVAR (n val n val ... [) ]
```

This option allows you to set special symbols from the command line. Each "n val" pair causes the symbol "&SYSVARn" to be set to the value "val". "n" may be any single number or letter, and "val" may be any combination of letters and numbers up to eight characters long, but without any imbedded blanks or parentheses. Since these values are part of a CMS command line, they will be converted to uppercase letters.

The maximum number of SYSVAR pairs is limited only by the length of the command line you can enter.

TWOPASS

The TWOPASS option causes two passes to be made through the input files; both passes process all the control words, but output occurs only on the second pass. If this option is not in effect, SCRIPT formats and outputs everything in one pass. Two passes are required if a symbol value is needed earlier in the book than it is set. Usually, the requirement for two passes results from having a table of contents at the front of a book. A first pass through the book is then needed so that SCRIPT can find out what page numbers to use for all the table of contents entries. Since two passes take twice as long as one pass, it is good practice to put your table of contents at the end of your book; those pages can always be moved to the front of the document before it is used.

The TWOPASS option may also be effective for correcting SCRIPT files that may contain errors. Since no output is performed on the first pass, you can locate and correct SCRIPT errors before any actual output is printed.

... (SET-LABEL)

The SET-LABEL control word marks a line of your SCRIPT file so that that line may be referred to in a .GOTO control word. The format of the ... control word is:

```
| ... | label [line] |
```

where:

label is a name of up to eight characters that can be used to refer to this line of your SCRIPT file.

line is the active part of this input line. The first nonblank character after the label is treated as the beginning of the line; it may therefore be a control word, but a text line associated with a label may not begin with blanks. If the input line has a label only and no active line, then the next line to be processed is the one following the labeled line.

Usage Notes

1. When the ... control word is encountered, SCRIPT saves the information necessary to enable it to find this line again if a GOTO control word is encountered. Any valid SCRIPT input line may follow the label, or the label alone may occupy this line of the SCRIPT file.

2. Use of labels and the .GO (GOTO) control word is restricted to one input file. That is, when a new file is imbedded or appended, a new set of labels is started for that file. SCRIPT can only branch to a label within the same input file.

3. Every label in a particular file must be unique. If two identical labels are found in the same file, an error message is generated.

4. The storage area where SCRIPT saves label information is large enough for approximately 120 labels. If another SET-LABEL control word is encountered after this storage area is full, an error message is generated. Labels are kept in the table as long as the associated file is open. When a file is closed, its labels are removed from the table, and that space is available for more labels. A file is closed when the real end is reached or when an APPEND control word is processed, but not when an end-of-file is simulated by the .EF (END-OF-FILE) control word, unless you specify the CLOSE operand.

5. The label/GOTO function can be relatively inefficient. You should use it sparingly in situations where it is the best way to achieve the required results. When going to a label that is later in the input file, it is most efficient when the label is not far from the GOTO; when going to a label that is earlier in the file, it is most efficient when the label is near the beginning.

6. A space is not required after the control word itself, if the short form is used. (This is the only control word where this is true.) To set a label called "HERE", either "... HERE" or "...HERE" may be used.

7. The ... for a label must begin in column 1.

Example

Suppose you had a file called REPORT1 that contained a summary of activity for January, another file, REPORT2, for February, REPORT3 for March, and so forth. Now, if you wanted to create a year-to-date report by imbedding all the report files up to last month's report, you could use this sequence of SCRIPT control words:

```
.se ctr = 1
...loop .im report&ctr
.se ctr = &ctr+1
.if &ctr lt &SYSMONTH .go loop
```

The first time the IMBED (.im) is processed, the value of the symbol "&ctr" is 1, so the filename "report&ctr" becomes "report1." The next control word adds one to the value of the symbol; it is now 2. If the month is later than March (month 03), then the value of the counter is less than the month number, and the loop is processed again. This time the filename "report&ctr" becomes "report2." The loop continues until the counter is equal to the current month number.
XXX

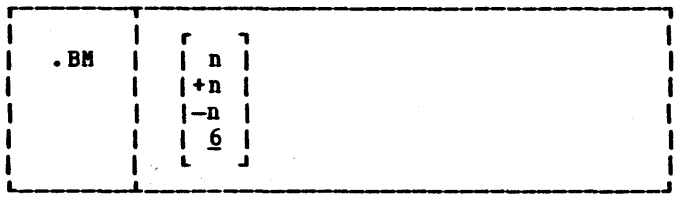
or by a HEAD-LEVEL control word or a keep that causes an eject to a new column.

4. If a page eject occurs while processing multiple columns, this does not mark the current column ineligible for balancing. A column eject that changes the current column from the last column of a page to the first column of the next page is the same as a page eject. Unlike intra-page column ejects, it does not mark the old current column as ineligible for balancing.

***

.BM (BOTTOM-MARGIN)

Use the BOTTOM-MARGIN control word to specify the number of lines to be skipped at the bottom of output pages, overriding the initial value of six. The format of the .BM control word is:



where:

n specifies the number of lines to be skipped at the bottom of output pages. n must be large enough to accommodate the footing margin (.FM) and the footing space (.FS), both of which are allocated from the bottom margin area. If +n or -n is specified, the current value of the bottom margin is incremented or decremented. If no value is specified for n, the default value of 6 is used.

Usage Notes

1. The value set by the .BM control word applies on the current page and all subsequent pages until another .BM is encountered. If there is not enough room left on the current page for the new bottom margin, the new value does not take effect until the next page.

2. The value given may not be so large that the top margin plus the bottom margin fill the entire page. An error message is issued if you try to set the bottom margin to more than the page length minus the top margin. If you intend to increase the bottom margin so that you can increase the footing margin or the footing space beyond what the old bottom margin would allow, be sure to do it in that order. The rule is, increase the bottom margin before the footing margin or footing space, but decrease the footing margin or footing space before the bottom margin.

3. This control word acts as a break. It is not allowed in a keep.

***

.BR (BREAK)

Use the BREAK control word to ensure that the next input line is not concatenated with the previous line or lines. The format of the .BR control word is:

Usage Notes

1. The .BR control word is necessary only when SCRIPT is concatenating input lines; it causes the preceding line to be typed as a short line, if it is shorter than the current column length.

2. Many other control words have the effect of a break. No BREAK control word is necessary when one of these is present.

3. A leading blank or tab on an input line has the effect of a break.

4. The .BR control word can ensure that some other control words are not effective too early or too late, for example:

```
.br;.tr $ 40
```

may be used to prevent the translation from being effective on the preceding text line, and:

```
.tr $ $;.br
```

may be used to make sure the translation does not affect the next line.

Example

```
Heading:
.br
First line of paragraph
```

```
┌──────┬──────────────────┐
│      │ ┌v1 v2 [...[vn]]┐ │
│ .BX  │ │OFF            │ │
│      │ └               ┘ │
└──────┴──────────────────┘
```

where:

v1 - vn are the positions at which you want to place vertical rules in output text. This format of the control word initializes the box and draws a horizontal line, with vertical descenders at the columns indicated. Subsequently, entering the .BX control word with no operands causes SCRIPT to print a horizontal line with intersections at the columns indicated.

OFF causes SCRIPT to finish drawing the box, by printing a horizontal line with vertical ascenders at the columns in effect.

Usage Notes

1. The .BX control word describes an overlay structure for subsequent text that is processed by SCRIPT. After the .BX v1 v2... line is processed, SCRIPT continues formatting output lines as usual. However, after a line is completely formatted and before it is printed, SCRIPT places vertical bars in the columns indicated by v1, v2, and so on, unless a column is already occupied by a data character. In this case, SCRIPT does not place a vertical bar in the column.

2. The .BX control word causes a break.

3. There are two sets of characters used in drawing boxes. If the output is in "typewriter" format, boxes are formed with dashes (-), vertical bars (|), and plus signs (+). If the output is in "printer" format, special box drawing characters available in the TN character set are used.

4. A .BX control word with different columns specified may be used while a box is being drawn. When this happens, vertical ascenders are put in for all the old columns and vertical descenders are used for all the new columns. The vertical rules are then placed in all subsequent output lines in the new columns designated.

5. The column specification for the .BX control word uses a different rule than is used elsewhere in SCRIPT. In control words like .IN, .TB, .CD, the numbers in the control word represent not columns but displacements. The SCRIPT control word .TB 5 means that a tab character should be expanded to enough blanks to fill up through column 5; the next word starts in column 6. In the .BX control word, .BX 5 means to put vertical rules in column 5. Thus, you can use the same numbers for a .TB control word as for a .BX control word, and the vertical bar will be placed in the column just before the beginning of the word following a tab. Further, you can define a box that is to be the full column width symbolically with the following control word:

```
.bx 1 &$cl
```

because the number represents the actual column where the vertical rules should be placed.

Example

There is a SCRIPT file called MARYHADA SCRIPT that looks like this:

```
Mary had a little lamb,
Whose fleece was white as snow,
And everywhere that Mary went,
The lamb was sure to go.
```

The following input sequence could be used to center this material in a box that is the same width as the current column length:

```
.bx 1 &$cl
.ce on
.im maryhada
.ce off
.bx off
```

The result:

```
┌──────────────────────────────────┐
│      Mary had a little lamb,     │
│   Whose fleece was white as snow,│
│    And everywhere that Mary went,│
│       The lamb was sure to go.   │
└──────────────────────────────────┘
```

***

.CB (COLUMN-BEGIN)

The COLUMN-BEGIN control word causes subsequent text to start a new column of output. The format of the .CB control word is:

The displacements of the columns do not control how wide the columns are to be; you must set the column lenth (using the .CL control word) to control this. If the current column length is greater than the number of positions between columns, there is no error; SCRIPT simply overlays part of the first column with the second. (It would be possible to define all columns to begin in the same position. If you did this, an entire column would be overlayed with the text of a later column.)

The gutter between columns is obtained by defining the column length to a value less than the distance between column starting positions.

2. If you specify fewer displacements than the number of columns, and had previously specified displacements on another .CD control word, those values remain in effect for any columns not respecified. Whenever a .CD control word is used, there must be available (either on the control line or previously specified) displacements for each column. If you specify .CD n without specifying any displacements, the arbitrary values 0, 46, 92, 0, 0, 0, 0, 0, 0 are used.

3. If you use several different column formats in a document you can create symbolic names (with the .SE control word) or macros (with the .DM control word) to establish column definitions, column lengths, and so on. If you use a single one-column format and a single multiple-column format, you can switch back and forth using the .SC (SINGLE-COLUMN-MODE) and .MC (MULTICOLUMN-MODE) control words.

4. This control word is not allowed in a keep.

Example

This book has some parts formatted in two columns and some parts done in a single column. In either case, the first column starts in the leftmost position. The first definition of the columns for this book looks like this:

```
.cd 2 0 46
```

This control word specifies that there are to be two columns, the first starting in position 1 (displacement 0), and the second in position 47 (displacement 46). Now, to switch to single column mode, this control word is sufficient:

```
.cd 1
```

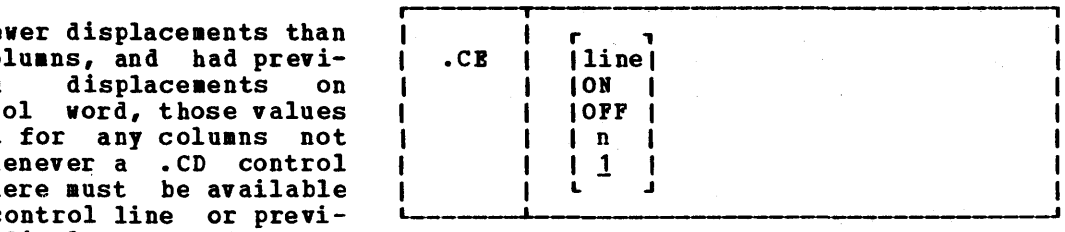
And to switch back to two column mode:

```
.cd 2
```

The displacements, 0 and 46, remain in effect until you change them.
***

.CE (CENTER)

Use the CENTER control word to center output lines between the margins. The format of the .CE control word is:



where:

line is the line that is to be centered.

ON specifies that subsequent text lines are to be centered.

OFF terminates centering mode if it was ON, or if n has been specified and has not been exhausted.

n specifies the number of lines to be centered. If omitted, 1 is assumed. If .CE n is specified when .CE ON is in effect, centering is turned off when n lines have been centered, or when .CE OFF is encountered.

Usage Notes

1. The line(s) are centered starting at the current left margin (including indent and offset values in effect); leading blanks are considered part of the line's length. When centering is in effect, no formatting is done on the line. That is, the line is centered as it stands, and it is not filled from other input lines or justified. If a tab character appears in the line to be centered, the tab is resolved before the line is centered.

2. This control word acts as a break.

3. If the line to be centered is longer than the current column length, it is truncated, and the excess is used on a

useful when attempting to locate a specific region of the file during editing.

2. If you want an entire line to be ignored, and not scanned for control word separators, you can use another form of comment. Any line that begins with ".*" is ignored. ".*" is not considered to be a control word, but .CM is.

3. The .CM control word can be used in conjunction with the .IF control word to enable or disable strings of control words. For an example of how to do this, see the discussion of the .IF control word.

Example

```
.cm   Remember to change the date.
```

The line above is seen when examining an unformatted listing of the SCRIPT file, and it reminds you to update the date used in the text.
***

.CO (CONCATENATE-MODE)

Use the CONCATENATE-MODE control word to cancel or restore concatenation of input lines and truncation at the current column length. The format of the .CO control word is:

where:

ON restores concatenation of input lines. ON is the initial setting, as well as the default value.

OFF cancels concatenation of input lines. If justification is still in effect, .CO OFF results in each line being padded with blanks to the column length.

Usage Notes

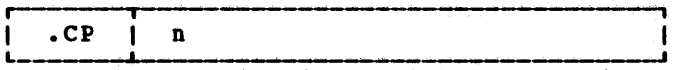
1. When SCRIPT is concatenating text, output lines are formed by shifting words to or from the next input line. The resulting line is as close to the specified column length as possible without exceeding it or splitting a word; if no-justify is in effect, output resembles normal typist output or the MT/ST. Concatenation is the normal mode of operation for the SCRIPT command.

   When SCRIPT is not concatenating text, there is a one-to-one correspondence between the words on the input and output lines. If SCRIPT is still justifying text, each line that is less than the column length is padded with blanks to fill the column.

2. This control word acts as a break.
***

.CP (CONDITIONAL-PAGE-EJECT)

The CONDITIONAL-PAGE-EJECT control word causes a page eject to occur if fewer than the specified number of lines remain in the current column[1]. The format of the .CP control word is:



where:

n is the number of lines that must remain on the current page for additional lines to be processed without a page eject. If it is omitted, the control word is ignored.

Usage Notes

1. The .CP control word is especially useful (1) before an illustration, to guarantee that sufficient space remains on the current page to accommodate its length, and (2) preceding a section heading to eliminate the possibility of a heading occurring as the last line of a page.

2. This control word is not allowed in a keep.

---

[1] If you are using multiple column processing, and a .CP is encountered when there are fewer than the specified number of lines remaining in the column, a page eject occurs, even if this is not the last column on the page.

```
.cs 1 on
256
.cs 1 off
.cs 2 on
1000
.cs 2 off
entries in a MACLIB file.
```

Since only conditional section code 2 is to be included, the generated output line is "In this version of the system there can only be 1000 entries in a MACLIB file".
***

.CW (CONTROL-WORD-SEPARATOR)

The CONTROL-WORD-SEPARATOR control word allows you to change the symbol used to separate multiple control words on a single line. The default control word separator symbol is the semicolon (;) character. The format of the .CW control word is:

```
┌──────┬──────────────────────────────┐
│ .CW  │ s                            │
└──────┴──────────────────────────────┘
```

where:

s specifies the character to be used as the "control word separator" symbol. Any character may be used. If the character 's' is omitted, no character is assigned as the control word separator, and therefore you cannot have more than one control word on a line.

Usage Notes

1. When the CONTROL-WORD-SEPARATOR control word is processed, the default control word separator (;) is reset. It may be necessary to change the control word separator symbol if it is inconvenient to type the default symbol, or if the default symbol is used as part of a control word operand (such as part of a symbol specification).

2. If a symbol value begins with the control word separator, the rest of the symbol value is treated as though it occupied the first position of the line.

3. Control word separators are recognized on a .CM (COMMENT) line, but not on a ".*" line.

4. The following control words must begin in column 1 and may not be placed after a control word separator:

```
.cs n off
.di off
.li off
...label
```

When SCRIPT is ignoring a conditional section, preparing a delay imbed, reading literal lines, or searching for a label no control word processing is done. Each input record is checked in column 1 for the presence of the control word that ends the special processing mode.

5. Control words that accept text data (for example, .US or .CE), should not contain the current control word separator.

Examples

1. Simple change:

```
.cw ,
.sp 2,.of 5,This section discusses ...
```

The above line is equivalent to the lines:

```
.sp 2
.of 5
This section discusses ...
```

2. Temporary cancelation to get the separator character into a symbol value:

```
.cw
.se 2col = ';.cd 2 0 46;.cl 43;'
.se 1col = ';.cd 1;.cl 89;'
.cw ;
```

In the sequence above, the control word separator is temporarily canceled so that the regular separator (;) can be used as part of the .SE (SET-SYMBOL) control line. Since the symbols 2col and 1col contain multiple control words, they can now be used instead of the actual control words involved. Since the control words are in a symbol that begins with the control word separator, they can be recognized as control words even if the symbol is encountered in the middle of a line. Since the symbols end with control word separators, the effective next line can be concatenated to the symbol name. With the symbols 2col and 1col set as shown, the line:

```
This is a line.&2col.Now start 2 columns.
```

Has the same effect as the sequence:

```
This is a line.
.cd 2 0 46
.cl 43
Now start 2 columns.
```
***

Usage Notes

1. The DELAY-IMBED control word is especially useful for positioning diagrams and tables. The next n lines of the current SCRIPT file are saved in a special temporary file called IKSUT1 SCRIPT. When the top of the next output page is reached, this temporary file is imbedded and processed by SCRIPT. After the inclusion of the saved lines, normal processing resumes.

2. This control word acts as a break.

3. An automatic page eject is not performed at the end of the inclusion. If you want SCRIPT to resume normal processing on a new page, you should end the delayed section with a .PA control word.

4. The .DI OFF control word must begin in column 1, not after a control word separator. When SCRIPT is processing a delay-imbed it is not processing input lines except to look for .DI OFF in column 1.

Examples

1. To delay the inclusion of one line:

```
.di .pa
```

At the end of the current page, a blank page, except for top and bottom titles, is generated. Output resumes on the page after the blank page.

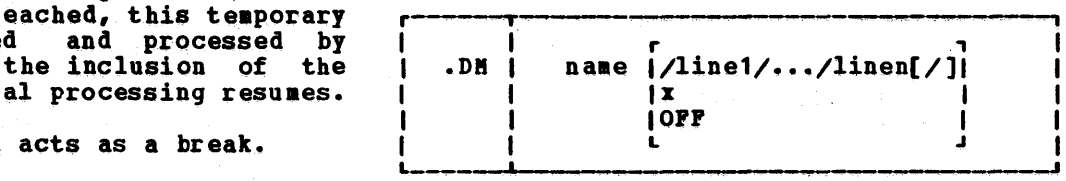
2. To include a figure at the top of the next page:

```
.di 3
.sp 2
.im figure5
.sp 5
```

The current page is processed as if the .DI and the three following lines had not existed. At the top of the next page, the three lines are processed. This results in spacing two lines, imbedding the SCRIPT file named FIGURE5, followed by spacing five more lines. If there is sufficient room remaining on the page, normal SCRIPT output resumes immediately.
***

.DM (DEFINE-MACRO)

Use the DEFINE-MACRO control word to establish macro definitions for sequences of SCRIPT control words. SCRIPT macros are invoked by preceding them with periods, as SCRIPT control words. No macro substitution is performed unless the .MS (MACRO-SUBSTITUTION) control word has been entered. The format of the .DM control word is:



where:

name is the symbolic name you want to assign to the macro, so that you can invoke it with the control line:

```
.name
```

It may contain a maximum of 10 nonblank characters.

/ is any delimiting character, used to separate the lines in the macro. The final delimiter may be omitted.

line is any SCRIPT control word line or line of data that you want to include in the macro definition. It may contain symbolic names, or any of the special macro variables &*, or &1 through &9 (see Usage Note 1).

x indicates that you want the current value of a macro assigned to the symbol &x. x may be any alphameric character.

OFF cancels a macro definition.

Usage Notes

1. The following symbols have special meanings when used to define SCRIPT macros with the .DM control word:

   &*: is the line passed to the macro when it is invoked. Thus, if the macro defined with the control line:

```
.dm typit /.ty ***/.ty &*/.ty ***
```

   is invoked with the line

```
.typit Hello!
```

   then the symbol &* has the value "Hello!". The processing of this macro results in the lines:

.EB (EVEN-PAGE-BOTTOM-TITLE)

The EVEN-PAGE-BOTTOM-TITLE control word saves a specified title line in a storage buffer for possible future use. This title may be used at the bottom of the current page, if it is even-numbered, and each subsequent even-numbered output page. The format of the .EB control word is:

```
| .EB | [n] /part1/part2/part3/ |
```

where:

n is the number of the bottom title line to be set. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even bottom title 1" sets the same storage buffer as "even top title 6." See the discussion of the .FS (FOOTING-SPACE) control word for information on how to allocate space on your output page for bottom titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Usage Notes

1. Titles may be printed at the bottom of the current page, if it is even-numbered, and each subsequent even-numbered output page, if space has been allocated for it using the .FS control word. The specific location of the bottom titles on the page is controlled by the .BM (BOTTOM-MARGIN) and .FM (FOOTING-MARGIN) control words; the number of bottom titles to be used on each page is controlled by the .FS control word. Any even title may be changed by including another .BT or .EB control word later in the SCRIPT file.

2. This control word is very useful for producing a manuscript that is to be printed in book style using both sides of a page of paper. By convention, when the book is opened, the even-numbered page is on the left. This distinction is important if, for example, it is desired to print the page number always on the outer edge of the page, which is on the left for even-numbered pages and on the right for odd-numbered pages.

***

.EF (END-OF-FILE)

The END-OF-FILE control word simulates the end of the current file. The format of the .EF control word is:

```
| .EF | [CLOSE] |
```

where:

CLOSE tells SCRIPT not to hold your place in the current file, but to close it, so that the next time the file is imbedded, SCRIPT begins processing at the top of the file, not at the line following the .EF control word.

Usage Notes

1. The END-OF-FILE control word causes an end of file condition to be simulated on the current input file. If the current input file is not an imbedded file (see the IMBED control word), all processing is terminated. If the current input file has been imbedded, the .EF control word causes input processing to continue in the outer file. In this latter case, SCRIPT remembers the position of the .EF control word; if the file is imbedded again, then SCRIPT begins reading at the line following the EF control word instead of the beginning of the file (unless the CLOSE operand is used).

2. The .EF control word, in conjunction with the IMBED, APPEND, and QUIT control words, provides an easy and flexible mechanism for producing tables, as demonstrated in the example below.

Example

In this example, a table is generated using two files. One file contains a single line

This allows you to have your output only on even-numbered pages, while leaving odd-numbered pages blank for artwork to be added later. The .EP ON mode is canceled by either .EP OFF, .OP ON, or .OP OFF.

2. This control word acts as a break It is not allowed in a keep.

Example

```
.ep
```

The rest of the current page is skipped. The page on which text resumes printing is even-numbered even if it is necessary to generate an empty odd-numbered page in between.
***

.ET (EVEN-PAGE-TOP-TITLE)

The EVEN-PAGE-TOP-TITLE control word saves a specified title line in a storage buffer for possible future use. This title may be used at the top of all subsequent even-numbered output pages. The format of the .EP control word is:

```
┌──────┬──────────────────────────────────┐
│ .ET  │ [n] /part1/part2/part3/          │
└──────┴──────────────────────────────────┘
```

where:

n is the number of the top title line to be set for even-numbered pages. The number may be from 1 to 6, and if it is omitted, 1 is assumed. The six possible title lines are the same for top titles and bottom titles. Bottom titles are numbered from bottom to top; top titles are numbered from top to bottom. Therefore, "even top title 1" sets the same storage buffer as even bottom title 6. See the discussion of the .HS (HEADING-SPACE) control word for information on how to allocate space on your output page for top titles.

part1 is the portion of the title to be left justified.

part2 is the portion of the title to be centered between the left and right margins.

part3 is the portion of the title to be right justified.

/ is any character that does not appear in part1, part2, or part3.

Usage Notes

1. Titles may be printed at the top of each subsequent even-numbered output page, if space has been allocated for them using the .HS control word. The specific location of the top titles on the page is controlled by the .TM (TOP-MARGIN) and .HM (HEADING-MARGIN) control words; the number of top titles to be used on each page is controlled by the .HS control word. Any even title may be changed by including another .ET or a .TT (TOP-TITLE) control word later in the SCRIPT file.

***

.EZ (EASYSCRIPT)

Use the EasySCRIPT control word to initialize the GML tags used by the EasySCRIPT processor. This control word is valid only if the SCRIPT module has been generated with the EasySCRIPT facility. The format of the .EZ control word is:

```
┌──────┬──────────────────────────────────┐
│ .EZ  │ { ON [lastDewey]}                │
│      │ { OFF            }               │
│      │ { tagval line    }               │
└──────┴──────────────────────────────────┘
```

where:

ON initializes the EasySCRIPT tags that provide the EasySCRIPT numbering, paragraphing, and heading functions.

OFF cancels the EasySCRIPT tags, so that they are not recognized by SCRIPT.

tagval is an EasySCRIPT tag without the preceding ampersand (&). You can call out an EasySCRIPT tag without having set EasySCRIPT on by using the .EZ tagval control word.

line is an input line of data.

lastDewey
is the Dewey decimal number of the last heading that would have been used. EasySCRIPT uses this number to set the counter it uses for numbered headings. If not specified, 0.0.0.0 is assumed. If you specify &xref, EasySCRIPT resumes numbering where it left off when .EZ OFF was last processed.

.FN (FOOTNOTE)

Use the FOOTNOTE control word to set aside up to ten lines[1] of formatted output text to be positioned at the bottom of the current column, if possible, or at the bottom of the next column. The format of this control word is:

```
.FN  {ON }
     {OFF}
```

where:

ON marks the beginning of the material in the footnote.

OFF marks the end of the footnote material.

Usage Notes

1. FOOTNOTE ON starts a footnote. All lines until the subsequent FOOTNOTE OFF control are put in the footnote. If .FN OFF is encountered when no footnote is in process, it is ignored. If the maximum number of lines is exceeded, the footnote is ended just as though a .FN OFF control word had been encountered, and a warning message is typed. The rest of the material is treated as ordinary text. (The real .FN OFF control word is ignored, since no footnote is then in process.)

3. The first footnote in a column is automatically started with a three line package consisting of two spaces and then a 15-character cutoff rule. The cutoff rule is composed of hyphens (-), but you may translate them to another character if you wish. Note that the material in the footnote is translated according to the translations in effect at the time the footnote is formatted, not when it is actually printed.

4. The .FN control word does not act as a break. The footnote is considered to fit in this column if the number of lines left is sufficient to accommodate the footnote itself, the three line cutoff package, and one more line for the callout. (In other words, there must be four more lines than the size of the footnote left in this column.) To ensure that your footnote and the callout appear in the same column, put the footnote itself before the callout. Since the footnote does not cause a break, the sentence containing the callout may be interrupted for the footnote itself without disrupting the formatted output.

5. The FOOTNOTE control word uses the same routines as the KEEP control word, so a keep and a footnote may not be in process at the same time.[2] A footnote in process is terminated by any disallowed control word or by a control word that starts a new keep, footnote, or headnote.

6. A footnote does not have any automatic offset. You must include an OFFSET (.OF) control word if you want the footnote offset. If an indent or offset is in effect when you begin a footnote, you should cancel and restore it within the footnote.

Example

The first footnote in this section was done with the following sequence:

```
.if SYSOUT eq PRINT .tr 1 b1
Use the FOOTNOTE control word
to set aside
.fn on
.of 1
1You may have more than one footnote
in a column, provided that
each one occupies no more than
ten formatted lines.
.of 0
.fn off
up to ten lines1 of
formatted output text...
.tr 1 1
```

Notice that the callout to the footnote was placed after the actual footnote. Now, if the footnote did not fit in the current column, a column eject would be done immediately, and the callout would still be in the same column as the footnote to which it refers.
***

[1] You may have more than one footnote in a column, provided that each one occupies no more than ten formatted lines.

[2] For this reason, the same control words that are disallowed in a keep are also disallowed in a footnote.

setting the bottom titles to null (.BT ////) because SCRIPT does not have to process any titles to determine that none are wanted.

Example

If you want three running bottom titles in your document, you could use the following sequence:

```
.bt 3 /Chapter 4//&/
.bt 2 ////
.bt 1 $$&SYSMONTH./&SYSYEAR.$$
```

At this point, only bottom title 1, the one nearest the bottom of the page, is used on formatted output pages because the default footing space of 1 is still in effect. Now that the three title lines have been saved, the following control word causes SCRIPT to print all three:

```
.fs 3
```

***

.GO (GOTO)

The GOTO control word causes SCRIPT to jump to another part of the SCRIPT input file. The format of the .GO control word is:

.GO	label

where:

label is the name of a line that has been set elsewhere in this SCRIPT file using the "..." (SET-LABEL) control word.

Usage Notes

1. Use the GOTO control word to branch to another place in your SCRIPT file. If the label designated on the GOTO control word is not defined elsewhere in the file, an error message is generated.

2. This control word does not cause an automatic break. The input line preceding the GOTO control and the line at the label designated in the GOTO control word are processed as though they were two sequential lines from the SCRIPT file.

3. Every GOTO control word must refer to a label that has been defined with the ... control word; but you may have more than one GOTO referring to the same label.

4. GOTO is particularly useful when performed conditionally as the subject of an IF statement. See the discussion of the .IF control word.

Example

Suppose you had a SCRIPT file that was designed to recognize the variable SYSVAR5 (see "Section 2. The SCRIPT Command"). In this example, if SYSVAR5 is set to SMALL, you want SCRIPT to format the output at 36 lines per page and 42 characters per line. Otherwise, the default values of 66 lines per page and 60 characters per line are to be used. This could be done with the following control words:

```
.if &SYSVAR5 ne SMALL .go default
.pl 36
.ll 42
...default
(etc.)
```

***

.H0 - .H6 (HEAD-LEVEL-N)

The control words .H0 through .H6 automatically format topic headings in SCRIPT output. The definition of a particular head level may also result in an entry in the table of contents for that heading. The format of the .Hn control word is:

.Hn	[text]

where:

n is the number of the head level from 0 to 6.

text is the data to be formatted as a subject head and optionally placed in the table of contents.

Usage Notes

1. The .Hn control words provide several automated functions for you. They can provide a topic heading that is underscored or capitalized with a specified number of skips before it and line spaces after it. They can cause the

Usage Notes

1. The heading line is placed a specified number of lines above the top margin. If no .HM (HEADING-MARGIN) control word is included in the file, the default value is 2.

   The heading margin specified must always be small enough so that it plus the heading space can be accommodated within the current top margin.

2. This control word acts as a break.

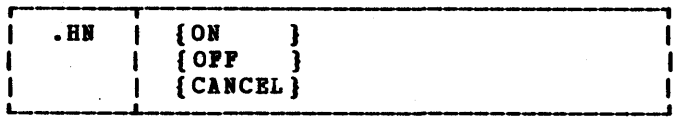
Example

```
.hm 3
```

Three lines are left between the heading lines and the first line of text. If the default top margin of 6 is in effect, the last top title is printed two lines from the top of the page, followed by three more blank lines (the heading margin), and then the text.
XXX

.HN (HEADNOTE)

Use the HEADNOTE control word when you want to set aside up to ten lines of formatted output text to be positioned at the top of the next page and all subsequent pages until canceled. The format of the .HN control word is:



where:

ON tells SCRIPT to start putting formatted text in a headnote.

OFF marks the end of the material to go in the headnote. If .HN OFF is encountered when no headnote is in process, it is ignored.

CANCEL clears the active headnote so that it will no longer print at the top of the text area of every output page.

Usage Notes

1. The .HN ON control word starts a headnote. All lines until the subsequent .HN OFF are put in the headnote. If the maximum number of lines is exceeded, the headnote is ended just as though a .HN OFF control word had been encountered, and a warning message is displayed. The rest of the material is treated as ordinary text. (The real .HN OFF is ignored, since no headnote is then in process.) The headnote is placed at the top of each subsequent page until canceled with a .HN CANCEL control word.

2. The maximum size of each headnote is 10 lines. If the material in the headnote occupies more than 10 formatted output lines, the headnote is terminated, and a warning message is displayed.

3. The HEADNOTE control word does not cause a break.

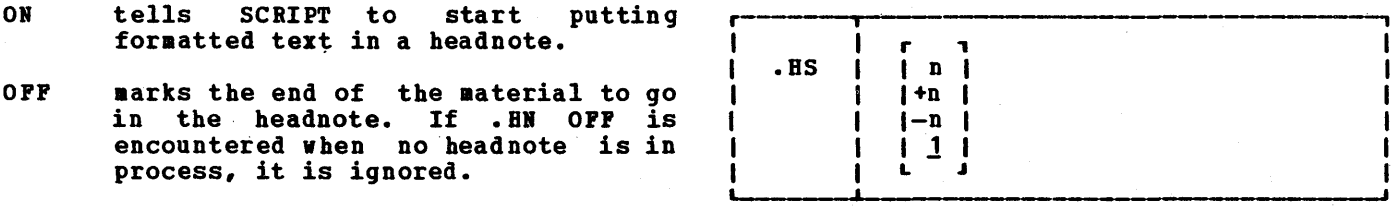
4. The HEADNOTE control word uses the same routines as the KEEP control word, so a keep and a headnote may not be in process at the same time. A headnote in process is terminated by any disallowed control word or by a control word that starts a new keep, footnote, or headnote.

5. Floating and delayed keeps are processed before headnotes at the top of a new page.

XXX

.HS (HEADING-SPACE)

The HEADING-SPACE control word allocates space from the top margin area for headings or titles. The format of the .HS control word is:



where:

n is the number of lines of top title you want on each subsequent output page. This number may be from 0 to 6. If no number is given, 1 is assumed. If +n or -n is specified, the current value for

off. This control word is valid only if the SCRIPT module has been generated with the hyphenation facility. The format of the .HY control word is:

```
.HY  {ON [filename]  }
     {OFF            }
     {SUP            }
     {SET {THRESH} n }
     {    {MINPT }   }
```

where:

ON — begins automatic hyphenation of SCRIPT output lines.

filename — is the name of an optional exception dictionary to be searched before the standard SCRIPT XDICT is searched (see Usage Note 2).

OFF — causes hyphenation to be turned off.

SUP — causes hyphenation to be suppressed until the next space. If hyphenation is off, then SUP does nothing, but if it is on, then SUP turns it off temporarily. It automatically turns on again the next time a line space is generated (as a result of .SP, .SK, head-level control words, or a page eject). This allows you to suppress hyphenation at the end of a paragraph without having to turn it off and then on explicitly.

SET — indicates that you are going to override the default hypenation values, THRESH and MINPT.

THRESH n — is a positive number indicating the hyphenation threshold. When SCRIPT is formatting a line, at least n spaces must remain before SCRIPT attempts hyphenation. The initial value of THRESH is 7.

MINPT n — is a positive number indicating the minimum hyphenation you want to allow. The initial value of MINPT is 4, which means that the first hyphenation point in a word must be at least four characters beyond the beginning of the word.

Usage Notes

1. When SCRIPT is formatting text, and the next word does not fit on the line, SCRIPT ordinarily moves the word onto the next output line. When hyphenation is in effect, SCRIPT attempts to break the word into two pieces: the longest piece that can fit on the line, and the remainder. If there are at least "THRESH" spaces left on the line, the word is examined by the hyphenator, which returns a number that is less than the number of remaining spaces. If this number is greater than MINPT," SCRIPT breaks the word after that number of letters.

2. The exception dictionary is a file called SCRIPT XDICT. This file contains words that may be incorrectly hyphenated by the algorithm that decides where to hyphenate words. When hyphenation is ON, and a word does not fit in the current line, this exception dictionary is searched for that word; if no match is found, the algorithmic hyphenator is used.

   You may specify the name of a private exception dictionary that is to be searched before the SCRIPT XDICT. To create or modify an exception dictionary, you must use the HYPEDIT command, which is described under "Automatic Hyphenation" in Section 1.

xxx

.IF (IF)

The IF control word allows a SCRIPT input line to be processed conditionally. The format of the .IF control word is:

```
.IF  {comp1    test   comp2    targ}

     {SYSPAGE  test  {EVEN}    targ}
     {                {ODD }      }

     {SYSOUT   test  {PRINT}   targ}
     {                {TERM }     }
```

where:

comp1 — is any word or number of eight or fewer characters to be used as the first comparand. This comparand may be the value of a set symbol.

comp2 — is any word or number of eight or fewer characters to be used as the second comparand. It too may be the value of a set symbol.

test — is a one or two character code that

the .CW control word is not processed, and the remaining line is treated as a comment. If the condition is true, the .CW is processed, and the new control word separator is recognized to allow the remaining line to be broken up into four active control words.

4. If there is a possibility that one of the comparands may be a null symbol, another trick should be used:

```
.if X&answer eq Xyes (do this)
```

Now, if the symbol "answer" is null, the line will become:

```
.if X eq Xyes (do this)
```

Otherwise, if you had not included the Xs, a null symbol could shift the fields over like this:

```
.if eq yes (do this)
```

and "yes" is not a recognized condition. Note that the symbol is null only if so set by the .SE or .RV control words. In SCRIPT, a symbol that has not been set is not null, as is true of an EXEC file in CMS. The only exceptions to this are the special symbols &0 through &9.
***

.IL (INDENT-LINE)

Use the INDENT-LINE control word to indent the next line only a specified number of characters. The format of the .IL control word is:

where:

n specifies the number of characters to shift the next line from the current margin. +n specifies that text is shifted to the right, and -n shifts text to the left.

Usage Notes

1. The .IL control word provides a way to indent only the next output line. The line is shifted to the right or the left of the current margin (which includes any indent or offset values in effect).

2. This control word acts as a break.

3. The .IL control word and the .UN (UNDENT) control word are opposites; thus, the control words UN 5 and .IL -5 are equivalent.

4. The .IL control word may be useful for beginning new paragraphs.

5. When successive .IL and .UN control words are encountered without intervening text, or when positive or negative increments are specified for .IL control words entered without intervening text, the indent amount is modified accordingly. Thus the lines

```
.il 4
.il +6
```

result in the next line being indented 10 characters.

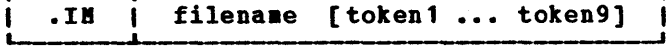
Example

```
.il 3
```

This line is preceded by the control word INDENT-LINE 3.
***

.IM (IMBED)

Use the IMBED control word to insert the contents of a specified file into the current file. Processing continues as though the material in the imbedded file were part of the current file. The format of the .IM control word is:



where:

filename specifies the filename of the file to be copied into the output. The filetype of the file must be SCRIPT.

tokens are positional values that may be passed to the imbedded file. The first token (word) becomes the value of the symbol &1, the second token becomes the value of the symbol &2, and so forth. The

.JU (JUSTIFY-MODE)

Use the JUSTIFY-MODE control word to cancel or resume right justification of output lines. The format of the .JU control word is:

```
┌──────┬──────────────────────────────────┐
│      │ ┌   ┐                            │
│ .JU  │ │ON │                            │
│      │ │OFF│                            │
│      │ └   ┘                            │
└──────┴──────────────────────────────────┘
```

where:

ON restores right justification of output lines. If neither ON nor OFF is specified, ON is assumed.

OFF cancels justification of output lines. If concatenation is still in effect, .JU OFF results in ragged right output.

Usage Notes

1. When SCRIPT is justifying output, lines are padded with extra blanks to the length of the column. If SCRIPT is also concatenating text, the concatenation process occurs before justification.

   When justification is stopped, but SCRIPT is still concatenating text, lines are formed that approach the current column length but are not forced to the exact length. The resulting lines resemble the output usually produced by a typist or an MT/ST (Magnetic Tape/Selectric Typewriter).

2. Justification may also be canceled by the .FO OFF (FORMAT-MODE) control word. The .FO OFF control word also cancels concatenation.

3. This control word acts as a break.

Example

```
1.  .ju off

These lines are being concatenated by
SCRIPT, but no blanks are added to fill the
lines to the column length.

2.  .ju

Output from this point on in the file is
padded to produce an even right margin on
the output page, as long as the input lines
do not exceed the column length.
```

***

.KP (KEEP)

The KEEP control word allows you to designate blocks of text that must be kept together without being separated by a column eject or a page eject. The format of the .KP control word is:

```
┌──────┬──────────────────────────────────┐
│ .KP  │ {ON   }                          │
│      │ {FLOAT}                          │
│      │ {DELAY}                          │
│      │ {OFF  }                          │
└──────┴──────────────────────────────────┘
```

where:

ON starts a regular keep. A regular keep is put in this column if it will fit, and otherwise an immediate column eject is done. It is similar to a conditional column eject, except that when you use .CC you tell SCRIPT how many output lines must remain in the column, but in a keep you tell SCRIPT how much input data must fit in the column.

FLOAT starts a floating keep. A floating keep is put in this column if it will fit; otherwise it goes at the top of the next column. Text following the floating keep is formatted into the rest of this column.

DELAY starts a delayed keep. A delayed keep is always printed at the top of the next column, even if there is room for it in this column.

OFF marks the end of a regular, floating, or delayed keep.

Usage Notes

1. The KEEP control word delimits blocks of text that must be kept together. A keep is started with .KP ON, .KP FLOAT, or .KP DELAY. It is ended with .KP OFF, starting a new keep, footnote, or headnote, or when a disallowed control word is encountered. Any control word that changes the number of lines that were left in the column before the keep is disallowed while a keep is in process. These are: .BM, .CB, .CC .CD, .CP, .EP, .MC, .OP, .PA, .PL, .PN, .SC, and .TC. Some head level control words

Example

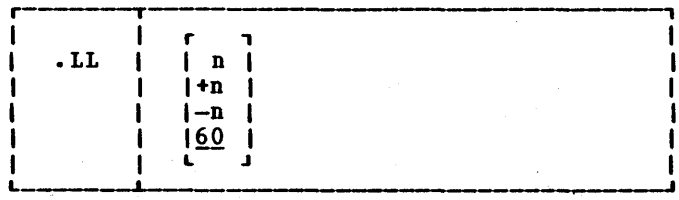
If a text line must begin with a period:

```
.li
..... Male    ..... Female  (check one)
```

The line "..... Male, etc." starts in column 1. Ordinarily, SCRIPT would attempt to interpret that line as a control word which would result in an error condition. The LITERAL control word inhibits interpreting the line as a control word.
***

.LL (LINE-LENGTH)

The LINE-LENGTH control word specifies the character width of top and bottom titles. It also changes the value of the column length, which governs the width of text lines, if the latter has never been set explicitly. The format of the .LL control word is:



where:

n specifies an output line length not greater than 132 characters. If no value is specified for n, the default value of 60 is used.

Usage Notes

1. The LINE-LENGTH control word sets the total length for output lines from the left margin to the right margin. The .LL value governs the length of title lines. Text lines are always governed by the .CL (COLUMN-LENGTH) control word, but if the column length has never been explicitly set, it has the same value as the line length. See the discussion of the .CL control word.

2. This control word acts as a break.

***

.LS (LINE-SPACING)

Use the LINE-SPACING control word to specify multiple-spacing of output text lines. This control word is a generalization of the .SS (SINGLE-SPACE-MODE) and .DS (DOUBLE-SPACE-MODE) control words. The format of the .LS control word is:

where:

n specifies the number of blank lines to be inserted after each standard text line.

Usage Notes

1. This control word acts as a break.

2. The .SS control word is identical to .LS 0, and the .DS control word is identical to .LS 1. When line spacing is in effect, the .SK (SKIP-LINES) and .SP (SPACE-LINES) control words act like multiple blank lines to which additional blank lines are inserted. For example, if the line spacing amount is 2 (triple-spacing), then .SP or .SK would result in three blank lines, .SP 2 results in six blank lines, and so on. However, if the .SP or .SK control word indicates "absolute" spaces, then the space count is not multiplied.

3. This control word overrides previous .LS, .SS, and .DS control words.

Example

```
.ls 2
```

Subsequent output is "triple-spaced" such that each text line is separated by two blank lines.

***

.MC (MULTICOLUMN-MODE)

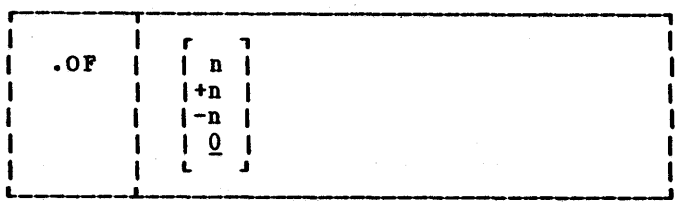
The MULTICOLUMN-MODE control word restores multiple column processing after it has

2. This control word acts as a break.

***

.OF (OFFSET)

Use the OFFSET control word to indent all but the first line of a block of text. The format of the .OF control word is:



where:

n specifies the number of spaces to be indented after the next line is formatted. If omitted, 0 is assumed, and indention reverts to the original margin setting. If you use +n or -n, the current offset value is incremented or decremented the specified amount, and a new offset is started.

Usage Notes

1. An OFFSET control word does not take effect until after the next line is formatted. The indention remains in effect until a .IN (INDENT) control word or another OFFSET control word is encountered.

   The .OF control may be used within a section which is also indented with the .IN control. Note that .IN settings take precedence over .OF, however, and any .IN request causes a previous offset to be cleared.

   If you want to start a new section with the same offset as the previous section, you need only repeat the .OF n request.

2. This control word acts as a break.

3. The .IL (INDENT-LINE) and the .UN (UNDENT) control words can be used to shift only the next lines to the left or right of the current margin.

4. Tabs should be used whenever possible to format numbered or bulleted lists, to ensure that the first text word on the line is even with subsequent offset lines. The items in this "Usage Notes" section are created using a tab setting of 4 and an offset of 4.

Examples

1. Starting an offset:

```
.of 10
```

The line immediately following the .OF control word is printed at the current left margin. All lines thereafter (until the next indent or offset request) are indented ten spaces from the current margin setting. These two examples were processed with OFFSET control words in the positions shown.

2. Ending an offset:

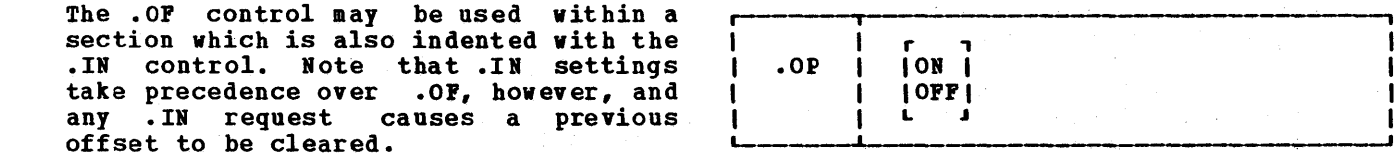
```
.of
```

The effect of any previous .OF request is canceled, and all output after the next line continues at the current left margin setting.

***

.OP (ODD-PAGE-EJECT)

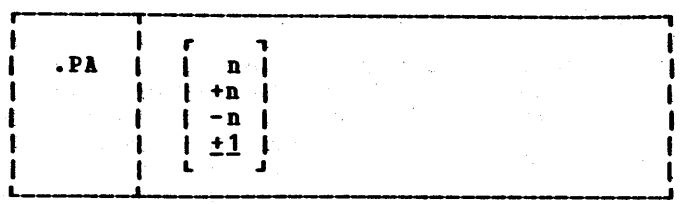
Use the ODD-PAGE-EJECT control word to cause either one or two page ejects, such that the new page is odd-numbered regardless of whether the current page is odd- or even-numbered. The format of the .OP control word is:



where:

ON specifies that subsequent text is to be printed only on odd-numbered pages. Even-numbered pages are left blank, except for top and bottom titles, if any.

OFF resumes processing so that text appears on even- and odd-numbered pages.



where:

n specifies the page number of the next page. If n is not specified, sequential page numbering is assumed (that is, the next page number is one greater than the current page number).

+n specifies that the next page should have a number that is equal to the normal next sequential page number plus n.

-n specifies that the next page should have a page number that is equal to the next sequential page number minus n. If subtracting n from the next page number yields a negative number, an error message is typed, and the control word is ignored.

Usage Notes

1. Whenever a PAGE-EJECT control word is encountered, the rest of the current page is skipped after printing any text lines accumulated thus far. Output advances to the next page, after printing any current bottom titles. The currently active top titles are inserted, and formatting resumes with the line following the .PA control word. If you use the STOP option of the SCRIPT command, SCRIPT waits for you to enter a null line (with the Return or Enter key) before starting the new page.

2. This control word acts as a break. It is not allowed in a keep.

3. If you want to add to or change the top titles to appear on the new page, the title control words must be processed before the .PA control word. This also applies to any control words that change the format of the top titles, including line length control words.

   Any bottom title control words that appear before the .PA control word take effect on the current page.

Examples

1. To start the next sequential page:

```
.pa
```

The rest of the current page is skipped. The top titles and page number are put in the top margin of the next page, and output resumes.

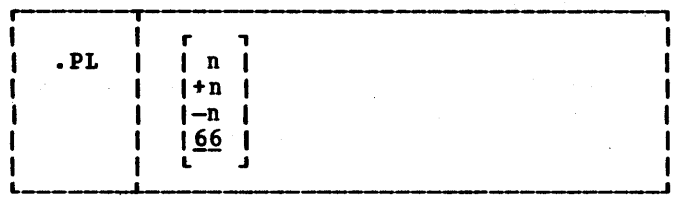
2. To repeat a page number:

```
.pa -1
```

The new page will have the same page number as the preceding page. The calculation is done after establishing the next sequential page number.
XXX

.PL (PAGE-LENGTH)

The PAGE-LENGTH control word specifies the length of output pages in lines. The value specified overrides the standard page length of 66 lines. The format of the .PL control word is:



where:

n specifies the length of output pages in lines. If no value is specified for n, the default value of 66 is used. This number should be the same as the physical number of print lines on each page of paper being used. However, when formatting in printer format, it may be different, as explained below.

Usage Notes

1. The PAGE-LENGTH control word allows varying paper sizes to be used for output. This is the correct size of standard typewriter paper for terminals typing six lines per inch. Page length may be changed anywhere in a file, with the change effective on the page on which the control word is encountered, if possible, or on the next page.

2. The .PN OFF and .PN OFFNO control words suppress the default top title "PAGE &." If you use the top or bottom title control words and include an &, only the page number and not the text is suppressed.

3. The .PN OFF effect can also be accomplished by redefining the top title without any page number symbol, or by setting the heading space to zero.

4. If both FRAC and ROMAN are in effect, the page number that is printed consists of only the fractional portion of the number in lowercase roman numerals. If ROMAN is in effect, no prefixes (as specified with .PN PREF) are printed.

5. Table of contents entries generated by the .Hn (HEAD-LEVEL-n) control words or the .PT (PUT-TABLE-OF-CONTENTS) control words show the page numbers in the same format they appear on the page, that is, if a prefix is used, it is shown in the table of contents; if roman numbers are in effect, the contents entry has a roman numeral, and so on.

Examples

1. .pn off

The internal page count continues to be incremented for each page printed.

2. .pn offno

No page numbers appear on SCRIPT output, and the internal page count remains at its current setting without further incrementing.

3. .pn on

Page numbering on SCRIPT output resumes using the current internal page count; this count is incremented for each page printed.

4. .pn roman

The page number in the title at the bottom of this page appears as a roman numeral. Notice that the page number also appears as a roman numeral in the table of contents in the front of the book.

The control word

.pn arabic

restores arabic numbering on the next page.
***

.PP (PARAGRAPH-START)

Use the PARAGRAPH-START control word to start a new paragraph. The format of the .PP control word is:

.PP	[line]

where:

line is the text that begins a new paragraph. If line is omitted, the text from the next input line after the .PP control word begins the new paragraph.

Usage Notes

1. When the .PP control word is encountered, a break occurs, a skip is generated, and the next line of text is indented three characters to the right of the current margin. The .PP control word is equivalent to the control words:

```
.sk
.il +3
```

If these values are not satisfactory for your paragraph formatting, you can redefine the .PP control word as a SCRIPT macro.

Example

```
.pp This line begins with
```

This line begins with a .PP control word.
***

.PS (PAGE-NUMBER-SYMBOL)

The PAGE-NUMBER-SYMBOL control word allows you to change the special page number symbol used in top and bottom title control words. The default page number symbol is the ampersand (&) character. The format of the .PS control word is:

.PS	[s]

Usage Notes

1. Since SCRIPT does not perform a final page eject after encountering the QUICK-QUIT control word, some output that has been formatted may never be displayed.

2. The .QQ control word is useful when you are using the .TE (TERMINAL-INPUT) control word to enter lines from the terminal, and you want to terminate processing quickly.

3. When the .QQ control word is processed, SCRIPT terminates processing without closing files.

***

.QU (QUIT)

QUIT causes processing to terminate with a final page eject. The format of the .QU control word is:

```
.QU
```

Usage Notes

1. The QUIT control word causes the output form to be advanced to the top of the next page and processing is then terminated immediately.

2. The .QU control word will cause termination no matter where or when it is encountered, including within imbedded files (see the .IM control word). All open SCRIPT files are closed before processing terminates.

***

.RC (REVISION-CODE)

The REVISION-CODE control word allows you to designate a revision code marker to be printed along the left hand margin. The format of the .RC control word is:

```
.RC  n  s
        ON
        OFF
        ON/OFF
```

where:

n specifies the revision code number from 1 to 9.

s specifies the revision code symbol to be printed along the left hand margin. It may be any single character, including the blank. If not specified, a blank character is assumed.

ON signifies the beginning of text to be marked with the code character associated with RC n.

OFF signifies the end of text to be marked with the code associated with RC n.

ON/OFF signifies that the next input line should be marked with the RC n code on output.

Usage Notes

1. The REVISION-CODE control word has two functions: (1) to define a revision code symbol and (2) to activate the revision code. You may have up to 9 revision codes defined at any time, and each revision code may be assigned a different symbol. The operands ON and OFF activate and deactivate the actual revision code marking, respectively. The operand ON/OFF has the effect of turning ON revision code n for one line only, the line that is next printed after the .RC n ON/OFF is processed.

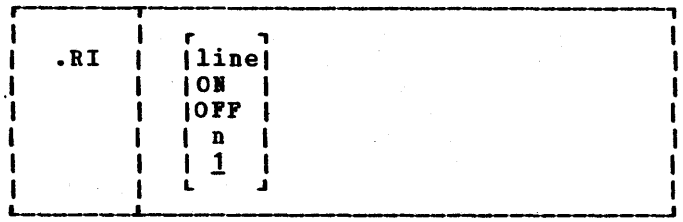
2. By assigning different symbols to different revision code numbers, including the blank, it is possible to selectively print specific revision code markers or differentiate between various levels of revision.

3. Since the .RC control word does not cause an automatic break, revision code markings may be turned on and off within a paragraph or even a sentence without disrupting normal SCRIPT formatting. An explicit .BR control word may be necessary under certain circumstances.

4. In order to provide space for the marker along the left hand margin, SCRIPT indents the output an additional amount. Whenever any nonblank revision code is defined, all output is indented an additional two spaces, even if the revision code has not been turned on. Therefore, you should define all your revision code symbols at the beginning of your SCRIPT file so that all the output is indented the same amount

.RI (RIGHT-ADJUST)

Use the RIGHT-ADJUST control word to position an output line flush with the right margin. The format of the .RI control word is:



where:

line is the line to be right-adjusted.

ON begins right adjusting of text lines.

OFF stops right adjusting of text lines if .RI ON is in effect, or if n was given and has not been exhausted.

n is the number of subsequent input lines to be right adjusted. If no number is given, 1 is assumed. If .RI n is specified when .RI ON is in effect, right-adjust mode is turned off when n lines have been formatted, or when .RI OFF is encountered.

Usage Notes

1. When the line is right-adjusted, leading blanks are considered part of the line's length. When a line is right-adjusted, no formatting is done. That is, the line is justified as it stands, and it is not filled from other input lines.
2. This control word acts as a break.
3. If the line to be right-adjusted is longer than the current column length, it is truncated, and the excess is used on a second line. However, this second line is not right-adjusted unless the number of lines to be adjusted is large enough to include it.
4. The .RI control word is a variant of the .CE (CENTER) control word. If either of these control words is processed, the other is canceled.

Example

```
.ri 3
                    These three lines are
                          right-adjusted,
                          as you can see.
```

***

.RV (READ-VARIABLE)

The READ-VARIABLE control word is just like the .SE (SET-SYMBOL) control word, except that the value of the symbol is read from the terminal. The format of the .RV control word is:

where:

symbolname is the name of the symbol to be set. It may be any name that would be allowable on the left hand side of the equal sign in a .SE control word.

Usage Notes

1. When the .RV control word is encountered, a VM READ is issued so that a line may be read from your terminal. This line is used as the right hand side of the equal sign to set the value of the symbol named in the .RV control word. Any expression that would be allowable as the value in a .SE control word is allowable here. If no name is given on the .RV control word, it is ignored, and no line is read from the terminal.
2. The .RV control word does not cause an automatic break.
3. No message is displayed before the terminal is unlocked to accept the input line. You may use the .TY (TYPE-ON-TERMINAL) control word to issue a prompting message before the .RV control word issues its terminal read.

Example

A symbol called "name" could be set with the following control word:

```
.se name = 'John Doe'
```

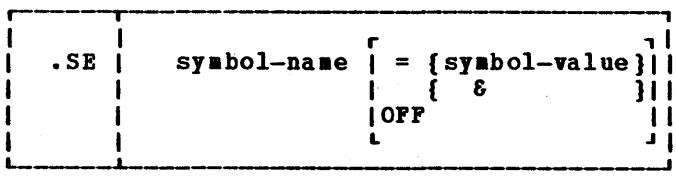
characters on each line governed by the line length instead of the column length. If no multiple column mode is in effect when this control word is processed, the only thing it might change is the active column length, if you had set one different than the line length.

2. This control word is not allowed in a keep.

***

.SE (SET-SYMBOL)

The SET-SYMBOL control word allows you to define and assign values to symbols or arrays of symbols. Using the SET-SYMBOL control word, you can give a symbolic name to a page number, a word, or even a string of SCRIPT control words. The format of the .SE control word is:



where:

symbol-name
: is the name you want to assign a symbolic value to be substituted during SCRIPT processing.

symbol-value
: assigns a value to the symbol-name; it may be a character string or arithmetic expression.

&
: assigns the symbol-name a value equal to the current page number.

OFF
: unsets the symbol symbol-name so that, as far as SCRIPT is concerned, it was never set.

Symbol Names

A symbol name may be any character string of 10 characters or less, but may not contain any of the following special characters:

```
.   -   +
*   /   (
)   '   &
```

During SCRIPT processing, a symbol name is recognized when it is preceded by an ampersand (&) and followed by a blank or a period:

```
&symbol-name
```

If the symbol name appears in any of the following forms:

```
symbol-name()
symbol-name(n)
symbol-name(&symbol)
```

it is an array symbol. See "Symbol Processing" in Section 1 for details.

Symbol Values

If a symbol-value is set to a character string that contains any embedded blanks or any special characters, it must be enclosed in single quotes. For example,

```
.se dog = cat
.se end = '.gu'
.se sentence = 'This is a sentence.
```

are all valid character strings. If you want a character string to contain a single quote ('), you must enter two of them, for example

```
.se title = 'Mrs. O''Grady''s Cat'
```

If symbol value is an arithmetic expression, it must be in the form:

```
[op0] n1 op1 n2 op2 n3 op3 n4...
```

where:

op0 is a unary + or - sign.

op1, op2, op3
: are arithmetic operators:

```
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
```

n1, n2, n3 ...
: are any valid integers. The integers may have been assigned their values as a result of a symbol substitution (including the page number symbol).

For example,

```
.se nextpage = & + 1
.se current = -100
.se addit = &current + 25
.se answer = 15 - 42
```

are all valid arithmetic expressions.

C indicates conditional spaces. No spaces are generated if the next control word processed is another .SP, a .SK, a .PA, or a control word (like a .Hn) or macro whose first action is one of these functions.

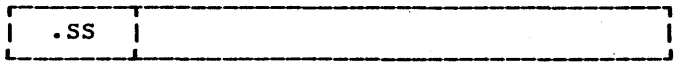
Usage Notes

1. This control word acts as a break.
2. If a page eject occurs while SCRIPT is processing a .SP control word, remaining blank lines are inserted after the top titles on the following page. If you do not want spaces to appear at the top of the page, use the .SK (SKIP-LINES) control word.
3. If double- or line-spacing is in effect, the number of blank lines generated is multiplied by the line spacing amount, unless absolute spacing is specified.

xxx

.SS (SINGLE-SPACE-MODE)

Use the SINGLE-SPACE-MODE control word to cancel a previous .DS (DOUBLE-SPACE-MODE) or .LS (LINE-SPACING) control word, and to resume single-spacing of output. The format of the .SS control word is:



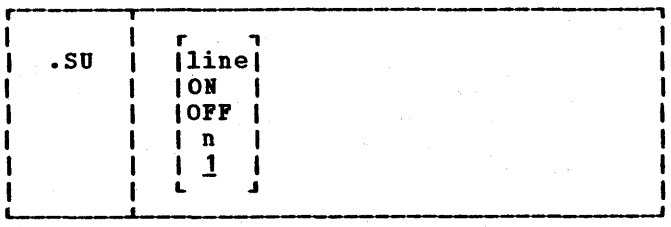
Usage Notes

1. This control word acts as a break.
2. Output following the SINGLE-SPACE-MODE control word is single spaced. Since this is the normal output format, .SS is needed only to cancel a previous .DS or .LS control word.

xxx

.SU (SUBSTITUTE-SYMBOL)

Use the SUBSTITUTE-SYMBOL control word to cause SCRIPT to stop substitution of defined set symbols or to restore substitution. The format of the .SU control word is:



where:

line is a line containing symbolic expressions that you want SCRIPT to substitute with values previously set. Symbols may be set via the .SE, .RV, .IM, or .AP control words, or by a macro call.

ON turns on an open ended substitution mode. ON is the initial setting.

OFF turns off substitution mode if it was ON, or if n was given and not yet exhausted.

n specifies the number of following lines to be scanned for set symbols to be substituted. If omitted, 1 is assumed.

Usage Notes

1. The SUBSTITUTE-SYMBOL control word causes a specified number of the following input lines, control words as well as text, to be scanned for defined set symbols. If the argument ON is in effect, every line up to a subsequent .SU OFF will be scanned. Substitution ON is the default mode of operation, but it is reset to OFF with .SU OFF; with .SU n, after n lines have been read; or with ".SU line" after the line is scanned.
2. Multiple scans are performed over the input line until no further set symbol substitution is necessary.
3. The substitution of set symbols may increase or decrease the length of the text line. If the line's length reduces to zero, it becomes a "null line." A null line causes a break. If the line's length expands so that it exceeds 240 characters, an error condition occurs if a single variable substitution caused the line overflow.
4. The TWOPASS option may result in defining symbols during pass1 which can be used for substitution during pass2 even though these symbols are defined physically later in the SCRIPT file. Under rare circumstances, the substitu-

Examples

1. .tb 10 20 */30 40

Tab positions are interpreted as columns 10, 20, and 30. If a tab character is processed between positions 20 and 30 of a line, the positions from the current position up through and including position 30 are filled with asterisks (*) instead of blanks. The next character goes in position 31. For example,

(TAB)text(TAB)text(TAB)text

results in:

text text******text

2. .tb

Tab positions revert to default values of 5, 10, 15, etc.

***

.TC (TABLE-OF-CONTENTS)

The TABLE-OF-CONTENTS control word causes the automatically generated table of contents to be imbedded and printed. Entries may be placed in the table of contents by head level control control words (.H0 through .H6) and by the .PT (PUT-TABLE-OF-CONTENTS) control word. The format of the .TC control word is:

.TC	[n [name]]

where:

n is the number of pages to be reserved for the table of contents. If omitted, 1 is assumed. This operand is meaningful when the table of contents is at the front of the document, and the TWOPASS option is used to process it.

name is an optional line to be used on the table of contents. If no name is given, the word CONTENTS is used. A pseudo head-level 1 is generated at the top of the table of contents using the name given or the word CONTENTS.

Usage Notes

1. When .TC is encountered, a pseudo head level 1 is processed. This means that a page eject is done if not already at the top of a page, but no entry is placed in the table of contents for the head. All table of contents entries that have been saved in the utility file IKSUT2 are then formatted and printed. The entries come from the head level control words whose definitions call for table of contents entries. By default, the control words .H0 through .H3 cause these entries.

   The table of contents is formatted according to the line and page dimensions in effect at the time the .TC control word is encountered, not those in effect when the head level was processed. Each line in the table has the revision code and the page number that was in effect when the head level was processed.

   When the table of contents is completely formatted, the utility file IKSUT2 is erased. Another page eject is done, and the new page is numbered as though sequential page numbering had occurred and the table of contents had occupied exactly n pages. If the table takes other than n pages, there will be either a gap or an overlap in pagination.

2. This control word acts as a break. It is not allowed in a keep.

Example

See the table of contents of this book for an example of an automatically generated table of contents. Note the treatment of heads too long to fit on one line with the page number.

***

.TE (TERMINAL-INPUT)

Use the TERMINAL-INPUT control word when you want to enter text or control lines during the processing of the input file. The format of the .TE control word is:

.TE	[n | ON | OFF] [1]

.TR (TRANSLATE-CHARACTER)

The TRANSLATE-CHARACTER control word allows you to specify the output representation of each character in the source text. The format of the .TR control word is:

.TR	[s t]

where:

s is a source character under consideration. It may be a single character or a 2-character hexadecimal code.

t is the intended output representation of the source character. It may be a single character or a 2-character hexadecimal code.

Usage Notes

1. After formatting of an input source line has been completed and immediately before actual output, each character of the output line may be translated to a different output code. The TRANSLATE CHARACTER control word is primarily of use when the final output device uses a different character set than was used to create the source SCRIPT file.

2. The text associated with title lines is translated under control of the translations in effect at the time that the title control word was processed. If you change the translations after the title has been saved for future use, it is too late to affect that title.

3. Since control words are only processed internally, they are never translated. However, text data associated with a control word (such as title data) can be translated.

4. Translate character specifications remain in effect until explicitly respecified.

5. A .TR control word with no operands causes the translation table to be reinitialized and all previously specified translations to be reset.

6. The UPCASE command option has the same effect as the 26 TRANSLATE CHARACTER control words: ".tr a A;.tr b B; ... ;.tr z Z".

7. By using the .IF, .CS, or .TE control words, you may specify different output character sets for different runs with different output devices.

8. The .TR control word does not cause a break. If you have a section of text that has translation characters in effect, followed by a .TR to reset the translations, the last line of the text may not yet have been printed. In this case, that last line is not translated.

The hexadecimal codes for each printable character on a TN print train is shown in Figure 7.

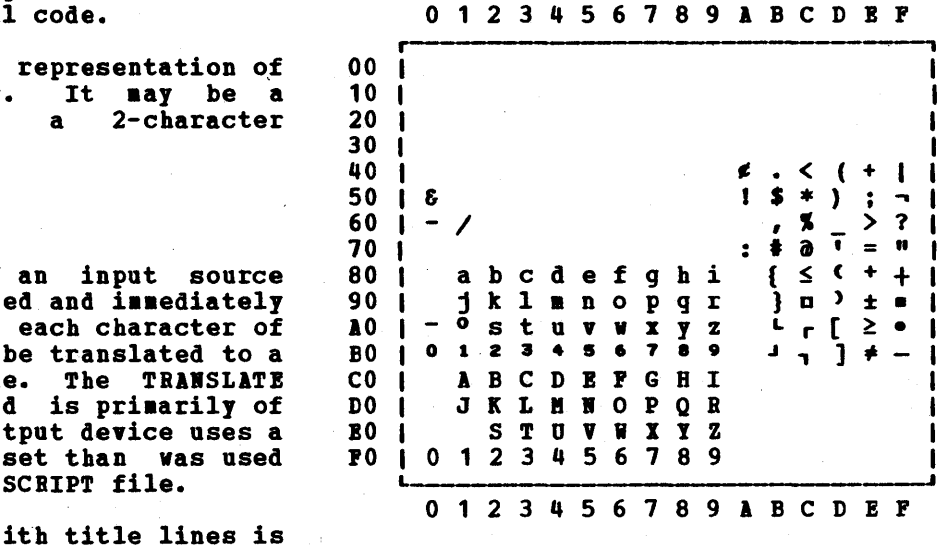


Figure 7. TN Translate Table.

Examples

1. .tr 0 b0;.tr 1 b1; ...;.tr 9 b9

This causes the characters 0, 1, ..., 9 to print as their corresponding superscript character if the output device is a printer equipped with the TN train. For example, the formula:

X2+Y2=Z3

prints as

$X^2+Y^2=Z^3$

2. .tr 40 ?

This causes all blanks in the file to be typed as questions marks (?) on output.

***

document driven messages when the formatted output is going to a printer or to a disk file.

Example

```
.ty Do you want 2 column output?
.rv answer
.if x&answer ne xyes .go by2col
.cd 2 0 46
.cl 43
...by2col
```

***

.UC (UNDERSCORE-CAPITALIZE)

The UNDERSCORE-CAPITALIZE control word automatically underscores and capitalizes an input line. The format of the .UC control word is:

```
┌──────┬──────────────────────────────┐
│ .UC  │ text                         │
└──────┴──────────────────────────────┘
```

where:

text is the line to be capitalized and underscored.

Usage Notes

1. Use the UNDERSCORE-CAPITALIZE control word whenever you have a line of data that is to be formatted in capital letters and underscored. This control word is a combination of the .UP (UPPERCASE) and the .US (UNDERSCORE) control words.

2. The .UC control word does not cause an automatic break; single words in a sentence may be underscored and capitalized.

Example

```
This sentence has
.uc one
word processed by .UC.
```

results in

This sentence has ONE word processed by .UC.

***

.UD (UNDERSCORE-DEFINITION)

Use the UNDERSCORE-DEFINITION control word to specify which characters should be underscored whenever automatic underscoring is done. The format of the .UD control word is:

```
┌──────┬──────────────────────────────┐
│      │ ┌ ┌  ┐                       │
│ .UD  │ │ │ON │ s1 s2 .. s9│         │
│      │ │ │OFF│            │         │
│      │ └ └  ┘              ┘         │
└──────┴──────────────────────────────┘
```

where:

ON specifies that the following characters are to be underscored.

OFF specifies that the following characters are not to be underscored.

s is either a single character or a 2-character hexadecimal code representing a character that is defined for underscoring (ON) or not underscoring (OFF).

Usage Notes

1. When a line is automatically underscored, certain characters are left not underscored. The control words that cause underscoring are .US (UNDERSCORE), .UC (UNDERSCORE-CAPITALIZE), and any .Hn (HEAD-LEVEL-n) control word whose current definition calls for it. Using the .UD control word, you can override the defaults and cause any character to be underscored (ON) or not underscored (OFF). If you do not give any options with the control word, but put ".UD" alone, the default values are restored.

2. This control word does not cause a break.

3. The maximum number of arguments on the .UD line is ten. If you want to change more than nine characters, you must do it with more than one .UD control word. (ON or OFF plus nine more arguments.) Each .UD control word only changes the characters specified, and leaves the rest of the characters unchanged.

4. The following characters are not underscored by default:

2. The .UP control word does not cause an automatic break. Single words in a sentence may be capitalized.

Example

```
This sentence has
.up one
capitalized word.
```

results in

This sentence has ONE capitalized word.

***

.US (UNDERSCORE)

This control word automatically underscores an input line. The format of this control word is:

```
| .US | text |
```

where:

text is the line to be underscored.

Usage Notes

1. It is a good idea not to have too much data underscored when the output is in typewriter format. In this format, each underscored character is actually three characters: underscore, backspace, character. Thus, the output line may be too long if there are many underscored characters in it.

2. The UNDERSCORE control word does not cause a break. Single words in a sentence may be underscored.

3. Special characters, such as punctuation marks, blanks, and tabs are not usually underscored. If you need to underscore these characters, or if you want to suppress underscoring of certain characters, use the .UD (UNDERSCORE-DEFINITION) control word.

Example

```
This sentence has
.us one
underscored word.
```

results in

This sentence has one underscored word.

***

WARNING MESSAGES

Warning messages indicate that some difficulties were encountered during SCRIPT processing. These difficulties did not prevent the command from continuing processing, but it might not yield the desired results. See "Message Descriptions" for the actual warning messages that SCRIPT issues.

ERROR MESSAGES

These messages are issued when you have not supplied the conditions required by SCRIPT to execute. Error situations ordinarily cause SCRIPT processing to terminate, but if the CONTINUE option of the SCRIPT command is in effect, processing continues after the error message is issued. However, the erroneous condition may have caused irremedial degradation that will later result in an error condition too severe to allow continuation. See "Message Descriptions" for the actual error messages that SCRIPT issues.

SEVERE ERROR MESSAGES

These messages are issued when an error occurs that is too severe for processing to continue. Processing terminates after the message is issued, even if the CONTINUE option is in effect. Some severe error conditions result from having continued after an Error message was issued. See "Message Descriptions" for the actual severe error messages that SCRIPT issues.

TERMINAL ERROR MESSAGES

Terminal error messages are issued for very serious and unexpected errors. See "Message Descriptions" for the actual terminal error messages that SCRIPT issues.

IKS010E: UNDENT>INDENT.
The execution of a .IN, .IL, .UN, or .OF control word would cause the left margin to move to the left of character position 1.
Return Code = 12

IKS011S: NO READ/WRITE DISK SPACE AVAILABLE FOR OUTPUT OR UTILITY FILE.
SCRIPT must have a read/write A-disk available to write the special files IKSUT1 (for delayed imbed files) IKSUT2 (for the table of contents generated by HEAD-LEVEL-n control words), or for the $filename SCRIPT created when you use the FILE option of the SCRIPT command.
Return Code = 40

IKS012E: HEADING MARGIN+HEADING SPACE>TOP MARGIN.
The heading space and the heading margin are part of the top margin; the top margin must be large enough to accommodate the other two. See Figure 2 to see the relationship.
Return Code = 12

IKS013E: FOOTING MARGIN+FOOTING SPACE>BOTTOM MARGIN.
The footing space and the footing margin are part of the bottom margin; the bottom margin must be large enough to accommodate the other two. See Figure 2 to see the relationship.
Return Code = 12

IKS014W: READ-ONLY SYMBOL MAY NOT BE SET.
An attempt was made to change one of the read-only symbols that SCRIPT assigns values internally ($CL, $LL, $PL, and so on). The values represented by these symbols may be changed using the appropriate control word. Often, there is other processing that must be performed before the value itself is changed.
Return Code = 4

IKS015E: .RC MODE WAS ON OR OFF ALREADY.
A .RC n ON was encountered while revision code n was already on, or a .RC n OFF was encountered while revision code n was either already off or stacked by an inner .RC ON. (See .RC)
Return Code = 12

IKS016E: INVALID .RC TERMINATION - NUMBER CURRENTLY ON.
An attempt was made to redefine a revision code character while that revision code was active (ON).
Return Code = 12

IKS017S: EXCESSIVE OR NEGATIVE SPACE COUNT GENERATED.
SCRIPT has calculated that it should space a negative number of lines. This shouldn't happen unless the CONTINUE option has allowed processing to continue after an illegal page length, top margin or the like has been set.
Return Code = 40

IKS018S: I/O ERROR OR DEVICE ERROR.
An error message should have been issued by CMS or CP to describe the error condition.
Return Code = 100

IKS019W: MESSAGE NOT ASSIGNED.
Return Code = 4

IKS020S: CORRECT FORM IS: "SCRIPT FILENAME (OPTIONS)"; TYPE "SCRIPT ?" FOR MORE INFORMATION.
A filename was not specified in the SCRIPT command.
Return Code = 24

IKS031E: UNDEFINED SYMBOL USED AS INDEX OF .SE (SET-SYMBOL) ON LEFT OF EQUAL SIGN.
A symbol name of the form symbol1(&symbol2) was used in a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word when symbol2 was an undefined symbol name.
Return Code = 12

IKS032E: INVALID (NON-DECIMAL) NUMBER USED AS INDEX OF .SE (SET-SYMBOL) ON LEFT OF EQUAL SIGN.
A symbol name of the form symbol1(n) or symbol1(&symbol2) was used in a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word when n or the value of symbol2 was not a valid decimal number.
Return Code = 12

IKS033E: INVALID (NON-DECIMAL) NUMBER ENCOUNTERED IN EXPRESSION ON RIGHT SIDE OF .SET.
The symbol value given for a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word was an arithmetic expression that included a term that was not a valid decimal number. This error can result from attempting to set a symbol to a character string such as FIFTY-FIVE, where SCRIPT interprets the hyphen as a minus sign. In this case, use single quotes around the character string ('FIFTY-FIVE').
Return Code = 12

IKS034E: UNDEFINED SYMBOL ENCOUNTERED IN EXPRESSION ON RIGHT SIDE OF .SET.
The symbol value given for a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word was an arithmetic expression that included a symbolic term, when the symbol for that term was undefined.
Return Code = 12

IKS035E: A TOKEN LONGER THAN 16 CHARACTERS ENCOUNTERED IN .SET.
A token of more than 16 characters has been encountered in a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word line. A token may not be more than 16 characters unless it is part of a character string delimited by single quotes. If it is in quotes (on the right-hand side of the equal sign), the string may be as long as 255 characters.
Return Code = 12

IKS036E: MORE THAN 10 TOKENS ENCOUNTERED IN .SET.
A maximum of 10 tokens (symbols, punctuation, numbers) is allowed in a .SE (SET-SYMBOL) or .RV (READ-VARIABLE) control word line. For example, the line ".se ALPHA = BETA * 2 - GAMMA + 13" has exactly 10 tokens.
Return Code = 12

IKS037S: INFINITE LOOP OCCURRED AS A RESULT OF RECURSIVE .SE (SET-SYMBOL) SUBSTITUTION.
Symbol substitution was attempted on an input line where every time a symbol value was substituted, the value contained another symbol name.
Return Code = 12

IKS038E: SUBSTITUTION FOR .SE (SET-SYMBOL) CAUSES LINE TO EXCEED MAXIMUM LENGTH.
Symbol substitution was attempted on an input line when substituting the symbols on the line would make the line longer than 240 characters.
Return Code = 12

IKS047W: KEEP TERMINATED BECAUSE TOO MANY LINES.
The maximum number of lines in a keep is the same as the number of text lines that can fit in a current column. In general, this number is the page length minus the top margin minus the bottom margin. The maximum number of lines in a footnote or a headnote is 10. If the maximum is exceeded, SCRIPT continues processing, after ending the keep, footnote, or headnote, but input lines after this are not part of the keep. Note that the maximum that has been exceeded is the number of formatted output lines, not the number of input lines.
Return Code = 4

IKS048S: LABEL NOT FOUND.
A GOTO control word was encountered that refers to a nonexistent label.
Return Code = 12

IKS049S: UNABLE TO ALLOCATE BUFFER SPACE.
There is not enough available storage for processing labels, or to maintain the save/restore or imbed stack. You may redefine your virtual storage size using the CP DEFINE STORAGE command. (See the VM/370: CP Command Reference for General Users.)
Return Code = 104

IKS050S: TOO MANY LABELS - NO ROOM IN BUFFER.
The label storage table is large enough for about 120 active labels. A ... (SET-LABEL) control word has been encountered when the table is already full.
Return Code = 40

IKS051E: DUPLICATE LABEL FOUND.
A ... (SET-LABEL) control word was encountered that specifies a label name that has already been set for this file.
Return Code = 12

IKS052W: MESSAGE NOT ASSIGNED.
Return Code = 4

IKS053W: LABELS NOT ALLOWED DURING TERMINAL INPUT.
A ... (SET-LABEL) or GOTO control word was entered from the terminal during terminal input mode. In this mode, the terminal keyboard is a simulated input file, except that it is meaningless to assign labels to the resulting input lines.
Return Code = 4

IKS054E: INVALID KEYWORD FOR CONDITION ON .IF.
An .IF control word was encountered and the condition test was not recognized. Conditions such as "eq" and "<=" are recognized. This condition can happen if the first comparand on the .IF control word is a null symbol because the tokens are then shifted over. See the .IF control word notes.
Return Code = 12

IKS055T: UNABLE TO ALLOCATE LINK STORAGE.
Link storage is the area SCRIPT uses for formatting the text. Without this storage area, SCRIPT can do nothing. Redefine your virtual storage size with the CP DEFINE STORAGE command. (See the VM/370 CP Command Reference for General Users.)
Return Code = 104

If a file named SCRIPT MODULE previously existed on your A-disk, it is replaced.

To verify the correct installation of SCRIPT, proceed to Step 3.

STEP 2B. EXECUTE THE IKSGEND EXEC PROCEDURE

Issue the commands:

```
tape rewind
tape fsf
tape load
```

You should receive the message:

```
LOADING...
IKSGEND   EXEC     C1
IKSPRT    TEXT     A1
IKSSEM    TEXT     A1
IKSSIO    TEXT     A1
IKSECW    TEXT     A1
IKSCOL    TEXT     A1
IKSSYM    TEXT     A1
IKSINT    TEXT     A1
IKSBOX    TEXT     A1
IKSERR    TEXT     A1
IKSEDC    TEXT     A1
IKSHYP    TEXT     A1
IKSEND    TEXT     A1
IKSEZS    TEXT     A1
IKSEDI    TEXT     A1
SAMPLE    SCRIPT   A1
```

All the files you need to generate the SCRIPT module are now loaded onto your A-disk.

To generate the SCRIPT module, enter the command:

```
iksgend
```

If you want your SCRIPT module to include EasySCRIPT or hyphenation capabilities, you can enter the keywords "HYP", "EZS", or "ALL" on the command line. If you don't want either, enter the keyword "NONE". If you do not enter any keywords, IKSGEND prompts you for the information it needs.

When the IKSGEND EXEC begins execution, it first checks to see if a file named SCRIPT MODULE A2 already exists. If it does, it issues the message:

```
' SCRIPT MODULE A2' EXISTS
```

Otherwise, the message

```
' SCRIPT MODULE A2' DOES NOT EXIST
```

This check ensures that an existing SCRIPT module is not inadvertently destroyed.

IKSGEND also checks for the existence of a file named SCRIPT MODOLD A1. It issues one of the following messages:

Error Conditions

If any errors occur during the loading process, the following messages are issued:

```
*** IKSGEND LOAD ERROR.
*** MAKE SURE ALL SCRIPT TEXT FILES ARE AVAILABLE.
```

You should check that all the files on the SCRIPT distribution tape were properly loaded onto a read/write disk before the IKSGEND EXEC was invoked. Then reexecute the IKSGEND EXEC.

Generate the HYPEDIT Module

If you chose to generate your SCRIPT module with hyphenation, you should now generate the HYPEDIT module. HYPEDIT is a special editor which must be used to create the hyphenation exception dictionary. To generate HYPEDIT, issue the commands

```
load iksedi
genmod hypedit
```

STEP 3. VERIFY CORRECT INSTALLATION OF THE SCRIPT MODULE

You should now execute the sample problem supplied on the distribution tape to ensure that the SCRIPT module is correctly functioning. This file is named SAMPLE SCRIPT.

If you followed the instructions in Step 2A of the installation procedure, you must now issue the command:

```
tape load sample script
```

to load the SAMPLE SCRIPT file onto your A-disk.

If you followed the instructions in Step 2B of the installation procedure, the SAMPLE SCRIPT has already been loaded.

To execute the sample problem with the results appearing on the terminal, issue the command:

```
script sample (twopass
```

You should receive the SCRIPT version identification message:

```
SCRIPT/370 VERSION 3, LEVEL n - mm/dd/yy
```

If you are using a typewriter terminal, you should also receive the message:

```
IKS101R: ADJUST PAPER; THEN PRESS RETURN:
```

The sample problem asks you some questions before beginning actual output, so you may want to wait to insert fresh paper, if you wish.

If you want the output from the sample problem to be output on the system printer, issue the SCRIPT command as follows:

SAMPLE PROBLEM FOR SCRIPT/370 VERSION 3

PLANNING TO USE SCRIPT

SCRIPT is a simple text processing program, but it is also a text processing language. While many users only take advantage of a small number of the SCRIPT language processing capabilities, you may want to consider using some of the techniques suggested below, as you plan for the growth and development of a SCRIPT shop.[1]

STANDARDIZED FORMATS

In general, while you may be using SCRIPT for a variety of documentation purposes, you will probably settle on a small number of basic formats. There are two basic ways that you can make the standardization process easier:

1. Create setup files, with the control words necessary to provide for particular formats. Then use the IMBED control word (.IM) to imbed this file in each document that you produce. A sample setup file to describe multiple-column formatting might look like the following:

```
.pl 84
.bm 8;.tm 8
.cw
.se 1col='.cd 1;.cl 89'
.se 2col='.cd 1 0 46;.cl 43'
.ll 89
.cw ;
```

2. Define high-level GML tags that describe formatting structures. These tags may be defined in symbol libraries (filetypes of MACLIB) whose use is requested by the SCRIPT command option LIB. For example, a member named NUM may be defined in the library OURLIB MACLIB whose contents are:

```
;.SK 2;.OF 5P
```

   To use the tag &num in a SCRIPT file, you must invoke

---

[1] These techniques are not formally recommended or tested by IBM, but represent practical considerations based on the experiences of a variety of users.

```
THEN BEGIN CREATING/EDITING THE FILE
&END
&READ VARS &FN
EDIT &FN SCRIPT
-LOOP &TYPE ENTER FILENAME OF NEXT FILE:
&TYPE PRESS NULL LINE TO QUIT EDITING SESSION
&READ VARS &FN
&IF .&FN EQ . &EXIT
EDIT &FN SCRIPT
&GOTO -LOOP
```

As users become more comfortable with the system, you can begin to introduce newer concepts. The SCRIPT/370 Version 3 User's Guide is organized along these lines: if you follow the presentation of topics as they are presented (according to your specific needs), you may find it easier.

FILE AND MINIDISK REQUIREMENTS

Your data base may be arranged on a minidisk basis: files pertaining to a particular document might be on a single minidisk, files belonging to another document on another minidisk, and so on. Each SCRIPT user might or might not have a personal ID to use for private files (or practicing), and to use when many people are cooperating on working on the same document.

Oftentimes, as in the case of setup files that describe standards, many users will need access to the same files. This can be most easily accomplished by placing all common files on a single minidisk, and providing all other minidisks with LINK directory statements so that these files are automatically accessed (with the help of an ACCESS command in the PROFILE EXEC).

In cases where a large document uses files from several other minidisks, you may find it advisable to place the CP and CMS commands necessary to link to and access these disks directly in the SCRIPT file, using the .SY (SYSTEM-COMMAND) control word. After imbedding the particular file, use the CMS RELEASE command to release the file directories from storage (CMS may need this storage later).

MAINTAINING SCRIPT XDICT

If you are using SCRIPT's automatic hyphenation capabilities, then one file that should be placed on the common disk is the SCRIPT XDICT, the exception dictionary

```
.* .et /Sample Problem Output///;.ot ///Sample Problem Output/
.sy set blip off
.if x&check = xtwo .go start
.ty Welcome to the Sample Problem for SCRIPT/370, Version 3
.ty Is your SCRIPT module generated with EasySCRIPT? (Respond yes or no):
.rv ez
.ty Is your SCRIPT module generated with Hyphenation? (Respond yes or no):
.rv hy
.ty Okay, now we're going to eject a page and begin output...
.ty The CMS BLIP function has been turned off to protect your output...
...start
.pn off
.if SYSOUT eq PRINT .tt ////
.pa 1
.sp 25
.dm centerit /.ce/.up &*/
.ms on
.centerit Sample Problem for SCRIPT/370 Version 3
.pa
.pn on;.pn roman
.if SYSOUT eq PRINT  .ob ///Contents   &/
.if SYSOUT eq PRINT  .eb $&   SCRIPT/370 Version 3$$$
.tc
.ms off
.pn arabic
.if SYSOUT eq TERM .tt /Sample Problem//Page &/
.pa
.h1 Planning to Use SCRIPT
.if SYSOUT eq PRINT  .ob ///Sample Problem   &/
SCRIPT is a simple text processing program, but it is also
a text processing language.
While many users only take advantage of a small number of the
SCRIPT language processing capabilities, you may want to
consider using some of the techniques suggested below,
as you plan for the growth and development of a SCRIPT
.fn on
.if SYSOUT eq PRINT .tr * b1
.of 1
*These techniques are not formally recommended or tested by IBM,
but represent practical considerations based on the experiences
of a variety of users.
.of 0
.fn off
shop.*
.br;.tr
.h2 Standardized Formats
In general, while you may be using SCRIPT for a variety of
documentation purposes, you will probably settle on a small number
of basic formats.
There are two basic ways that you can make the standardization
process easier:
.sk;.tb 5;.of 5
 1.  Create setup files, with the control words necessary to
provide for particular formats.
```

```
of training in the use of the CMS Editor, CMS command language,
and so on.
.pp You may bypass the problems of teaching CMS and CP commands immediately
by providing new users with EXEC procedures that call the editor
recursively, spool the printer, and so on.
Here's an example of an EXEC named EDIT:
.sk;.in +5;.fo off;.su off
&CONTROL OFF
&IF &INDEX EQ 0 &GOTO -DO
&IF &INDEX = 1 EDIT &1 SCRIPT
&IF &INDEX GT 1 EDIT &1 &2 &3 &4 &5 &6 &7
&EXIT
-DO &BEGTYPE
ENTER THE FILENAME (1 TO 8 CHARACTERS):
THEN BEGIN CREATING/EDITING THE FILE
&END
&READ VARS &FN
EDIT &FN SCRIPT
-LOOP &TYPE ENTER FILENAME OF NEXT FILE:
&TYPE PRESS NULL LINE TO QUIT EDITING SESSION
&READ VARS &FN
&IF .&FN EQ . &EXIT
EDIT &FN SCRIPT
&GOTO -LOOP
.in -5;.fo on;.su on
.pp As users become more comfortable with the system, you can
begin to introduce newer concepts.
The
.us SCRIPT/370 Version 3 User's Guide
is organized along these lines:
if you follow the presentation of topics as they are presented
(according to your specific needs), you may find it easier.
.h2 File and Minidisk Requirements
Your data base may be arranged on a minidisk basis: files
pertaining to a particular document might be on a single minidisk,
files belonging to another document on another minidisk, and so on.
Each SCRIPT user might or might not have a personal ID to use for
private files (or practicing), and to use when many people are
cooperating on working on the same document.
.pp Oftentimes, as in the case of setup files that describe standards,
many users will need access to the same files.
This can be most easily accomplished by placing all common files
on a single minidisk, and providing all other minidisks with LINK
directory statements so that these files are automatically accessed
(with the help of an ACCESS command in the PROFILE EXEC).
.pp In cases where a large document uses files from
several other minidisks, you may find it advisable to
place the CP and CMS commands necessary to link to and access these
disks directly in the SCRIPT file, using the .SY (SYSTEM-COMMAND)
control word.
After imbedding the particular file, use the CMS RELEASE command
to release the file directories from storage (CMS may need this
storage later).
.if x&hy ne xyes .go nohy
```

Control Word	Operand Format	Function	Page	Break	Default Value
.co (CONCATENATE-MODE)	ON|OFF	Causes output lines to be formed by concatenating input lines.	84	Yes	ON
.cp (CONDITIONAL-PAGE-EJECT)	n	Causes a page eject if fewer than n lines remain on the page.	84	No	
.cs (CONDITIONAL-SECTION)	n ON|OFF INCLUDE|IGNORE	Allows conditional inclusion of input in the formatted output.	85	No	
.cw (CONTROL-WORD-SEPARATOR)	s	Specifies the character used for separation of control words on a single input line.	86	No	
.dh (DEFINE-HEAD-LEVEL)	n options	Changes the characteristics associated with the heading levels.	87	No	
.di (DELAY-IMBED)	n|ON|OFF|line	Delays the inclusion of a portion of the input file until the next page eject occurs.	87	Yes	1
.dm (DEFINE-MACRO)	name /line... name x name OFF	Creates a macro definition using SCRIPT control words, text lines, and special symbols.	88	No	
.ds (DOUBLE-SPACE-MODE)		Specifies that subsequent formatted output will be double spaced.	89	Yes	
.eb (EVEN-PAGE-BOTTOM-TITLE)	n /.../.../.../	Specifies that a title line for the bottom of the current page, if it is even-numbered, and all subsequent even-numbered pages.	90	No	
.ef (END-OF-FILE)	CLOSE	Simulates an end-of-file condition.	90	No	
.ep (EVEN-PAGE-EJECT)	ON|OFF	Causes one or two page ejects such that the next page is even-numbered; or prints text only on even-numbered pages.	91	Yes	
.et (EVEN-PAGE-TOP-TITLE)	n /.../.../.../	Specifies a title line for the top of each subsequent even-numbered page.	92	No	
.ez (EASYSCRIPT)	ON lastDewey OFF tagval line	Invokes the EasySCRIPT GML tags to provide automatic formatting of SCRIPT files.	92	No	

Figure 8. SCRIPT Control Word Summary (Part 2 of 7)

Control Word	Operand Format	Function	Page	Break	Default Value
.il (INDENT-LINE)	n|+n|-n	Indents only the next line the specified number of characters.	102	Yes	0
.im (IMBED)	filename tokens	Inserts a file of text and/or control words into the one being processed by the SCRIPT command.	102	No	
.in (INDENT)	n	Specifies the number of spaces subsequent text is to be indented.	103	Yes	0
.ju (JUSTIFY-MODE)	ON|OFF	Causes right justification of output.	104	Yes	ON
.kp (KEEP)	ON|OFF FLOAT|DELAY	Ensures that formatted text lines remain in the same column when printed.	104	No	
.li (LITERAL)	n|ON|OFF|line	Insures that input lines are read as text lines by SCRIPT.	105	No	1
.ll (LINE-LENGTH)	n|+n|-n	Specifies the number of characters, including blanks, in each subsequent line.	106	Yes	60
.ls (LINE-SPACING)	n	Specifies the number of blank lines to be inserted after each subsequent output text line.	106	Yes	0
.mc (MULTI-COLUMN-MODE)		Causes SCRIPT to revert to to formatting in multiple columns after single-column processing.	106	Yes	
.ms (MACRO-SUBSTITUTION)	ON|OFF	Causes SCRIPT to begin making automatic macro calls, or cancels macro calls.	107	No	
.ob (ODD-PAGE-BOTTOM-TITLE)	n /.../.../.../	Specifies a title line for the bottom of the current page, if it is odd numbered, and all subsequent odd-numbered pages.	107	No	
.of (OFFSET)	n|+n|-n	Provides a technique for indenting all but the first line of a section.	108	Yes	0

Figure 8. SCRIPT Control Word Summary (Part 4 of 7)

Control Word	Operand Format	Function	Page	Break	Default Value
.re (RESTORE-STATUS)		Restores status of SCRIPT variables from a push down stack created by SAVE-STATUS.	115	No	
.ri (RIGHT-ADJUST	n|ON|OFF|line	Causes lines to be printed flush with the right margin.	116	Yes	1
.rv (READ-VARIABLE)	name	Allows you to enter a value for a symbol name from the terminal.	116	No	
.sa (SAVE-STATUS)		Saves the status of SCRIPT variables.	117	No	
.sc (SINGLE-COLUMN-MODE)		Causes SCRIPT to revert to formatting text into a single column.	117	Yes	
.se (SET-SYMBOL)	symbol=value & OFF	Defines and assigns values to symbolic names, interfacing with the macro capabilities of SCRIPT.	118	No	
.sk (SKIP-LINES)	n A C	Specifies the number of blank lines to insert before the next line, unless it is at the top of a column.	119	Yes	1
.sp (SPACE-LINES)	n A C	Specifies the number of blank lines to be inserted before the next output line.	119	Yes	1
.ss (SINGLE-SPACE-MODE)		Specifies that subsequent formatted output will be single spaced.	120	Yes	
.su (SUBSTITUTE-SYMBOL)	n|ON|OFF|line	Controls whether SCRIPT symbol names with their previously-assigned values.	120	No	1
.sy (SYSTEM-COMMAND)	commandline	Executes the specified CP or CMS command during SCRIPT processing.	121	No	
.tb (TAB-SETTING)	n n n n... f/n f/n f/n...	Specifies the logical tabs used when the document is printed or typed by SCRIPT/370.	121	Yes	5,10,15...75
.tc (TABLE-OF-CONTENTS)	n name	Imbeds the IKSUT2 file, which consists of table entries made with the .PT control word and the heading control words.	122	Yes	

Figure 8. SCRIPT Control Word Summary (Part 6 of 7)

Control Word	Changes
.AP	Tokens passed to an appended file (&1 through &9) are reset by macro calls. &0 contains the number of tokens passed.
.BC	New operands ON and OFF restore and cancel column balancing.
.BM	Now uses the default value, 6, if no number is specified. The bottom margin can now also be specified as a positive or negative increment to the current value.
.BX	BOX (New/3)
.CD	You can now define up to 9 displacements for columns, even if you specify only 1 column at this time. The remaining displacements are used when you later increase the number of columns with a .CD n.
.CE	Now accepts a data line as a parameter.
.CL	Now uses the default value, 0, if no number is specified. The column length can now also be specified as a positive or negative increment to the current value.
.CO	New operands, ON and OFF, restore and cancel concatenation.
.DH	DEFINE-HEAD-LEVEL (New/2)
.DM	DEFINE-MACRO (New/3)
.EF	New operand, CLOSE, causes SCRIPT to close the file, so that if it is imbedded again, SCRIPT begins reading at the top of the file, not at the line following the .EF.
.EP	The ON operand allows you to specify printing only on even-numbered pages. The OFF operand restores printing on all output pages.
.EZ	EASYSCRIPT (New/3)
.FI	Obsolete. Use .FO.
.FO	New operands, ON and OFF, allow you to restore and cancel formatting, including concatenation and justification.
.FM	Now uses the default value, 2, if no number is specified. The footing margin can now also be specified as a positive or negative increment to the current value.
.FN	FOOTNOTE (New/2)
.FS	Now uses the default value, 1, if no number is specified. The footing space can now also be specified as a positive or negative increment to the current value.
.FT	Obsolete. Use bottom title (.BT, .EB, .OB) control words.
.GO	GOTO (New/2)
.HE	Obsolete. Use top title (.TT, .ET, .OT) control words.

Figure 9. SCRIPT Control Word Compatibility (Part 1 of 3)

Control Word	Changes
.PN	Has new operands: (1) FRAC and NORM, to initiate fractional page numbering and restore normal numbering (2) PREF, to specify a character string prefix for all page numbers.
.PP	PARAGRAPH-START (New/2)
.PT	PUT-TABLE-OF-CONTENTS (New/3)
.QQ	QUICK-QUIT (New/2)
.RI	Now accepts a data line as a parameter.
.RV	READ-VARIABLE (New/2)
.SC	SINGLE-COLUMN-MODE (New/2)
.SE	The OFF operand allows you to cancel a symbol value. You can now use two single quote marks to achieve a single quote within a symbol value.
...	SET-LABEL (New/2)
.SK	SKIP-LINES (New/2)
.SP	New operands, A (for absolute spacing) and C (for conditional spacing).
.SU	The initial value has been changed to ON.
.SY	SYSTEM-COMMAND (New/2)
.TC	TABLE-OF-CONTENTS (New/2)
.TM	Now uses the default value, 6, if no number is specified. The top margin can now also be specified as a positive or negative increment to the current value.
.TT	No change; however, you can now get a single top title to print on page 1.
.UC	UNDERSCORE-CAPITALIZE (New/2)
.UD	UNDERSCORE-DEFINITION (New/2)
.UN	An undent can now also be specified as a positive or negative increment to the current value.
.UP	UPPERCASE (New/2)
.US	UNDERSCORE (New/2)

Figure 9. SCRIPT Control Word Compatibility (Part 3 of 3)

SH20-1857-0

SCRIPT/370 Version 3 User's Guide Printed in U.S.A. SH20-1857-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
360 Hamilton Avenue, White Plains, New York 10601
(International)

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail

No postage stamp necessay if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
1133 Westchester Avenue
White Plains, New York 10604

Att: Technical Publications/Industry – Dept. 825

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
360 Hamilton Avenue, White Plains, New York 10601
(International)

SCRIPT/370 Version 3 User's Guide Printed in U.S.A. SH20-1857-0