SH20-1114-0

Installed User Program

SCRIPT/370 Text Processing Facility Under Virtual Machine Facility/370 (VM/370) Program Description/ Operations Manual

Program Number 5796-PAF

This manual describes an IBM internally-developed program called SCRIPT/370. This program executes as a command of the Conversational Monitor System (CMS), a component of VM/370. SCRIPT/370 is a successor to SCRIPT, a text processing Type III program supplied with CP-67/CMS. Through the facilities of SCRIPT/370, text files developed using the CMS Editor may be formatted in single- or multiple-columns, justified or ragged, and with automatic pagination. Additional facilities of the SCRIPT processor permit accepting input from a terminal during processing, the inclusion of other SCRIPT files, and extensive top and bottom title (i.e., running head and foot) capabilities. Other formatting and control is facilitated by special symbols that may be substituted for frequently-used control word sequences or used to generate tables of contents.

IBM

First Edition (November 1972)

This edition corresponds to Release 1 of SCRIPT/370 and to all subsequent modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are periodically made to the specifications herein; before using the publication in connection with the operation of IBM systems, refer to the latest IBM System/360 and System/370 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office in your locality.

Address comments concerning the contents of this publication to IBM Corporation, VM/370 Publications, 24 New England Executive Park, Burlington, Massachusetts, 01803.

This document has been formatted entirely by SCRIPT/370. The original copy was printed on an IBM 1403 printer under control of VM/370.

Users of SCRIPT/370 will find further information about the CMS and CP commands available to them in the following IBM publications:

> _IBM Virtual Machine Facility/370_: _EDIT Guide_, GC20-1805
> _IBM Virtual Machine Facility/370_: _Terminal User's Guide_, GC20-1810
> _IBM Virtual Machine Facility/370_: _Command Language User's Guide_, GC20-1804

Persons evaluating the use of SCRIPT/370 in an installation should review, in addition to this manual, the following IBM publications:

> _IBM Virtual Machine Facility/370_: _Introduction_, GC20-1800
> _IBM Virtual Machine Facility/370_: _Planning and Systems Generation Guide_, GC20-1801

Programmers planning to extend the facilities of SCRIPT/370 through modification will need the following licensed publication:

> _SCRIPT/370 IUP_: _Systems Guide_, Form LY20-0762

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

The SCRIPT/370 Text Processing Facility (SCRIPT/370) is invoked via the SCRIPT command of the Conversational Monitor System (CMS), a component of the Virtual Machine Facility/370 (VM/370). Installation of the Script processor in a VM/370 system automatically validates the SCRIPT command. SCRIPT/370 operates on files developed through use of the CMS Editor. As the user develops his file from the terminal, he includes SCRIPT control words in the form

```
.control-word xxxx
```

which direct the operation of the SCRIPT processor as it reads the file.

SCRIPT/370 has many formatting capabilities that make it useful for many different kinds of documents such as working papers, program documentation, reports and internal newsletters. Combined with the powerful context editing capabilities of the CMS Editor, SCRIPT/370 is an efficient and economical means of preparing these documents. Modifications and additions to documents can be made easily. In addition, the virtual machine environment of VM/370 permits text processing applications to be run concurrently with other installation work.

SCRIPT/370 provides various text processing capabilities, including

- One to nine columns of text per page
- As-is, justified, right-adjusted, or ragged right composition
- Automatic page numbering in Arabic or lower-case Roman
- Automatic generation of top and bottom titles, with even and odd page options
- Simple generation of form letters with or without variable information from a terminal
- Inclusion of the contents of other SCRIPT files

- Extensive macro capabilities to permit defining shorthand symbols for frequently-used sequences of control words

- As many as nine revision indicators that will print at the left margin

- Conditional printing of different sections of a document

PREPARING TO USE SCRIPT/370

Because SCRIPT/370 operates in the CMS environment of VM/370, to make use of it you must first establish a connection with VM/370 from your typewriter terminal. Once you have done this, you use the CMS editor to develop a Script file which contains the information for your document. This information consists of text, the information you wish to be printed, and Script control words, special statements that direct the operation of SCRIPT/370 and determine the appearance of your printed output.

YOUR TYPEWRITER TERMINAL

Your communication with the computer may be by means of an IBM 2741 Communications Terminal, similar to an IBM Selectric (R) typewriter. You will find a switch on the left side of the cabinet in which the typewriter is mounted; when the switch is set to LCL you have an ordinary Selectric typewriter; by switching to COM you can be linked to the central computer by telephone line, and your typewriter is now a typewriter terminal. In this latter mode, you and the computer converse through messages that you type via your terminal to the computer, and responses you receive back at your terminal. If your terminal is not an IBM 2741, refer to the document _IBM Virtual Machine Facility/370_: _Terminal User's Guide_ for information about your terminal.

PREPARING TO LOG IN

In order to log in, that is, to establish a connection with the system from your terminal, you need a user identification (userid or ID) and a password, and which you can obtain from the operations group at your installation. To prepare your terminal, set the left margin at 1 and the right margin at 130, in order to provide maximum typing width. (You should have a #963 typing element for an EBCDIC terminal, and a #015 element for a correspondence terminal.) Make sure the switch on the left side of the cabinet is on COM.

LOGGING IN

Procedure 1. If your terminal has a direct wire connection, simply press the ON button. If there is no response, hit ATTN. The system types

```
     vm/370 online   ijh359 qsyosu
```

Press the ATTN key.

Procedure 2. If your terminal has a telephone line connection, proceed as follows:

a) Turn the terminal on.
b) Press the TALK button on the data set and dial the proper telephone number.
c) When a high-pitched continuous tone is heard on the phone, depress the DATA button on the data set and replace the phone in its holder.
d) After the system types out

```
     vm/370 online  ijh359 qsyosu
```

e) Press the ATTN key.

After you have completed either procedure 1 or procedure 2, you are ready to log into the system. Log in by typing

```
     login 'ID'
```

For example,

```
     login smith
```

Press RETURN. System responds with

```
     ENTER PASSWORD:
```

Enter your password and hit RETURN. (Typeout is suppressed so that no copy of your password appears on the terminal sheet if your terminal has the Print Inhibit feature.)

The system may respond with messages giving special instructions for system users. It always responds with

```
LOGON AT 'time' 'day' 'date'
```

Now that you are logged in, you are ready to access CMS, which contains SCRIPT/370. Type

```
     ipl cms
```

and press RETURN. The system responds with

```
     CMS..VERSION n.m mm/dd/yy hh.mm
```

Note: In this publication, single quotation marks denote that the entry is not a literal entry; it is merely a description of the information to be entered. For instance, 'ID' indicates that some particular userid should be entered--without the quotation marks.

The printout at your terminal should look something like this

```
     vm/370 online  ijh359 qsyosu

     login smith
     ENTER PASSWORD:

     CP WILL RUN UNTIL 17:00 AND FROM 18:00 TO 24:00
     LOGON AT 12.00.00 EST THURSDAY 11/30/72

     ipl cms
     CMS..VERSION 1.0 11/30/72 12:00
```

You are now ready to create a file, print an existing file, edit (make changes to) an existing file, etc. You are in the CMS command environment. By entering simple commands (such as EDIT, TYPE, etc.) from the terminal, you access an extensive set of CMS commands, as well as create and format Script files.

To distinguish your entries from system responses, it is a good idea to type your commands in lowercase. (Text lines of a Script file should, of course, be typed exactly as you wish them to appear on output.) The system generally types its messages to you in uppercase. For example, you could type

```
    login SMITH            (combination of uppercase
                              and lowercase)
```

or

```
    LOGIN SMITH            (uppercase only)
```

or simply

```
    login smith            (this is recommended)
```

Your VM/370 command entries appear to the system as uppercase, no matter how you type them in; the practice of using only lowercase for commands is merely a convention to improve the readability of your terminal printout.

YOUR SCRIPT FILE

The source data for your document is stored for you by the data management facilities of CMS and is called a Script file. It is composed of records and resides on disk space allocated to your virtual machine and identified by your userid. Files are identified by a unique combination of filename (of your own choosing), filetype (SCRIPT, for all Script files), and filemode (you need not concern yourself with this for a while). Script files consist of text lines and special command words that you type in at your terminal.

CREATING AND MODIFYING YOUR SCRIPT FILE WITH EDIT

The EDIT facility enables you to create your Script file, make changes to the file, or simply peruse its contents.

There are two modes of operation when using the Edit facility: Input and Edit.

To make corrections to a file you have created in Input mode, you must enter Edit mode. Entry to Edit is automatic when you issue the command shown below. You can use Edit commands to make revisions to your file; for example, you can locate a word or string of words, change a word string to another word string, go to the next line in your file, delete words from that line, and so on.

You can go back and forth between Edit and Input modes (you will learn how to do this later on), typing in text and Script commands (Script commands all begin with a period in position 1) in Input mode, and correcting your entries in Edit mode.

The EDIT command is used to invoke the Edit facility:

```
    edit 'filename' script
```

It is important to familiarize yourself with the system by means of a practice session at the terminal; in no other way can you gain the facility that you will need later on for creating and operating on your own files. While at the terminal, you might wish to refer to Appendix B, which lists and briefly describes all the SCRIPT commands explained in greater detail in the body of this manual.

CORRECTING TYPING ERRORS

Errors must be corrected before hitting the RETURN key. (If you have failed to do this, the EDIT facility can be used to correct errors to a file that has been written and "saved". This will be described later.)

One or several @'s deletes one or several preceding characters--which may be blanks--and effectively backspaces the typing element.

Thus

```
     This es@xail@@mple
```

is interpreted as

```
     This example
```

since the first @ canceled the s (and backspaced one space), and the next two @'s canceled the i and l (and backspaced twice).

A line is canceled by the ¢ symbol. Thus if, instead of correcting individual mistakes in the above example, you had decided to start over again, you would have typed

```
     This esail¢
```

and hit RETURN. Then you would have retyped

```
     This example
```

Note: @ cannot cancel ¢. Once a ¢ has been typed, all preceding characters (and blanks) are canceled, and the effective line begins with the next character after the ¢ sign.

CREATING A SCRIPT FILE

Entering a New File

First, decide on a filename, sometimes abbreviated to fn in this manual. The maximum length of a filename is eight characters. Any of the alphameric characters may be used in the filename: A-Z, a-z, 0-9, #, @, $. The filename and filetype (SCRIPT, in this case) uniquely identify a file for access by CMS commands.

When you are creating a file, make certain you choose a new and unique filename. Next, issue the command

```
     edit 'filename' script
```

and hit RETURN. The system responds with

```
     NEW FILE.
     EDIT:
```

You are now in Edit mode; enter Input mode by typing

```
     i
```

or

```
     input
```

The system responds with

```
     INPUT:
```

You are now in Input mode and ready to type in your file, one line at a time, hitting RETURN at the end of each line.

Your Script file contains textual information plus special Script command words for controlling the format of your output. These commands are typed on separate lines from the text, and always start with a period in position 1.

Filing It Away

When you have finished typing in your file you must store it. To do this, you must leave Input mode and enter Edit mode. To enter Edit, hit RETURN on a "null" line (a line on which nothing has been typed--not even spaces--and, therefore, the typing element is at the left margin).

The system responds with

```
     EDIT:
```

You type

```
     file
```

and hit RETURN. The system types

```
     R; T= 'CPU times' 'time of day'
```

Saving Entries For Future Filing

At intermediate points in developing your file, it is generally a good idea to save your entries up to that point (as a safeguard against general system failure), without leaving the Edit facility. You can do this by pressing RETURN (on a null line).

System responds with

```
     EDIT:
```

You type:

```
     save
```

and hit RETURN. System types

```
     EDIT:
```

You may now continue editing or adding to your file from the point where you stopped in order to issue the Save command. If you want to add to your file, you must type:

```
     input
```

and hit RETURN.

The File command takes you from Edit mode to the general CMS command environment. It is more useful when you have finished creating or modifying a file.

Correcting Your File

To make changes, additions, or deletions to your file, you simply reissue

```
     edit 'fn' script
```

which places you in Edit mode. (See "Using SCRIPT/370" for a fuller description of the Edit facility. A complete description can be found in the publication, *IBM Virtual Machine Facility/370: EDIT Guide*, Order No. GC20-1805.)

Printing Your File

Later on, you will learn the commands for printing out your file at your terminal and on the offline printer. These printouts can be done either in the format you have determined with your Script control words or unformatted, that is, with text lines and commands in the same format in which they were entered. (See "The SCRIPT Command".)

LOGGING OUT

At the completion of all your work, including printing your file, you will want to end your session at the terminal by logging out of VM/370. To do so, type:

```
     logout
```

and hit RETURN. The system types out three sets of times:

```
     CONNECT='time' VIRTCPU='time' TOTCPU='time'
```

and

```
     LOGOUT AT 'time' on 'date'
```

Press the OFF button on the terminal.

SAMPLE TERMINAL SESSION

The following example illustrates the use of VM/370, CMS and SCRIPT/370. You should take this example as a guideline for your first terminal session.

By the convention of this document, system responses are shown in uppercase while commands to the system are shown in lowercase. Of course the file called SAMPLE SCRIPT is a combination of upper- and lowercase and is stored that way by CMS.

```
vm/370 online      ijh359 qsyosu

login jones
ENTER PASSWORD:

CP WILL RUN UNTIL 17:00 AND FROM 18:00 TO 24:00
LOGON AT 11:00:00 EST THURSDAY 11/30/72

ipl cms
CMS...VERSION 1.0  11/30/72 11:00

edit sample script
NEW FILE:
EDIT:
input
INPUT:
 Text lines and characters are entered in upper and
lower case as appropriate.  If no Script control
words are specified, SCRIPT/370 will automatically
generate lines justified at both the left and right
margins (that is, text printed in the format
of this publication), at 66 lines per page
and 60 characters per line.
 When entering text, you can begin each
new paragraph with a tab stroke or blank as the
first character. This serves as a break to the
process of joining (or concatenating)
lines for output.
 When all the text is entered, type a null
line.

EDIT:
file
R;

script sample
SCRIPT/370  VERSION 1-LEVEL 0.
ADJUST PAPER; PRESS RETURN
```

```
 Text lines  and characters are  entered in upper  and lower
case  as  appropriate.  If  no  Script  control  words  are
specified,  SCRIPT/370  will  automatically  generate  lines
justified at both the left and  right margins (that is, text
printed in the format of this  publication), at 66 lines per
page and 60 characters per line.
 When entering text, you can begin each new paragraph with a
tab stroke or blank as the first character. This serves as a
break to the process of joining (or concatenating) lines for
output.
 When all the text is entered, type a null line.


R;


logoff jones
CONNECT=
LOGOFF AT
```

When the SCRIPT command is entered in CMS without any options, a formatted copy of the Script file is typed on the terminal if a carriage return (null line) is entered after the prompting message. See the section on Script Command Options for other capabilities of the SCRIPT command.

SCRIPT/370 CONTROL WORDS

This section describes SCRIPT/370 control words and how they are used to determine the appearance of documents printed via the SCRIPT command. The first set of control words presented consists of those used for page setup and titling. These control words enable you to specify top and bottom margins, line length, and top and bottom titles (running heads and feet).

The second set of control words described in this section comprises those used to control text formatting. These control words determine whether your text will be centered or flush with the left or right margins, whether it will be right-justified, as-is, or ragged-right. (These terms are explained and illustrated as part of the appropriate control word descriptions.)

Following the text formatting control words are those for supplemental page formatting such as tab setting, indenting, page ejection and reserving space for illustrations or material from other SCRIPT/370 files.

The remaining control words are those that permit you to specify margin indicators for revised portions of your document, include only certain portions of it when printing, accept input from a terminal during printing, and direct selected portions of your document to the terminal even though the remainder is being printed offline. Two control words, set-symbol and substitute-symbol, are described separately at the end of this section.

NOTATIONAL CONVENTIONS USED IN THIS MANUAL

Each control word is shown in both its full and abbreviated form. You may enter either form when developing your document, as long as you begin each control word with the period character (.). Control words are separated from their arguments by a single blank. Multiple arguments are separated by single blanks. Although the diagonal character (/) is shown as the delimiter between portions of an argument, as in the even-page top-title control word, any character that does not appear in the argument can be used.

PAGE SETUP AND FORMATTING

Page setup and formatting consists basically of determining the dimensions of the final text on the page. Using the control words described below, you can specify top and bottom margins, left and right margins, the margins between body text and top and bottom titles (heads and feet), and the type and position of page numbering. The diagram below illustrates a piece of paper, marked by the *s. The body of the manuscript text is enclosed within the inner rectangle marked "Text". The following margin settings are indicated on the diagram:

.BM	Bottom-margin
.FM	Footing-margin
.HM	Heading-margin
.IN	Indent
.LL	Line-length
.PL	Page-length
.TM	Top-margin

```
************************************************************
*    |    !    |    |                             |        *
*    |    !    |    |                             |        *
*    |    !   .TM   |   Top Title (if any)        |        *
*    |    !    |    |            |                |        *
*    |    !    |    |           .HM               |        *
*    |    !    |    |            |                |        *
*    |    !------------------------------------------------*
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*   .PL   !         |           Text              |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !         |                             |        *
*    |    !------------------------------------------------*
*    |    !    |    |            |                |        *
*    |    !    |    |           .FM               |        *
*    |    !    |    |            |                |        *
*    |    !   .BM   |    Bottom Title (If any)    |        *
*    |    !    |    |                             |        *
*    |    !    |    |                             |        *
************************************************************
          !<--.IN-->|
          !<-----------------.LL----------------->|
          !<---Column 1 of typewriter or printer platen.
```

SPECIFYING THE OVERALL DIMENSIONS OF A PAGE

You specify the overall dimensions of a page primarily with two control words

LINE-LENGTH or .ll
and
PAGE-LENGTH or .pl

These are supplemented by use of the various top and bottom margin control words to determine exactly where the text begins and ends on the page whose length you have specified.

LINE-LENGTH (.ll)

The LINE-LENGTH control word determines the maximum number of characters that will be printed as measured from the left margin. This value includes spaces, indents, tabs, and offsets (See the OFFSET control word.), but not underscores. Its format is

```
.ll n
```

n specifies the maximum number of characters in each output line. If you do not enter a LINE-LENGTH control word, SCRIPT/370 will choose an output line length of 60.

The actual number of characters in any given output line is determined by the combination of the line length you have specified and the text formatting you have selected (See Text Formatting Control Words). If you have specified CONCATENATE-JUSTIFY, SCRIPT/370 will first attempt to fill up the line by taking words from the next line without exceeding the line length specified by the .ll control word. SCRIPT/370 then fills the remainder of the available line length by inserting blanks between words. Lines longer than the specified output line length are shortened by moving words to the next line and inserting blanks in the resulting shorter line.

The line length you specify takes effect on the first page started after the control word is read by SCRIPT/370.

PAGE-LENGTH (.pl)

The PAGE-LENGTH control word communicates to SCRIPT/370 the maximum number of lines and line-spaces that can fit on the physical page. This allows SCRIPT/370 to keep track of how many lines remain before it must eject to the next page, or pause so that you may insert a new page. Its format is

```
.pl n
```

n is the number of lines possible on the output page. If you do not enter a PAGE-LENGTH control word, SCRIPT/370 will choose a page length of 66. This corresponds to six lines per inch (standard for most terminals) using 11-inch paper.

Because the PAGE-LENGTH control word may be included anywhere in the document as often as necessary, you are not limited to using only one size of paper for your output. By specifying PAGE-LENGTH as 82, for example, your document, or portions of it will print properly on legal-size paper. This same page length could be used to print your document or certain pages of it on oversize pages which were to be photo-reduced during reproduction.

TOP-MARGIN and BOTTOM-MARGIN (.tm and .bm)

Using the TOP-MARGIN and BOTTOM-MARGIN control words, you determine what portion of the total page length will be occupied by body text. These control words allow you to specify the number of blank lines that will appear at the top and bottom, respectively, of each page of your document. If, for example, you had specified a page length of 66 (.pl 66), and top and bottom margins of 30, your document would be printed with six lines per page.

The format of the TOP-MARGIN control word is

```
.tm n
```

n is the number of blank lines to appear between the top of the page and the first line of body text. If you do not enter a TOP-MARGIN control word, SCRIPT/370 will choose a value of 6.

If you change top margin settings within your document, each new setting takes effect on the first page started after the TOP-MARGIN control word is read by SCRIPT/370.

The format of the BOTTOM-MARGIN control word is

```
                    .bm n
```

n   is the number of blank lines to appear between the last line of body text and the bottom of the page.If you do not enter a BOTTOM-MARGIN control word, SCRIPT/370 will choose a value of _6_.

If you change bottom margin settings within your document, each new setting takes effect on the first page to be ended after SCRIPT/370 reads the BOTTOM-MARGIN control word. This means that the bottom margin of the _current_ page is changed.

HEADING MARGIN (.hm)

With the HEADING-MARGIN control word, you specify how many blank lines are to appear between the heading and the first line of body text. This means that the heading actually occupies one line of the top margin. Naturally, the value specified for the heading margin must be less than that specified for the top margin.

The format of the HEADING-MARGIN control word is

```
                    .hm n
```

n   is the number of blank lines to appear between the heading and the first line of body text. If you do not enter a HEADING-MARGIN control word, SCRIPT/370 will choose a value of _1_.

FOOTING-MARGIN (_.fm_)

The FOOTING-MARGIN control word determines the number of blank lines that will appear between the last potential line of body text and the footing text. Since the footing text occupies one line of the bottom margin, the value specified for the footing margin must be less than that specified for the bottom margin.

The format of the fotting margin control word is

```
.fm n
```

n	is the number of blank lines to appear between the last potential line of body text and the footing text.

TOP-TITLE (.tt)

Top titles or running heads are useful to readers of your document as an aid in finding the portion of the document that contains a given block of information. (Notice the use of headings in this manual.)

SCRIPT/370 provides the TOP-TITLE control word so that you need specify a heading only once, and have it printed at the top of each subsequent page until another TOP-TITLE (or other titling control word) is encountered. The TOP-TITLE control word allows you to specify three separate parts so that on subsequent pages, you can retain a chapter heading, for example, and change a section heading.

The format of the TOP-TITLE control word is

```
.tt /left/center/right/
```

/	is any character that does not appear anywhere in the title. It is used by SCRIPT/370 as a separator between adjacent parts of the top title.
left	is the text to be printed beginning at the left margin.
center	is the text to be printed centered.
right	is the text to be printed justified at the right margin.

When specifying a top title, you should take into account the current line length. since the top title text may not exceed this value. Because SCRIPT/370 formats the top title from left to right, the 'center' text may overlay the last portion of the 'left' text, and may in turn be overlayed by the beginning of the 'right' text if you have inadvertently specified them as being too long to fit within their respective portions of the line.

When writing the text of your top title, you should consider the character ampersand (&) reserved. That character is used as the page number symbol (unless you have specified another character with the PAGE-NUMBER-SYMBOL control word) and can be used by you in the text of the title whenver you want the number of the page to print.

The TOP-TITLE control word may be used anywhere within your document to change the heading of subsequent pages.

EVEN-PAGE-TOP-TITLE(.et)
ODD-PAGE-TOP-TITLE(.ot)

If you require the page number to be part of the top title, and you expect your document to be reproduced using both sides of the sheet, you could respecify the top title with each page. This would allow you to cause the page number to print at the left on even-numbered pages, and at the right on odd-numbered pages. This would be cumbersome. Therefore, SCRIPT/370 provides two special-purpose variations of the TOP-TITLE control word called EVEN-PAGE-TOP-TITLE (.et), and ODD-PAGE-TOP-TITLE (.ot).

The format of the EVEN-PAGE-TOP-TITLE control word is

```
.et /left/center/right/
```

The format of the ODD-PAGE-TOP-TITLE control word is

```
.ot /left/center/right/
```

The meanings of (/), 'left', 'center', and 'right' are the same as those in the explanation of the TOP-TITLE control word.

By specifying the page number symbol (ususally ampersand

(&)) in the 'left' text of your EVEN-PAGE top title and as the 'right' text of your ODD-PAGE top title, you will cause SCRIPT/370 to print the page number in the proper position for conventional reproduction.

BOTTOM-TITLE(.bt)
EVEN-PAGE-BOTTOM-TITLE(.eb)
ODD-PAGE-BOTTOM-TITLE(.ob)

SCRIPT/370 provides three other control words for analagous use with footing text or bottom titles. These control words are BOTTOM-TITLE (.bt), EVEN-PAGE-BOTTOM-TITLE (.eb), and ODD-PAGE-BOTTOM-TITLE (.ob). Notice that page number has been specified in the bottom titles of this manual.

The format of these control words is

```
.bt /left/center/right/

.eb /left/center/right/

.ob /left/center/right/
```

SPECIFYING PAGINATION

SCRIPT/370 provides you with many facilities for controlling the pagination of your document. You may suppress page numbering altogether or just on output, cause the numbering to be in Arabic or lower case Roman, and reset page numbers within a document.

PAGE-NUMBER-SYMBOL(.ps)

As mentioned above, in the explanation of the TOP-TITLE control word, the ampersand (&) is usually reserved for use as the page number symbol. SCRIPT/370 substitutes the current page number for each occurrence of the ampersand within a title line. Naturally, there may be times when you must use the ampersand within a top or bottom title. There may also be instances in which the terminal or printer on which your document is to be printed does not include the ampersand. To allow for these situations, SCRIPT/370 provides the PAGE-NUMBER-SYMBOL control word.

The PAGE-NUMBER-SYMBOL control word allows you to substitute another character for the ampersand as a representation of the current page number.

The format of the PAGE-NUMBER-SYMBOL control word is

```
                    .ps s
```

s	is any single character except the blank. This character will be interpreted by SCRIPT/370 as a request to substitute the current page number whenever it is encountered in a top or bottom title.

The character specified in the PAGE-NUMBER-SYMBOL control word takes effect for all subsequent top and bottom titles. This means that if you have previously used the ampersand or some other character as the page number symbol, you will have to revise those control words to reflect use of the new symbol. If, for instance, you have previously entered an ODD-PAGE-TOP-TITLE control word in the form

```
          .ot /Chapter I/Industrial Relations/&/
```

and you later change the page number symbol to the plus sign (+) SCRIPT/370 will not recognize the ampersand in your odd page top title as a page number symbol. Instead, it will print the ampersand, treating it as text. To prevent this, you must change the ampersand in the title control word to a plus sign, using the CMS editor, to correspond to the changed page number symbol.

<u>PAGE-NUMBERING-MODE</u> (<u>.pn</u>)

The PAGE-NUMBERING-MODE (.pn) control word lets you determine whether automatic page numbering is to take place, and whether it is to be in Arabic or lower case Roman. SCRIPT/370 maintains two sets of page numbers for your document: internal and external. (The counter used by the SCRIPT command program to record page numbering as the document is processed contains the internal page number.) With the PAGE-NUMBERING-MODE control word you can control these separately or dispense with both. If you wanted to print certain pages for inclusion in another document, for example, you would probably prefer to have page numbers suppressed so that they could be filled in later in the final composite document. If you are following the usual conventions, you would also number your frontis pages in lower case Roman, and not begin numbering in Arabic until the first page of text.

The format of the PAGE-NUMBERING-MODE control word is

```
.pn on|off|offno|arabic|roman
```

on	causes page numbering to be resumed if you have previously entered a .pn off or .pn offno. If you do not enter any PAGE-NUMBERING-MODE control word, SCRIPT/370 will choose the On condition.
off	causes external page numbering to be discontinued. Internal page numbering continues, however. This allows you to resume printing the page number at the correct point when you enter .pn on.
offno	suppresses both internal <u>and</u> external page numbering. A subsequent .pn on causes both internal and external page numbering to be resumed beginning with the next sequential page number, regardless of how many actual pages have been processed since the .pn offno was entered. This can be especially useful if your document is to include pages meant to be removed. Using .pn offno followed by .pn on, you can suppress numbering of those pages, and preserve the numbering sequence that should exist after they have been removed.
arabic	causes all subsequent page numbers to be printed as Arabic numerals. If you do not enter a PAGE-NUMBERING-MODE control word, SCRIPT/370 chooses Arabic numerals.
roman	causes all subsequent page numbers to be printed as lower-case Roman numerals.

You can suppress inclusion of page numbers in top and bottom titles simply by omitting the page number symbol from the title definition.

EXAMPLE OF PAGE SETUP AND FORMATTING

This example is based on one shown in the Announcement Notice for SCRIPT/370 (Form G320-1520-0). Since the margin settings are not specified, they assume the default values. The text itself will be formatted with lines justified at the left and right margins since that is the default mode of output. The page-eject function causes the top title to print on the first page.

Terminal input:

```
.line-length 35
.page-length 27
.top-title ///ANNOUNCEMENT NOTICE/
.page-eject 1
.bottom-title /IBM IUP//PAGE &/
SCRIPT/370 provides text processing
capabilities to users of the IBM Virtual Machine
Facility/370, VM/370.  It executes as a command
of the Conversational Monitor System, the
time-shared component of VM/370.
.space-lines 1
The SCRIPT command creates formatted output
from one or more CMS files, each of which contains
text and/or Script control words.  The Script
files are created and modified at a terminal
using the CMS editor.
```

SCRIPT/370 Output:

```
                    ANNOUNCEMENT NOTICE


   SCRIPT/370 provides text-processing
   capabilities  to users  of the  IBM
   Virtual     Machine    Facility/370,
   VM/370. It executes as a command of
   the Conversational  Monitor System,
   the   time-shared   component    of
   VM/370.

   The     SCRIPT     command   creates
   formatted output  from one  or more
   CMS files,  each of  which contains
   text and/or  Script control  words.
   The  Script files  are created  and
   modified  at a  terminal using  the
   CMS editor.


   IBM IUP                      PAGE 1
```

TEXT FORMATTING

SCRIPT/370 gives you many choices as to the appearance of your final text. The control words described in the previous section give you the facilities for determining the overall dimensions and relative positions of the different elements of the page: text, top and bottom title, margins, etc. The control words described in t<u>his</u> section allow you to determine exactly how your body text is to appear within the margins.

The five basic text formats are

1. Justified - even left and right margins
2. Ragged right - even left margin, uneven right margin
3. As-is - output line corresponds exactly to input line regardless of margins
4. Right-adjusted - Even right margin, uneven left margin
5. Centered - Equidistant from both margins

JUSTIFIED TEXT

SCRIPT/370 justifies your text by inserting blanks as needed between words until the line length you have specified is filled. In CONCATENATE-JUSTIFY mode, SCRIPT/370 obtains words from the next line until no more words can be placed on the line without exceeding the line length. It then inserts blanks as needed to fill out the line. If the input line is too long to fit within the output line length, SCRIPT/370 removes words from it and spills them into the next input line, and fills the first line with blanks as needed.

FORMAT-MODE (.fo)

This mode of operation is the one chosen by SCRIPT/370 unless you enter appropriate control words to specify some other mode. In other words, it is the default mode of SCRIPT/370. It is called CONCATENATE-JUSTIFY mode, or FORMAT mode. The two principal control words associated with the standard mode are FORMAT-MODE (.fo) control word and the NO-FORMAT-MODE (.nf) control word which is explained later.

The format of the FORMAT-MODE control word is

```
.fo
```

The only use of the FORMAT-MODE control word is to restore SCRIPT/370 to CONCATENATE-JUSTIFY mode after you have entered a NO-FORMAT-MODE, NO-JUSTIFY-MODE, or NO-CONCATENATE-MODE control word. The FORMAT-MODE control word is simply a convenient abbreviation for the combination of CONCATENATE-MODE and JUSTIFY-MODE. The majority of this manual has been printed in the CONCATENATE-JUSTIFY mode.

JUSTIFY-MODE (.ju)

The JUSTIFY-MODE control word specifies that output lines are to be filled with blanks to justify the right-hand margins of text. JUSTIFY-MODE is implied in the default FORMAT-MODE of SCRIPT/370 processing, and is used to restore right justification after the NO-JUSTIFY-MODE control word (described below) is used, or to specify justification of output lines without concatenation if NO-FORMAT-MODE (also described below) is in effect.

The format of the JUSTIFY-MODE control word is

```
.ju
```

CONCATENATE-MODE (.co)

The CONCATENATE-MODE control word specifies that output lines are to be formed by shifting words to or from the next input line. The resulting line is as close to the specified line length as possible without exceeding it, splitting a word, or, as in FORMAT-MODE, padding the line with blanks; this resembles normal typist output.

The format for the CONCATENATE-MODE control word is

```
        .co
```

Output from this point on in the file is formed to approach the right margin without exceeding it.

NO-CONCATENATE-MODE (.nc)

A variation of justified text in CONCATENATE-JUSTIFY mode is NO-CONCATENATE mode. In NO-CONCATENATE mode SCRIPT/370 does not shift words back and forth between input lines. Instead, blanks are inserted as necessary to make each individual input line extend the full distance between margins. You might find this mode of operation useful when you have tabular material in which the elements are of uniform length such as number conversion tables. By specifying NO-CONCATENATE mode, you automatically achieve uniform spacing between the elements of each input line without the necessity of using tabs. You might also use NO-CONCATENATE mode in conjunction with the Or character (|) to produce vertical rules.

The format of the NO-CONCATENATE-MODE control word is

```
                            .nc
```

To restore concatenation of input lines, you enter a CONCATENATE-MODE control word, or a FORMAT-MODE control word, as appropriate.

BREAK (.br)

Often, as when entering the first line of a new paragraph, you will need to suspend concatenation of input lines for just one line. As shown previously, one way to do this is to begin the line with one or more blanks or the tab stroke. You also can achieve the effect of

```
.nc
text line for new paragraph
.co
subsequent text
```

by using the BREAK control word.

The format of the BREAK control word is

```
.br
```

Enter the BREAK control word just ahead of the text that must start on a new line in the document. You need not enter a BREAK control word if the first line of the new paragraph begins with a blank or a tab. Many SCRIPT control words also cause a BREAK to occur automatically; these are noted in Appendix B. In general they are ones used between paragraphs to specify the format of output text.

RAGGED-RIGHT OUTPUT

<u>NO-JUSTIFY-MODE</u> (<u>.nj</u>)

If you enter a CONCATENATE-MODE control word, SCRIPT/370 will shift words between lines to fill the current line length, but it <u>will</u> <u>not</u> insert blanks. Thus, by specifying CONCATENATE-MODE, you cause SCRIPT/370 to generate <u>ragged-right</u> output. Another method of achieving ragged-right output is to enter a NO-JUSTIFY-MODE control word. The NO-JUSTIFY-MODE control word causes SCRIPT/370 to stop inserting blanks so as to achieve the full line length. If CONCATENATE mode is still in effect, the output is ragged right. If CONCATENATE mode is not in effect, the output is as-is.

The format of the NO-JUSTIFY-MODE control word is

```
.nj
```

If you have entered a NO-JUSTIFY-MODE control word, you can cause SCRIPT/370 to resume justification by entering a JUSTIFY-MODE control word or, if you require CONCATENATE-JUSTIFY operation, the FORMAT-MODE control word which was discussed previously.

AS-IS OUTPUT

NO-FORMAT-MODE (.nf)

The format of the NO-FORMAT-MODE control word is

```
.nf
```

The NO-FORMAT-MODE control word is your means for causing as-is output. By entering .nf, you cause SCRIPT/370 to stop both concatenating and line justification. You will find this mode of operation especially useful if portions of your document must contain tabular material, or literal representations and illustrations. For example, the "Sample Terminal Session" in the first section of this manual was done primarily in no-format mode. Figure 1 below illustrates the difference between ragged-right and formatted output. With as-is output the output would look exactly like input text lines, that is, the break control word would not print.

RIGHT-ADJUSTED OUTPUT

RIGHT-ADJUST (.ri)

A single SCRIPT/370 control word allows you to specify right-adjusted, or flush right, output. Entering a RIGHT-ADJUST control word causes succeeding input lines to be printed even with the right margin as is commonly done in the sender's address of a letter. The input lines are not concatenated.

The format of the RIGHT-ADJUST control word is

```
.ri on|off|n
```

```
INPUT:

 +---------------------------------------+
 | aaaa bb c dddddd ee ffff ggggg hhhh   |
 | i jjjjj kk llll mm                    |
 | nnn oooooooooo                        |
 | .br                                   |
 | ppppp qqqq r                          |
 | ssss ttttttt uuu vvvvv                |
 | w xxxxxxxxx yy zz                     |
 +---------------------------------------+


using .ll 25, .co, and .nj           using .fo

 +------------------------------+     +------------------------------+
 |  aaaa bb c dddddd ee ffff    |     | aaaa  bb c dddddd ee ffff    |
 |  ggggg hhhh i jjjjj kk       |     | ggggg  hhhh  i  jjjjj  kk    |
 |  llll mm nnn oooooooooo      |     | llll mm nnn oooooooooo       |
 |  ppppp qqqq r ssss ttttttt   |     | ppppp qqqq r ssss ttttttt    |
 |  uuu vvvvv w xxxxxxxxx yy    |     | uuu  vvvvv w xxxxxxxxx yy    |
 |  zz                          |     | zz                           |
 +------------------------------+     +------------------------------+
```

Figure 1. Text Formatting

on causes subsequent input lines to be printed flush right.

off restores whatever mode of operation was in effect at the time .ri on was specified.

n causes the next n lines to be printed flush right. This can be used as an alternate to the combination .ri on, .ri off when the number of lines to be right adjusted is known.

Specifying the control word .ri by itself is equivalent to specifying .ri 1, that is, the next line of input text will be right adjusted on output.

CENTERED OUTPUT

<u>CENTER</u> (<u>.ce</u>)

A single SCRIPT/370 control word allows you to specify centered output. Entering a CENTER control word causes succeeding input lines to be printed equidistant from the left and right margins. The input lines are not concatenated.

The format of the CENTER control word is

```
.ce on|off|n
```

on	causes subsequent input lines to be centered until a .ce off is encountered.
off	restores whatever mode of operation was in effect at the time .ce on or .ce n was specified.
n	causes the next n lines to be centered. When you know in advance how many lines you require to be centered, using .ce n is more convenient and relieves you of having to remember to enter a .ce off control word.

Specifying the control word .ce by itself is equivalent to specifying .ce 1, that is, the next line of input text will be centered on output.

SUPPLEMENTAL PAGE FORMATTING

LINE SPACING

SCRIPT/370 provides you with three control words that allow you to determine spacing between lines of body text. Using these control words you can cause double- or single-spacing, or cause a given number of blank lines to be inserted between two lines of text so that, for example, illustrations can be introduced in the reproduction copy of your document.

SPACE-LINES (.sp)

With the SPACE-LINES control word you can cause a specified number of blank lines to be inserted in the page before the next line of body text is printed. The SPACE-LINES control word is especially useful following a paragraph heading because it generates a break in addition to inserting the specified number of blank lines.

The format of the SPACE-LINES control word is

```
.sp n
```

n is the number of blank lines to be inserted, _unless_ a DOUBLE-SPACE-MODE or LINE-SPACING control word is in effect. If DOUBLE-SPACE-MODE is in effect, _twice_ the number of lines you specify will be inserted. If LINE-SPACING is in effect, the number of lines you specify in the SPACE-LINES control word will be multiplied by the line-spacing increment you have specified in the LINE-SPACING control word.

SINGLE-SPACE-MODE (.ss)

If you have entered a DOUBLE-SPACE-MODE control word or a LINE-SPACING control word specifying other than single-spacing, you can cause single-spacing to resume by entering a SINGLE-SPACE-MODE control word. The SINGLE-SPACE-MODE control word takes effect immediately.

The format of the SINGLE-SPACE-MODE control word is

```
.ss
```

<u>DOUBLE-SPACE-MODE</u> (<u>.ds</u>)

To cause one blank line to be inserted between each line of body text, you enter a DOUBLE-SPACE-MODE (.ds) control word. The DOUBLE-SPACE-MODE control word takes effect immediately. It remains in effect until you enter a SINGLE-SPACE-MODE control word or a LINE-SPACING control word that specifies other than double-spacing.

The format of the DOUBLE-SPACE-MODE control word is

```
.ds
```

<u>LINE-SPACING</u> (<u>.ls</u>)

Occasionally, you may want your document or some portion of it to be printed with line spacing greater than double- or single-spacing. You may require triple-spacing for a review copy you expect to be proof-marked, or perhaps you might wish to leave space for writing in answers to questions on a test or inserting reproduction copy from another source. Rather than entering a SPACE-LINES control word before each line of text, you can enter a LINE-SPACING (.ls) control word specifying the number of blank lines to be inserted after <u>each</u> line of body text.

The format of the LINE-SPACING control word is

```
.ls n
```

n	is the number of blank lines to be inserted after <u>each</u> line of body text. This is the value which will be used by SCRIPT/370 as the multiplier in any subsequent SPACE-LINES control word you enter.

PAGE SPACING

Frequently, you may require a new page to be started even though space may remain on the current page. Conventionally, new sections or chapters begin on new pages, for example. You may want to insert an illustration that will not fit in the space remaining on the current page, or you may deliberately want a blank page so that the the back-up page (reverse side) can be removed from your document. SCRIPT/370 provides four control words so that you can cause a new page to be started under such circumstances. Three of these control words also include control of page numbering for the page to be started.

PAGE-EJECT (.pa)

To cause a new page to be started unconditionally, you enter a PAGE-EJECT control word. If you are using continuous-form paper, the form is advanced to the top of the next page immediately. If you are using individual sheets, the form is ejected, and SCRIPT/370 pauses to allow you to insert a fresh sheet, providing you have specified the STOP (ST) option when you entered the SCRIPT command.

The PAGE-EJECT control word also allows you to specify the page number for the new page, or apply an increment or decrement based on the number of the current page.

The format of the PAGE-EJECT control word is

```
.pa n|+n|-n
```

n	is the page number to be printed on the new page
+n	is the increment to be added to the current page number
-n	is the decrement to be subtracted from the current page number

When SCRIPT/370 encounters the PAGE-EJECT control word, it prints the bottom title (if applicable) on the current page, and advances to the next page, or pauses, as described above. You will probably use the PAGE-EJECT control word often in conjunction with one or another of the top title control words. Because the top title control words do not take effect until the next page, you must follow one with a PAGE-EJECT control word if you want to force the beginning of a new top title.

CONDITIONAL-PAGE-EJECT (.cp)

Sometimes, rather than having a paragraph or table be split between two pages, you will require that a new page be started for it. Because the amount of text preceding the paragraph or table varies as the document is developed or revised, SCRIPT/370 provides you with the CONDITIONAL-PAGE-EJECT control word. This control word causes a new page to be started only if the current page contains fewer lines than you require between the current line and the bottom margin.

The format of the CONDITIONAL-PAGE-EJECT control word is

```
.cp n
```

n	is the number of lines that must remain between the current line and the first line of the bottom margin if printing is to continue on the current page.

EVEN-PAGE-EJECT (.ep) and ODD-PAGE-EJECT (.op)

In certain kinds of documents you may require that narrative for an illustration or table appear on an even-numbered page (left-hand page) facing the illustration. Often, as mentioned above, you will require that new sections begin on an odd-numbered page (right hand page). To meet these and similar requirements, SCRIPT/370 provides you with two control words. You may enter an EVEN-PAGE-EJECT or ODD-PAGE-EJECT control word at any point in your document. Like the PAGE-EJECT control word, these operate immediately and unconditionally.

The format of the EVEN-PAGE-EJECT control word is

```
.ep
```

The format of the ODD-PAGE-EJECT control word is

```
          .op
```

The following information is about the EVEN-PAGE-EJECT control word for convenience of explanation. You may read it as applying equally to the ODD-PAGE-EJECT control word simply by substituting odd for even, and vice versa, wherever they appear.

When you enter an EVEN-PAGE-EJECT control word, the result is one page eject if the current page is odd-numbered, and two page ejects if the current page is even-numbered. Thus, the new page will be even-numbered. If the page you terminate with the EVEN-PAGE-EJECT control word was odd-numbered, then naturally the _next_ page is even-numbered and SCRIPT/370 resumes printing body text on that page. If the page you terminated was even-numbered, so that the next page must be odd-numbered, SCRIPT/370 prints only top and bottom titles on that page and resumes printing body text on the even-numbered page that follows.

MARGIN MODIFICATION

In preparing a document, you will probably require that the location of the effective left margin be changed from time to time. You may require that certain paragraphs be indented from the left margin, that certain text be printed with hanging indent, or that a block of formatted text be centered. SCRIPT/370 provides four control words that simplify these tasks for you. With them you can specify indent amounts, temporary additions or subtractions from the indent, and tab stops. These are described below.

INDENT (_.in_)

With an ordinary typewriter, you probably would use the TAB key for hanging indents. Using SCRIPT/370, however, you need only enter a single INDENT control word, and continue entering the text that is to be indented.

The format of the INDENT control WORD is

```
.in n|0
```

n specifies the number of spaces succeeding text is to be indented from the left margin.

0 restores the left margin.

Although the INDENT control word does not affect the right margin, by using it in conjunction with the LINE-LENGTH control word, you can cause blocks of formatted text to be centered or shifted within the margins. For instance, the following control words

```
.in 5
.ll 55
```

preceding text in this document would result in text which is centered, as

> This text line was deliberately included and made longer than 55 characters on input so as to illustrate the SCRIPT/370 INDENT control word used for centering of format-mode output.

OFFSET (.of)

Often, when indenting paragraphs, two indents are needed: one for a paragraph number or other designation, and a second for the paragraph text itself. SCRIPT/370 provides two methods for accomplishing this. In the first, using the OFFSET (.of) control word in conjunction woth the INDENT control word, you simulate the use of the tab key on an ordinary typewriter. The first line of a numbered paragraph will appear as if it had been typed in the sequence TABnTABtext. Succeeding lines will appear as if they had been typed in the sequence TABTABtext.

The format of the OFFSET control word is

```
.of n
```

n is the number of spaces to be added to the current margin or margin plus indent after the next line is printed. You may restore the margin or margin plus indent by omitting n or specifying it as 0.

Because the OFFSET control word's effect does not take place until _after_ the next text line, the first line of your paragraph will begin at the current left margin or margin plus indent; succeeding lines will begin at the margin plus indent plus offset, until you enter an additional offset or until you reset the offset to zero by entering .of 0, or .of. You can enter a series of paragraphs that uses the same offset simply by repeating the .of n before the first line of each paragraph; no intervening .of 0 is necessary. If you require a series of paragraphs with different offsets, you must enter .of or .of 0 before entering the new OFFSET control word.

To assure that SCRIPT/370 does not attempt to insert blanks between the paragraph number and the beginning of text, you should be sure that the TAB-SETTING control word in effect includes a tab for the print position at which the text is to begin. For instance, if your indent were five, and your offset were three, the TAB-SETTING control word in effect at that time must include print position 8 as one of its operands. For example, if you have set

```
.in 5
.of 3
```

then you should also have entered a TAB-SETTING control word that includes print position 8 (5+3), as explained below.

UNDENT (_.un_)

SCRIPT/370 provides you with another method of accomplishing hanging indents, through use of the UNDENT control word. The UNDENT control word works from the current indent towards the left margin, unlike OFFSET which works from the left margin towards the right. Also, the UNDENT control word operates on the line _immediately following_ it. Thus, you can specify an indent for the text to be printed hanging, and use the UNDENT control word _before_ the first line of each new paragraph so that the paragraph number is printed nearer the left margin.

The format of the UNDENT control word is

```
.un n
```

n   is the number of spaces the beginning of the next line is to be moved towards the left margin. It must be less than the current indent. Note that this requirement means that you cannot cause a line to begin to the left of the basic left margin, because the margin is equivalent to .in 0.

In general, the choice between using the UNDENT and OFFSET control words depends on your personal preference. It is important, however, not to use the UNDENT control word in portions of text that are offset, because the UNDENT control word operates with respect to the margin plus indent, not the margin plus indent plus offset.

TAB-SETTING (.tb)

When using SCRIPT/370 you will work with two kinds of tabs: the mechanical tab stops on your terminal, with which you are already familiar, and logical tabs, which SCRIPT/370 uses to format your output on a printer. To set tabs for your document, or a portion of it, proceed as follows

1. Clear the mechanical tabs on your terminal.

2. Set the mechanical tabs on your terminal as required.

3. Enter a TAB-SETTING (.tb) control word specifying the settings of your mechanical tab stops.

When you use the TAB key as you are entering your text, tab characters are generated which act as logical tabs when SCRIPT/370 recognizes them during output processing. SCRIPT/370 uses the information you have furnished it in the TAB-SETTING control word to convert these logical tabs into the appropriate number of blanks to simulate actual tab stops.

The format of the TAB-SETTING control word is

```
.tb n1 n2 n3...
```

n1... specifies the print position of the tab stop, not the number of spaces between it and the last tab stop. If you do not enter a TAB-SETTING control word, SCRIPT/370 chooses logical tab stops at print positions 5, 10, 15, etc., up to 75.

If you have entered a TAB-SETTING control word, you can restore the standard tabs simply by entering a TAB-SETTING control word with no stops specified.

By using the TAB key to begin a paragraph you accomplish two things at once: the first line of the paragraph is indented to the tab stop, and, because the tab begins the line an automatic break occurs, assuring that the formatted paragraph will begin on a new line.

SCRIPT/370 provides a second form of the TAB-SETTING control word that you will find especially useful for drawing lines, and for filling otherwise blank lines with a non-blank character. This tab fill facility permits you to enclose portions of text in boxes, generate periods in tables of contents, etc.

When used for this purpose, the format of the TAB-SETTING control word is

```
.tb n1 c1/n2 c2/n3...
```

n1...	specifies the print position of the tab stop
c1...	specifies the fill character
/n2...	specifies the tab stop at which the preceding fill character is to terminate

For example, specifying

```
.tb 5 +/15 -/25 */35
```

followed by the sequence TABxTAByTABzTABi, results in

```
x+++++++++y---------z*********i
```

INCORPORATING OTHER FILES

When developing long documents, or documents that you expect to revise frequently, it is good practice to establish a number of individual Script files that each contain a separate portion of your document. For instance, each chapter could be a separate file, and your keyboard-developed illustrations could be kept in a file of their own. Then, using the control words described in this section, you can establish a master file that combines the individual sections in any order you require.

In this way, you can reorganize your document if necessary, and add to it or delete from it with a minimum of effort. Different master files can be established so that different versions of your document can be printed simply by referring to the appropriate master file, without having to maintain separate (duplicate) files. This conserves storage space and relieves you of the necessity of making sure that changes are incorporated into duplicate files.

SCRIPT/370 provides you with five control words that permit you to combine files in different ways. Using these control words, you can insert all or part of a file into another file, add a file to the end of another file, and assure that a file is not inserted into another until a new page is begun in the receiving file.

IMBED (.im)

The IMBED control word is your principal means of inserting or including other Script files within the one you are developing. SCRIPT/370 does not limit the number of IMBED control words you may have within a file, but it does limit the number of "nested" IMBED control words to eight. (A nested imbedded file is one which has been imbedded in a file which is itself an imbed.)

You may use the IMBED control word anywhere within a document. When SCRIPT/370 encounters it, the file to be imbedded is inserted into the file containing the IMBED control word. At the end of the imbedded file SCRIPT/370 resumes printing the original file.

The format of the IMBED control word is

```
.im file
```

file is the name of the file that is to be imbedded. Script is assumed as the filetype

One useful application of the IMBED control word is that of including frequently-used sets of control words in a file. Using the IMBED control word, you need not repeat the sequence; you need only enter an IMBED control word referring to the file that contains the sequence.

The operation of SCRIPT/370 when you make use of imbedded files is shown below in Figure 2. Note that the master file contains the page and text formatting control that are to be in effect for the entire document. You will find the IMBED control word extremely useful whenever you are working with large documents. For example, the section "Multiple Column Processing" is a separate file, as is the section "Terminal Input/Output;" both are imbedded into the document from a master file.

The IMBED control word does not permanently change the file in which it is encountered. The imbedded file does not become part of the master file; it simply becomes part of your output.

Because SCRIPT/370 does not automatically perform a page eject at the end of the imbed, the n+1 line should be one of the page eject control words if you require that the remaining text in the receiving document begin on a new page.

DELAY-IMBED (.di)

You will find the DELAY-IMBED control word useful if you are working with a document which includes tables or diagrams smaller than a full page. Its purpose is to allow you to delay the inclusion of a portion of your Script input file until the next page eject occurs. DELAY-IMBED does not force a page-eject. Instead, SCRIPT/370 continues to place lines on the current page. When a new page is begun, the portion of your file which you specified as delayed is printed.

The format of the DELAY-IMBED control word is

```
.di n|ON|OFF
```

n	is the number of input lines saved for processing at the top of the next page by SCRIPT/370. If n is omitted, 1 is assumed. These input lines may contain text and/or control words.
ON	specifies that all input lines following the control word are to be delayed. Used with ".di off."

```
UNFORMATTED OUTPUT

      filea
┌─────────────┐
| aaaaaaaaaaa |               fileb
| aaaaaaaaaaa |         ┌─────────────┐               filec
| .im fileb   |──────>| bbbbbbbbbbb |          ┌─────────────┐
| aaaaaaaaaaa |<──────| .im filec   |──────>| ccccccccccc |
| aaaaaaaaaaa |         | bbbbbbbbbbb |<──────| ccccccccccc |
| aaaaaaaaaaa |         | bbbbbbbbbbb |          └─────────────┘
| aaaaaaaaaaa |         └─────────────┘
| aaaaaaaaaaa |
| .di 2       |               filed
| .sp 3       |         ┌─────────────┐
| .im filed   |──────>| dddddddddd  |
| aaaaaaaaaaa |<──────| dddddddddd  |
| aaaaaaaaaaa |         └─────────────┘
| aaaaaaaaaaa |
| aaaaaaaaaaa |
└─────────────┘


FORMATTED OUTPUT

┌─────────────┐         ┌─────────────┐
|             |         |             |
| aaaaaaaaaaa |         |             |
| aaaaaaaaaaa |         |             |
| bbbbbbbbbbb |         |             |
| ccccccccccc |         | dddddddddd  |
| ccccccccccc |         | dddddddddd  |
| bbbbbbbbbbb |         | aaaaaaaaaaa |
| bbbbbbbbbbb |         | aaaaaaaaaaa |
| aaaaaaaaaaa |         | aaaaaaaaaaa |
| aaaaaaaaaaa |         | aaaaaaaaaaa |
| aaaaaaaaaaa |         | aaaaaaaaaaa |
| aaaaaaaaaaa |         |             |
|         -1- |         | -2-         |
└─────────────┘         └─────────────┘
```

Figure 2. IMBED/DELAY-IMBED Handling

OFF marks the end of the set of input lines being delayed. Used with ".di on."

The operation of the DELAY-IMBED control word is shown in Figure 2. This control word can be used to ensure that a table or illustration begins at the top of the page following the first reference to it in the text. As indicated in the figures, DELAY-IMBED is often used in combination with the IMBED control word.

APPEND (.ap)

Another method of organizing the Script files containing your document is to end each individual file with an APPEND control word. This method differs from using the IMBED control word in that SCRIPT/370 does not resume processing the first file after it has processed the file named in the APPEND control word. When SCRIPT/370 encounters the APPEND control word, it terminates processing the original file, and begins processing the file to be appended as though it were a continuation of the original file. Unless the appended file itself ends with an APPEND control word, SCRIPT/370 returns control to CMS at that point.

The format of the APPEND control word is

```
.ap filename
```

filename is the name of the file to be appended to the current file

END-OF-FILE (.ef)

You can simulate an end-of-file within a file by using the END-OF-FILE control word. You will find this capability especially useful for dividing into segments a file that is to be imbedded in portions.

For example, you may have developed a Script file containing illustrations for an entire document. However, the illustrations are to appear interspersed with different portions of the text, rather than all together. By entering an END-OF-FILE control word after each illustration, you cause SCRIPT/370 to terminate the imbed rather than continue to the end of the illustration file. Thus, in your text file you simply enter an IMBED control word at each point you wish to insert the next illustration. If it is the first IMBED control word referring to that file, SCRIPT/370 begins at the top of the file and imbeds each successive line until it encounters an actual end-of-file condition, or until it encounters an END-OF-FILE control word. The next IMBED control word in your text file that refers to the illustration file begins inserting the illustration file at the point where it left off, immediately following the END-OF-FILE control word.

The format of the END-OF-FILE control word is

```
                    .ef
```

An illustration of the use of the IMBED, DELAY-IMBED, and END-OF-FILE control words is shown in Figure 3.

QUIT (.qu)

With the QUIT control word, you can cause processing of your file to be terminated immediately. You may use the QUIT control word in conjunction with the END-OF-FILE control word to control the printing of form letters, or in conunction with the TERMINAL-INPUT and TYPE-ON-TERMINAL control words to terminate processing of a document from the terminal. When SCRIPT/370 encounters a QUIT control word, even in a file that is being imbedded, it advances the form to the top of the next page, and ends processing.

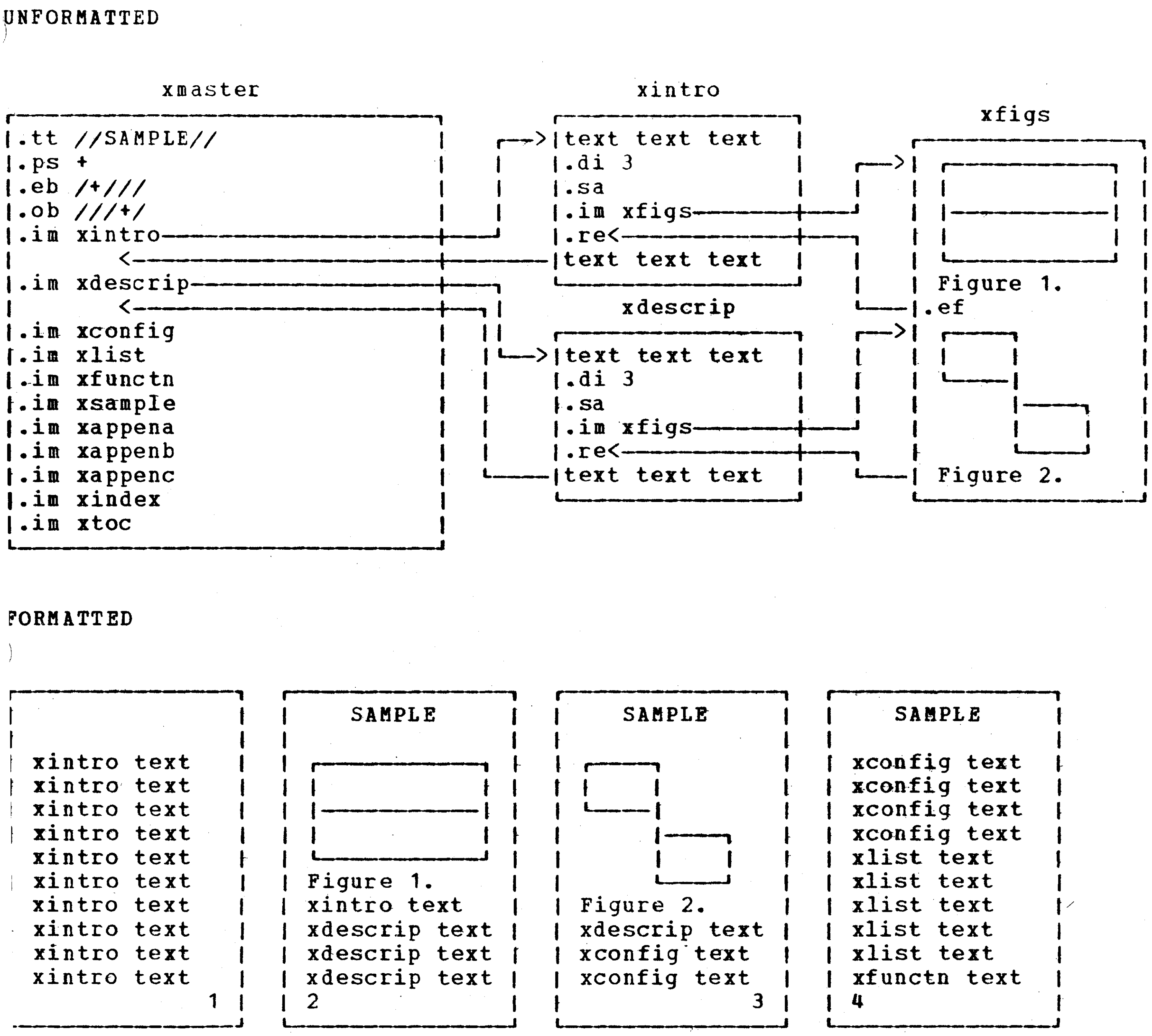
The format of the QUIT control word is

```
                    .qu
```

An example showing a technique for printing personalized form letters using QUIT, IMBED, END-OF-FILE, and APPEND can be found in the section "Using Set Symbols."

PRESERVING FILE STATUS

When you cause another file to be included in your output by use of the IMBED or DELAY-IMBED control words, the setup values of the imbedded files remain in effect after the return to the receiving file until new control words are encountered. Often, this is undesirable. For instance, if the file you imbedded used double-spacing, it remains in effect for the remainder of the file in which you imbedded it, even though you have specified single-spacing at some point before the IMBED. You could prevent problems of this



'igure 3. Master File Using IMBED, DELAY-IMBED, and END-OF-FILE kind by keeping track of all your setup values and entering the appropriate control words to reset them after each IMBED or DELAY-IMBED, but this would be extremely inconvenient. Frequently, it is difficult to know what the original settings were.

To prevent the problem SCRIPT/370 provides you with two control words that relieve you of the necessity to keep track of the values in effect at the time of the IMBED or DELAY-IMBED. The SAVE-STATUS control word causes SCRIPT/370 to store the values; the RESTORE-STATUS control word allows you to put them back in effect.

<u>SAVE-STATUS</u> (<u>.sa</u>)

To insure that you will be able to restore your current control values after an imbedding a file, you must enter a SAVE-STATUS control word before entering the IMBED or DELAY-IMBED control word. This causes SCRIPT/370 to store your current values such as margin and tab settings, line length, text format, etc. It does <u>not</u>, however, store your top and bottom titles. If you wish to resume printing of the same top and bottom titles that were in effect at the time of the imbed, you must reenter appropriate control words at the point where SCRIPT/370 resumes processing the original file.

The format of the SAVE-STATUS control word is

```
        .sa
```

When SCRIPT/370 encounters the SAVE-STATUS control word, it stores the settings of the following control words

.CE	CENTER
.CO	CONCATENATE-MODE
.DS	DOUBLE-SPACE-MODE
.FI	FILL-MODE
.FM	FOOTING-MARGIN
.FO	FORMAT-MODE
.HM	HEADING-MARGIN
.IN	INDENT
.JU	JUSTIFY-MODE
.LI	LITERAL
.LL	LINE-LENGTH
.NC	NO-CONCATENATE-MODE

.NF	NO-FILL-MODE
.NF	NO-FORMAT-MODE
.NJ	NO-JUSTIFY-MODE
.OF	OFFSET
.PL	PAGE-LENGTH
.PN	PAGE-NUMBERING-MODE
.SS	SINGLE-SPACE-MODE
.TB	TAB-SETTING
.TM	TOP-MARGIN
.UN	UNDENT

The SAVE-STATUS control word does not change the settings of any of these; it merely stores them so that they can be reset later. Because of this, you may find it necessary to set certain values explicitly if you do not know what they are at the time you enter the SAVE-STATUS control word. For example, you may want to set your indent value to zero in the file to be imbedded. Remember also that SCRIPT/370 may encounter subsequent SAVE-STATUS control words in the imbedded files before reaching the RESTORE-STATUS control word corresponding to the SAVE-STATUS control word you entered. As many as five nested save-restore sequences are permitted.

RESTORE-STATUS (.re)

As mentioned above, the RESTORE-STATUS control word resets control word values in accordance with those stored by the corresponding (preceding) SAVE-STATUS control word. A RESTORE-STATUS control word that does not have a corresponding SAVE-STATUS control word causes an error message to be printed during output.

The format of the RESTORE-STATUS control word is

```
.re
```

An example of use of SAVE-STATUS and RESTORE-STATUS is shown by the following sequence.

```
  ...
.DELAY-IMBED ON
.SAVE-STATUS
.INDENT 0
.TAB-SETTING 5 15
```

```
.SINGLE-SPACE-MODE
.SPACE 2
.CENTER
Table III.
.SPACE
The following table shows the legal commands:
     FORTRAN   FORTRAN compiler
     PLI  PL/I Compiler
  ...
  ...
.PAGE-EJECT
.RESTORE-STATUS
.DELAY-IMBED OFF
  ...
```

The material from the DELAY-IMBED ON to the DELAY-IMBED OFF will be activated at the end of the current page of SCRIPT output. When this section is activated, it first saves the SCRIPT status. Since it is not known whether an INDENT or OFFSET was in affect, the INDENT 0 ensures that printout will start at column 1. Likewise, since DOUBLE-SPACE-MODE may have been in affect, the SINGLE-SPACE-MODE control word insures that single spacing will occur for this table. The tab setting is specially set for the tabular information. At the end of the table, the PAGE-EJECT control word forces printout to continue on a new page. The RESTORE-STATUS restores the SCRIPT status variables, in particular the INDENT, SINGLE/DOUBLE SPACE-MODE, and TAB-SETTING, to their original values. The DELAY-IMBED OFF terminates the imbed section.

ADDITIONAL FACILITIES

SCRIPT/370 includes many facilities for increasing the usability of your document in both final and unformatted form. Through the control words described in this section, you can

- cause selective printing of anything from a single word to an entire document
- indicate as many as nine different revision codes at the left margin
- cause characters entered from a terminal to be translated when your document is printed offline
- cause lines beginning with a period (.) to be treated as literal text rather than as control words
- include comments that will not appear in the final document
- change the symbol used to separate successive control words that are entered on the same line

CONDITIONAL-SECTION (.cs)

To cause only selected portions of your document to be printed, you associate a conditional section code with the portion of the document you require to be controlled. You do this through the CONDITIONAL-SECTION control word. Through this facility you have the means for printing only unclassified portions of a confidential or proprietary document or including information on different versions of a system or procedure in the same document, and having only the information pertaining to a given version printed. Because the CONDITIONAL-SECTION control word does not cause a break, you can even use it within a sentence to cause certain word to be included or ignored, as required.

The format of the CONDITIONAL-SECTION control word is

```
.cs n on|n off
```

and

```
.cs n include|n ignore
```

n	represents the conditional section number, and may be any digit from one to nine
on	specifies that the conditional section n indicator be set
off	specifies that conditional section n indicator be reset
include	specifies that all following conditional sections with code number 'n' are to be included
ignore	specifies that all following conditional sections with code number 'n' are to be ignored

The first three operands are the ones you will use when entering text to define which portions of text are associated with a given conditional section. The material within a conditional section may include any SCRIPT control word, text, or both. For example, the sequence

```
This manual contains
.cs 1 on
three
.cs 1 off
.cs 2 on
five
.cs 2 off
sections.
```

associates the word three with conditional section 1, and the word five with conditional section 2. When you want to print your document at some later time, and you want to use the version that will refer to there being five sections, you simply edit the document or your master file to insert the following

```
.cs 1 ignore
.cs 2 include
```

and invoke the SCRIPT command specifying the appropriate filename. The resultant output will be

```
This manual contains five sections.
```

In the same manner, you could develop a document such as a combined course and instructors guide by defining all instructor-oriented material as a conditional section, and all student-oriented material as a second conditional section. You would then be able to print three different versions of the document:

1. One for students, that contained only student-oriented material

2. One for instructors that contained both student- and instructor-oriented material

3. A second version for instructors, that contained only instructor-oriented material

REVISION-CODE (.rc)

When making changes to a published document you may need to distinguish in the final copy between original and new material. Often, this is done by printing a bar or other symbol adjacent to the new text. SCRIPT/370 provides you with the capability of printing as many as nine different symbols, representing nine different levels of revision. In addition, because the blank is a valid revision symbol, you may suppress printing of some or all revision symbols simply by redefining them as blanks.

When SCRIPT/370 finds that it must print revision indicators it accommodates them by shifting body text three spaces to the right. This allows the revision indicator to be printed at the left margin with two spaces between it and the beginning of the text line. If an indent is in effect at that point, then, naturally, the number of spaces between the revision indicator and the text will be greater. Because of this shift to the right you should define your revision code at the beginning of your document so that it will be printed uniformly. If you must define a revision code _within_ the document, you should do so immediately following a page eject so that the entire page will be indented uniformly.

For defining a revision indicator, the format of the REVISION-CODE control word is

```
.rc n s
```

n   is the revision number, and must be between 1 and 9

s   is the revision indicator to be associated with the revision number specified by n. It may be any single character, including blank

Having defined your revision codes, you use a different form of the control word for associating them with given portions of your document, similarly to the CONDITIONAL-SECTION control word. For this purpose, the format of the REVISION-CODE control word is

```
.rc n on|n off
```

n on   causes revision number n to be associated with all subsequent text until a corresponding .rc n off is encountered.

n off   terminates the association of revision number n with text.

It is important when using more than one revision code within a document, as when you are working on a second or later revision, that you always turn OFF your revision codes in the reverse order from which you turned them ON. You may turn ON the revision codes in any order, but you must always turn them OFF in the reverse order

For example:

```
.rc 3 on
i
text...
.rc 1 on
text...
.rc 2 on
text...
.rc 2 off
text...
.rc 1 off
text...
.rc 3 off
```

Of course, you may have control words and/or intervening text at the point where you turn OFF the revision codes.

At times, you will require that a revision code apply to a single line or a portion of a line. To do this, you need not use separate ON and OFF control words. Instead, you may use the REVISION CODE control word in the following format

```
        .rc n on/off
```

This causes SCRIPT/370 to turn on revision code n immediately, and turn it off at the end of the next text line. You will find this helpful, because it relieves you of the necessity for keeping track of ON/OFF sequences. In practice, it is more reliable to use this method than to break up your text line with the REVISION-CODE control words. Using the CONTROL-WORD-SEPARATOR control word (described below), you can even enter the REVISION-CODE control word and the text on the same line.

CONTROL-WORD-SEPARATOR (.cw)

To save typing SCRIPT/370 allows you to enter more than one control word, or a series of control words and text, on a single line. You do this by separating the control words from each other and from the text on that line with a character recognized by SCRIPT/370 for that purpose. Normally, the semicolon(;) is the control word separator character. Because you may want to change this in a given document, SCRIPT/370 provides you with the CONTROL-WORD-SEPARATOR control word.

The format of the CONTROL-WORD-SEPARATOR control word is

```
        .cw c
```

c   is the chracter to be used in place of the semicolon as a separator. If c is omitted, you may not enter more than one control word per line, and may not combine control words and text on a line.

This facility is especially useful for such sequences as

```
     .sp 2;.of 5;This section contains the following
```

and

```
     .rc 2 on/off;This line has been revised a second time.
```

The FIND subcommand of the CMS Editor operates only on information beginning in terminal position one. Therefore, if the control word you wish to CHANGE or DELETE is not the first one in a line containing a series of control words, you will have to use the LOCATE subcommand.

COMMENT (.cm)

With the COMMENT control word you can include notes to yourself or special references in your document. These comments are seen when you are editing your document, or printing it using the UNFORMAT option of the SCRIPT command. They do not appear in formatted output. Thus using the COMMENT control word you can include reminders to yourself or reviewers to fill in a date, suppress revision indicators for final printing, etc.

The format of the COMMENT control word is

```
        .cm text
```

text is the comment which is to appear in unformatted output. If the comment exceeds one line, simply enter a COMMENT control word at the beginning of each subsequent line.

LITERAL (.li)

Because SCRIPT/370 control words begin with a period (.), you may encounter problems with your output if you begin a text line with a period. There may be occasions, however, when you need to do this, such as when entering a number preceded by a decimal point, or beginning a line with an ellipsis. There are many ways to avoid the problem by rearranging text, rewording it, or using TAB or blank to begin the line. The former are inconvenient and the latter may be incompatible with your format requirements. Using the LITERAL control word, you may begin any number of subsequent input lines with a period and have them interpreted by SCRIPT/370 as the text lines they actually are.

The format of the LITERAL control word is

```
        .li n
```

n is the number of subsequent lines to be processed as text even if they begin with a period. If you omit n, only the next line is affected.

TRANSLATE-CHARACTER (.tr)

Often, text that is entered from a terminal is to be printed OFFLINE on a printer equipped with the TN chain, which contains both upper and lower case letters and a number of special characters, superscripts, etc. Many of the special characters of the TN train are not available to you at the keyboard of your terminal. However, using the TRANSLATE-CHARACTER control word, you can use a keyboard character in place of a special character and have SCRIPT/370 interpret this usage at the time your output is printed. At that time, the appropriate character is substituted for the keyboard character.

The format of the TRANSLATE-CHARACTER control word is

```
.tr i o
```

i	is the input (keyboard) character to be used as a representation of the desired output character. It may be a character or a two-digit hexadecimal number (see the table below).

o	is the desired output character which SCRIPT/370 substitutes for the input character specified by i. It may be a character or a two-digit hexadecimal number.

After formatting of an input source line has been completed and immediately prior to actual output, each character of the output line is analyzed for possible translation to a different output code. Consequently, the TRANSLATE-CHARACTER control word is primarily of use when the final output device uses a character set different from that used to create the Script file.

Title lines are processed by SCRIPT/370 under control of the TRANSLATE specifications in effect at the time the title lines were entered. Thus, you can translate an input character in a title differently from the way you cause it to be translated in text.

Once you have entered a TRANSLATE-CHARACTER control word for a given input and output character combination, it remains in effect until you explicitly redefine it. You can, however, reset all your translation specifications to the original (default) settings, by entering a TRANSLATE-CHARACTERS control word in the form

```
        .tr
```

You need not specify the translations of lower-case alphabetic characters to upper-case for an entire document, since this can be accomplished automatically if you specify the TRANSLATE option of the SCRIPT command.

The hexadecimal code for each printable character is shown in the table below:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00																	00
10																	10
20																	20
30																	30
40											¢	.	<	(	+	|	40
50	&										!	$	*	)	;	¬	50
60	-	/										,	%	_	>	?	60
70											:	#	@	'	=	"	70
80		a	b	c	d	e	f	g	h	i		{	≤	(	+	┼	80
90		j	k	l	m	n	o	p	q	r		}	⌑	)	±	■	90
A0	−	°	s	t	u	v	w	x	y	z		└	┌	[	≥	•	A0
B0	^{0}	^{1}	^{2}	^{3}	^{4}	^{5}	^{6}	^{7}	^{8}	^{9}		┘	┐	]	≠	—	B0
C0		A	B	C	D	E	F	G	H	I							C0
D0		J	K	L	M	N	O	P	Q	R							D0
E0			S	T	U	V	W	X	Y	Z						A	E0
F0	0	1	2	3	4	5	6	7	8	9							F0
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Examples:

a. .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

will print as

$X^2+Y^2=Z^3$

b. .TRANSLATE-CHARACTER 40 ?
This causes all blanks in the file to be typed as questions marks (?) on output.

c. .tr $ 7B
This causes each occurrence of the character "$" to be

replaced by the hexadecimal character 7B, which is the "#" character. This may be necessary since the # character may have special significance to the CMS Edit facility.

MULTIPLE-COLUMN PROCESSING

Perhaps one of SCRIPT/370's most useful features is its ability to produce multiple-column output. Using the various multiple-column control words, you can determine the number of columns per page,their width and separation, and whether all columns on a page are to be the same length (balanced) or the last column will be allowed to be shorter than the others (unbalanced). Using these control words, you can produce output with as many as nine columns per page. This allows you to produce double-column output similar to that of many technical manuals, six- or eight-column output similar to that of most newspapers, or other formats which may be of special use in your installation.

COLUMN-DEFINITION (.cd)

With the COLUMN-DEFINITION control word, you specify the number of columns per page, and the beginning print position of each. When SCRIPT/370 encounters a COLUMN-DEFINITION control word, it processes all text up to that point in accordance with the old column definitions before putting the new definitions into effect.

The format of the COLUMN-DEFINITION control word is

```
.cd n p1 p2...
```

n is the number of columns to be printed on each subsequent page, and may be any number from 1 to 9.

p1 p2... is the starting print position of columns 1 through n. The physically leftmost position is designated position zero.

If you specify fewer than n starting positions, the previously defined values remain in effect for those not respecified.

COLUMN-LENGTH (.cl)

The COLUMN-DEFINITION control word specifies only the number of columns and the starting print positions. To prevent unintentional overlaps, you use the COLUMN-LENGTH control word to specify the number of print positions each column will occupy. To insure that you have space between columns, ("gutter"), you should set column length to occupy less space than that between defined column starting positions. The column length also is the reference for the operation of such control word as CENTER and RIGHT-ADJUST when you have specified multiple-column output.

The format of the COLUMN-LENGTH control word is

```
.cl n
```

n — is the number of print positions to be occupied by each column defined in the current COLUMN-DEFINITION control word.

If you have not entered a COLUMN-LENGTH control word, SCRIPT/370 sets column width equal to the current line length, and changes it in accordance with subsequent changes to line length.

COLUMN-BEGIN (.cb)

The COLUMN-BEGIN control word causes subsequent text to start a new column. If the COLUMN-BEGIN control word occurs within the last column on a page, SCRIPT/370 causes a page eject and begins the new column on a new page. When you use a COLUMN-BEGIN control word, the column in which it occurs remains unbalanced (fewer lines than the current page length), even if a BALANCE-COLUMNS control word is in effect.

The format of the COLUMN-BEGIN control word is

```
.cb
```

CONDITIONAL-COLUMN-BEGIN (.cc)

The CONDITIONAL-COLUMN-BEGIN control word operates similarly to the CONDITIONAL-PAGE-EJECT (.cp) control word.

The format of the CONDITIONAL-COLUMN-BEGIN control word is

```
.cc n
```

n is the number of lines that must remain between the current line and the first line of the bottom margin if printing is to continue in the current column.

If there are more than n lines remaining, this control word is ignored.

BALANCE-COLUMNS (.bc)

To cause all columns on a page to have the same number of lines, you enter a BALANCE-COLUMNS control word. When SCRIPT/370 encounters this control word, it processes all accumulated text to that point and formats it so that all columns occupy the same number of lines.. This allows you to end a section within a page, and resume multiple-column formatting on the same page after balancing the end of the first section.

The format of the BALANCE-COLUMNS control word is

```
.bc
```

You need not use the BALANCE-COLUMNS control word unless you wish to cancel a previous NO-BALANCED-COLUMNS control word.

NO-BALANCED-COLUMNS (.nb)

The NO-BALANCED-COLUMNS control word causes the lines printed as the result of a new COLUMN-DEFINITION control word or page eject to be printed without being balanced.

The format of the NO-BALANCED-COLUMNS control word is

```
.nb
```

The difference in effect between BALANCE-COLUMNS and NO-BALANCED-COLUMNS is shown below in Figure 4.

INPUT:

```
┌──────────────────┐
|aaaaaaa           |
|.sp               |
|bbbbbbb           |
|.sp               |
|ccccccc           |
|.sp               |
|ddddddd           |
|.sp               |
|eeeeeee           |
|.sp               |
|fffffff           |
|.sp               |
|ggggggg           |
|.sp               |
|hhhhhhh           |
|.sp               |
|iiiiiii           |
|.sp               |
|jjjjjjj           |
|.sp               |
|kkkkkkk           |
|.sp               |
|lllllll           |
|.sp               |
|mmmmmmm           |
|.sp               |
|nnnnnnn           |
|.sp               |
|ooooooo           |
|.sp               |
|ppppppp           |
|.sp               |
|qqqqqqq           |
|.sp               |
|rrrrrrr           |
|.sp               |
|sssssss           |
|.sp               |
|ttttttt           |
|.sp               |
|uuuuuuu           |
|.sp               |
|vvvvvvv           |
|.sp               |
|wwwwwww           |
|.sp               |
|xxxxxxx           |
|.sp               |
|yyyyyyy           |
|.sp               |
|zzzzzzz           |
└──────────────────┘
```

Figure 4. (Part 1 of 2)

FORMATTED OUTPUT:

Using .cd 3 0 11 22, .cl 7, and .bc (default)

```
┌────────────────────────────────────────┐
|                                        |
|   aaaaaaa               rrrrrrr        |
|             jjjjjjj                    |
|   bbbbbbb               sssssss        |
|             kkkkkkk                    |
|   ccccccc               ttttttt        |
|             lllllll                    |
|   ddddddd               uuuuuuu        |
|             mmmmmmm                    |
|   eeeeeee               vvvvvvv        |
|             nnnnnnn                    |
|   fffffff               wwwwwww        |
|             ooooooo                    |
|   ggggggg               xxxxxxx        |
|             ppppppp                    |
|   hhhhhhh               yyyyyyy        |
|             qqqqqqq                    |
|   iiiiiii               zzzzzzz        |
|                                        |
└────────────────────────────────────────┘
```

Using same column definition, .nb, and a .pl of 20

```
┌────────────────────────────────────────┐
|                                        |
|   aaaaaaa    mmmmmmm    wwwwwww        |
|                                        |
|   bbbbbbb    nnnnnnn    xxxxxxx        |
|                                        |
|   ccccccc    ooooooo    yyyyyyy        |
|                                        |
|   ddddddd    ppppppp    zzzzzzz        |
|                                        |
|   eeeeeee    qqqqqqq                   |
|                                        |
|   fffffff    rrrrrrr                   |
|                                        |
|   ggggggg    sssssss                   |
|                                        |
|   hhhhhhh    ttttttt                   |
|                                        |
|   kkkkkkk    uuuuuuu                   |
|                                        |
|   lllllll    vvvvvvv                   |
|                                        |
└────────────────────────────────────────┘
```

Figure 4. Balance versus Non-balance Multiple Column Output (Part 2 of 2)

USING MULTIPLE-COLUMN FACILITIES

For most users the separate .CD and .CL are cumbersome, thus it is recommended that SCRIPT set-symbol macros be used instead. For example, the symbol "&1col" could be assigned to mean "enter 1 column format" and "&2col" mean "enter 2 column format".

By creating an initialization file, named "$COLUMN SCRIPT" for example, which contains:

```
      .control-word-separator ?
      .page-length 84
      .column-definition 2 0 46
      .set 1col = '.cd 1;.cl 89;'
      .set 2col = '.cd 2;.cl 43;'
      .control-word-separator ;
      .substitute on
```

You can create text files using $COLUMN SCRIPT, as follows:

```
      .im $COLUMN
      &1col  (start in 1 column mode)
        ...
      &2col  (enter 2 column mode)
        ...
        ...
      &2col  (dump out columns
              but stay 2 column mode)
        ...
        etc.
```

WARNING: The Page-Length (.PL) should never be changed except when in one column mode, as is true at the very beginning of the SCRIPT file.

Also, the CENTER or NUMBER parameters to SCRIPT should only be used with very small explicit nn values since "standard" double column format takes 89 characters, leaving very little extra space on the line. Since the file identification space needed by NUMBER is about 16 characters, the parameter NUMBER00 requires:

89+2*16 = 121 characters

whereas the default value for NUMBER is NUMBER30 which requires:

89+2*46 = 181 characters

which won't fit on a printer line.

TERMINAL INPUT-OUTPUT

Using the terminal input/output control words you will find SCRIPT/370 very useful for form letters with variable information such as name and address, questionnaires, and making one-shot changes to documents SCRIPT/370 provides you with two control words that cause it to accept input from a terminal during output processing, and one that causes it to print information at the terminal that will not be processed as part of your document.

READ-TERMINAL (.rd)

With the READ-TERMINAL control word you can enter text lines from the terminal at the time of formatted output without their becoming part of your file, and without their being processed by SCRIPT/370. Thus, you could precede the date, recipient's name, and recipient's address lines of a "form" letter with READ-TERMINAL control words. During output processing of the file containing the form letter, SCRIPT/370 suspends processing when it encounters the READ-TERMINAL control words and unlocks the keyboard of your terminal so that you can type in the appropriate information. This informations will appear only on the individual page being printed at the terminal.

The format of the READ-TERMINAL control word is

```
.rd n
```

n is the number of lines to be entered from the terminal. If you omit a value for n, SCRIPT/370 allows you to enter one line and then resumes processing.

TERMINAL-INPUT (.te)

The TERMINAL-INPUT control word differs from the READ-TERMINAL control word in that the lines you enter from the terminal are passed to SCRIPT/370 and processed by it as though they were part of the file. Because of this, you may enter SCRIPT/370 control words as well as text. Thus, you can enter text and determine the format in which it is to be printed, just as though it were originally part of the file being processed.

The format of the TERMINAL-INPUT control word is

```
.te n|on|off
```

n	is the number of lines from the terminal to be accepted and processed. If you omit a value for n, SCRIPT/370 allows you to enter one line and then resumes processing, beginning with the line you entered.
on	specifies that file input is to be suspended, and terminal input to be accepted for processing until a TERMINAL-INPUT control word specifying off is entered from the terminal.
off	is entered from the terminal when you have completed entering your terminal input. This causes SCRIPT/370 to resume file input.

It is generally good practice to precede a TERMINAL-INPUT control word with a TYPE-ON-TERMINAL control word that prints an appropriate prompting message at the terminal. This allows you to instruct the terminal user as to what input is expected or what will be the results of different input. (The TYPE-ON-TERMINAL control word is described later in this section.) The TERMINAL-INPUT control word should not be used unless you have specified the FILE or OFFLINE option of the SCRIPT command. If you use it when printing at the terminal, the input lines you enter <u>and</u> the formatted output corresponding to them will appear in your output unless you avoid it manually by using separate sheets of paper. Remember also that the lines you enter from the terminal do not become part of the file. To make permanent changes to your file you must Edit it in the normal manner.

Because the TERMINAL-INPUT control word allows you to enter control words, you can use it to "tailor" the processing of a document. For example, by entering appropriate REVISION-CODE or CONDITIONAL-SECTION control words you can actually control which portions of a document are to printed, and which revision indicators are to be printed. You could also, by entering an IMBED control word, cause an entire file to be included for processing. If you do not receive a prompting message, and do not know if only one line will be accepted, you can enter a TERMINAL-INPUT control word specifying on so that SCRIPT/370 will continue to accept input from the terminal until you enter a TERMINAL-INPUT control word specifying off.

TYPE-ON-TERMINAL (.ty)

With the TYPE-ON-TERMINAL control word you can include information in your file that will be printed on the terminal during output processing but will not be part of your document. Of course if you are printing your document at the terminal, the line associated with each TYPE-ON-TERMINAL control word will appear interspersed with the rest of your document. Therefore, you should not use this control word unless you are planning on OFFLINE output, or unless you intend to prompt or question the person at the terminal.

The format of the TYPE-ON-TERMINAL control word is

```
.ty text
```

text is the line to be printed at the terminal. If more than one line is to be printed, simply precede each with another TYPE-ON-TERMINAL control word.

This control word can be used for informing the person who is printing or editing your document of tab settings, imbedded files, etc. When used in conjunction with the TERMINAL-INPUT control word, you effectively develop a "questionnaire" mode of operation. Remember, you must use the TERMINAL-INPUT control word if you want the lines entered from the terminal to become part of a printed document. If all you require is copy at the terminal, you can use the READ-TERMINAL control word.

SET-SYMBOL AND MACRO FACILITIES

The set-symbol facilities of SCRIPT/370 in conjunction with the SUBSTITUTE-SYMBOL control word and the 2PASS option of the Script command provide an extremely powerful method of generating tables of contents, assigning section numbers, and defining shorthand symbols to substitute for special phrases or control word sequences. You use the SET-SYMBOL control word to define a symbol and the name of the value to be substituted for it. You use the SUBSTITUTE-SYMBOL control word to determine whether the current value is substituted for the symbol in a given portion of your document. Thus you could define a symbol called chap1 with a value of & (pagenumber). Later, in your table of contents, you could type in chap1 as the page number associated with the heading of Chpater 1 . SCRIPT/370 would then substitute the actual page number for the symbol. Many more involved applications of these facilities are possible, but they are all based on the type of action described above. Additional details describing the set-symbol facilities are in Appendix C.

SET-SYMBOL (.se)

To define a symbol that will be used later in conjunction with the 2PASS option, you use the SET-SYMBOL control word. This control word can also be used to define groups (arrays) of symbols, so that you can cause series of values to be substituted rather than just a single value. Such arrays are useful in generating indexes which have multiple references for each index entry.

The format of the SET-SYMBOL control word is

```
        .se symbol-designator=symbol-value
```

The symbol-designator may be any one of the following forms:

```
symbol1
symbol1()
symbol1(n)
symbol1(&symbol2)
```

where symbol1 is any string of up to 10 non-blank characters excluding the characters:

.	period	=	equal sign
+	plus sign	-	minus sign
*	asterisk	/	slash
(	left parenthesis	)	right parenthesis
'	quote mark	&	ampersand

n must be a non-negative integer and symbol2 must be a defined set-symbol with any integer value.

The symbol-value may be either a character string or an arithmetic expression, as follows:

A character string must be less than or equal to 14 characters, and may be of the form:

'character string'
or
string

where string does not contain any of the arithmetic operations or the character blank (e.g., the quote marks may be omitted if the string does not contain any of the special characters).

An arithmetic expression is of the form:

```
op0 operand1 op1 operand2 op2 operand3 etc.
```

where operand1, operand2, etc. are either integers (n), defined set-symbols (&symbol), or the special SET page number symbol (&); op1, op2, etc. are the arithmetic operators:

+	addition
-	subtraction
*	multiplication
/	division

op0, if present, may be either a unary + or - sign. Expressions are evaluated in the conventional adding-machine left-to-right order.

Each symbol-designator may be viewed as a single element of an array that can span the subscripts from symbol(-32768) to symbol(32767). In particular, symbol1 alone is a short-hand notation for symbol1(0). The symbol-designator

```
.SET-SYMBOL  symbol1() = ...
```

is a short-hand notation for the sequence

```
.SET-SYMBOL  symbol1 = &symbol1 + 1
```

```
.SET-SYMBOL   symbol1(&symbol1) = ...
```

which results in stepping one-by-one through the array with symbol1(0) used as an index counter. This is very useful for creating an array of references.

The number of unique symbol elements that may be defined is dependent upon the storage size of the user's virtual machine. This may be changed by the DEFINE command of VM/370, or by the operations department of your installation.

In symbol names, upper case and lower case letters are considered to be different, thus the symbols: symbol1, Symbol1, and SYMBOL1 are three distinct symbols. Symbol-names beginning with SYS, such as SYSYEAR, SYSMONTH, etc., are reserved for system use and should not be used as symbol-designators. See Appendix C. The symbol for the current page number, &, remains the same even if the PAGE-NUMBER-SYMBOL control word is used.

An iterative substitution, as described in the SUBSTITUTE-SYMBOL control word, is automatically performed on all character string symbol-values. If the symbol-value is ommitted, the symbol's value is set to a null character string (length zero).

SUBSTITUTE-SYMBOL (.su)

With the SUBSTITUTE-SYMBOL control word you cause SCRIPT/370 to scan input lines for symbols defined with a previous SET-SYMBOL control word, and replace the symbol with it's current value. Successive scans are performed on each input line until no further set-symbol substitution can be found; then SCRIPT/370 proceeds to the next line.

The format of the SUBSTITUTE-SYMBOL control word is

```
.su n|on|off
```

n	is the number of succeeding lines to be scanned for set-symbols. If you omit a value for n, SCRIPT/370 chooses 1.
on	specifies that all succeeding lines are to be scanned until a subsequent .su off is encountered.

off specifies that scanning of input lines and substitution of values for symbols is to terminate.

Set-symbols may appear in any of the following forms:

```
&symbol
&symbol.
&symbol(*)
&symbol(*).
```

In each case, the symbol _must_ be immediately followed by a blank or a period. When the period (.) is used to terminate the set-symbol, it is removed when the substitution is performed. The period must be used if the symbol cannot be followed by a blank (e.g., "&symbol," must be written as "&symbol.,").

When the representation &symbol is used, the current value of symbol(0) is substituted in the line. When the representation &symbol(*) is used, the current value of all defined and non-null elements of the array are substituted except for symbol(0). When an array substitution is used, elements are ordered by subscript value (from lowest to highest) and separated by a comma and a blank.

Multiple scans are performed over the input line until no further set-symbol substitution can be found.

The substitution of set-symbols may increase or decrease the length of the text line. If the line's length reduces to zero, it is ignored. If the line's length expands so that it exceeds 130 characters, an error condition occurs if a single variable substitution caused the line overflow. If the overflow occurred as the result of an array substitution, the current line is terminated at the end of preceeding array element's comma. The next input line starts with the remaining array elements. As many lines, of up to 130 characters each, as necessary are generated until the array substitution is completed.

Substitution only occurs for symbols that are currently defined. The 2PASS option permits use of symbols defined physically later in the SCRIPT file. Thus, a Table of Contents can be created using set-symbols for the page numbers, as shown in "Using Set Symbols." Under rare circumstances, the substitution of symbols during pass2 which were not defined during pass1 may affect the length of certain pages and disrupt the page numbers assigned during pass1.

USING SET SYMBOLS

The page numbers in a table of contents or an index change each time a document is revised. The use of set-symbols can make such changes automatic. Suppose you are creating a table of contents. You should first define all the items which will be specified in the table of contents, and, then, associate each with a set-symbol name preceded by an & character. For example

```
Chapter 1 . . . . . . &chapter1
Chapter 2 . . . . . . &chapter2
Chapter 3 . . . . . . &chapter3
```

In order for SCRIPT/370 to substitute the assigned page numbers for the set-symbol names, you must specify substitute-mode. Put the following control words at the top and bottom of your table of contents, respectively:

```
.substitute-symbol on
    .
    .
    .
.substitute-symbol off
```

Now you must assign values to the set-symbols you have defined. The character &, when used alone, represents the current page number of the formatted output. For example, at the point in the SCRIPT file where chapter 2 begins, you must include a SET-SYMBOL control word which assigns to the name &chapter2 the current page number, as

```
.page-eject
.space-lines 2
CHAPTER 2. THE DOUBLE TASK OF LANGUAGE
.set-symbol chapter2=&
.space-lines 2
```

Note that when substitution is desired, you reference "&chapter2", but in the SET-SYMBOL control statement, you reference "chapter2". Note also that SCRIPT/370 would regard chapter2 and Chapter2 as different set-symbol names.

Figure 5 demonstrates another use of the SET-SYMBOL control word. The input files are LETTER SCRIPT and NAMES SCRIPT. When the command SCRIPT LETTER is issued, three pages will be printed. Each page will be a separate letter addressed to each of the three people designated in the NAMES file. The page shown is the second letter.

LETTER SCRIPT

```
.substitute-symbol on
.no-format;.center on
IBM VM/370 Development Group
19 May 1971
.center off;.sp 1
.imbed NAMES
.break
.sp 1
.format
     &name., the enclosed report contains a
further description of the SCRIPT manuscript
processing facility. I hope, &name., that
you will find it helpful.
.sp 1
.center on
Very truly yours,
.sp 3
John Smith
.center off;.substitute-symbol off
.pa
.append LETTER
```

NAMES SCRIPT

```
Tom Jones
947 Wood St.
Poughkeepsie, N.Y.
.set name='Tom'
.end-of-file
Paul Tardif
114 Maple St.
Montreal, Quebec
CANADA
.set name='Paul'
.end-of-file
Stuart Madnick
162 Winona St.
Peabody, Mass.
.se name='Stuart'
.end-of-file
.quit
```

```
                    VM/370 Development Group
                          19 May 1971

Paul Tardif
114 Maple Street
Montreal, Quebec
CANADA


     Paul,   the   enclosed   report   contains   a   further
description of the SCRIPT manuscript processing facility.  I
hope, Paul, that you will find it helpful.

                      Very truly yours,



                          John Smith
```

Figure 5. Sample SET-SYMBOL Control Word Usage

IMBED PARAMETER PASSING

Frequently, imbedded SCRIPT files have the same relationship to the master file as program subroutines have to the main program. It is possible to pass parameters to an imbedded file, processing them with the facilities provided by set symbols.

The full format of the IMBED control word is

```
.im file arg1 arg2 ... arg9
```

Up to 9 arguments can be specified in an IMBED request. These arguments will be assigned to the special set symbols &1 through &9. The set-symbol &0 will be set to the number of parameters specified only if arguments are specified. The set symbols &1 through &9 are not automatically stacked if an IMBED file is called within an imbedded file.

If the file HANGPARA SCRIPT is:

```
.substitute-symbol on
.in 0
.space-line
.offset &1
.substitute-symbol off
```

Then the following control word would cause subsequent text to be processed with an offset value of 5:

```
.im hangpara 5
```

Script files specified in imbed control words may be on any active disk associated with the userid. The standard CMS order of search is used to find the specified file.

THE SCRIPT COMMAND

The SCRIPT command is your means of causing the Script file containing your document to be processed by SCRIPT/370 and printed. The processing of your file is accomplished by SCRIPT/370 in accordance with the control words you entered into your document as you developed it. When you issue the SCRIPT command, you must name the file you wish to be processed, and indicate to SCRIPT/370 which of the output options are to be operative.

The format of the SCRIPT command is

```
     script filename [(option1 option2... optionN [)]]
```

filename specifies a file with a filetype of SCRIPT.

Options:

CENTER (CE)
: causes offline output to be shifted 30 characters to the right on the printer paper.

CENTERnn
: causes offline output to be shifted nn characters to the right on the printer paper. Two digits are required.

CONTINUE (CO)
: causes processing to continue after detecting and printing any errors, if possible.

DEBUG (DE)
: allows breakpoints to be set by the CMS DEBUG command; if this option is not used, breakpoints will cause a terminal error to occur.

FILE (FI)
: writes the edited and formatted output of SCRIPT into a file named "$filename", instead of at the terminal or offline printer.

MARK (MA)
: marks the beginning of each line of the original input by underlining the first character.

NOWAIT (NO)

starts SCRIPT output immediately without waiting for the first page to be adjusted.

NUMBER (NU)

prints in the left margin the SCRIPT filename and line number corresponding to each line of printed output. The text is shifted 30 characters to the right.

NUMBERnn

same as NUMBER, except that text output is shifted nn characters to the right. The SCRIPT filename and line number require 16 spaces reserved; the nn value is added to this automatic shift amount.

OFFLINE (OF)

prints the edited and formatted output of SCRIPT on the offline printer, instead of at the terminal.

PAGExxx

causes printout to start at page xxx.

QUIET (QU)

causes the SCRIPT version number identification line, normally printed immediately after issuing the SCRIPT command, to be suppressed.

SINGLE (SI)

terminates printing after one page, usually used in conjunction with the PAGExxx option.

STOP (ST)

causes a pause at the bottom of each page during SCRIPT printout.

TRANSLATE (TR)

translates lowercase letters to uppercase in printout.

UNFORMATTED (UN)

prints the inputted SCRIPT file along with the control words; the control words being ignored with no formatting of the output.

2PASS (2P)

causes 2 passes through the input files to occur; both passes process all the control words, but actual output only occurs on the second pass.

USING THE OPTIONS OF THE SCRIPT COMMAND

Filename must be specified with the SCRIPT command. The filetype SCRIPT is assumed. If "SCRIPT ?" is typed, a brief explanation of the SCRIPT command is typed including the list of valid control words.

When the SCRIPT command is issued, the specified SCRIPT file is typed either at the user's terminal, on the offline printer, or into a file. Execution is controlled by format control words included in the specified SCRIPT file. When the file is located, and typing is ready to begin, a response is typed, and execution pauses until a carriage return is entered at the terminal, unless the NOWAIT, OFFLINE, or FILE option has been specified. This pause allows the user to position the output paper at the top of a page. If STOP is specified with the command, the pause is repeated at the bottom of each page, allowing the user to change paper if noncontinuous forms are being used. If STOP is used, the paper should be positioned to the first line to be printed (the heading) rather than to the physical top of the page. Typing resumes when a carriage return is typed.

The TRANSLATE option is needed if output is to be directed to an offline printer that is not equipped with the uppercase and lowercase letters (TN-chain). In conjunction with the UNFORMATTED option, TRANSLATE provides a means of printing the original SCRIPT file on a printer that does not have the TN-chain (this can also be done by the CMS command PRINT filename SCRIPT (CC)).

The PAGExxx option, in conjunction with the SINGLE option, provides a means for selectively formatting and printing portions of a manuscript. The xxx represents a three-digit page number and must include leading zeros (for example, page 12 only should be requested by SINGLE PAGE012). Another means of selectively manipulating a formatted manuscript is to use the FILE option to generate the entire or relevant portion of a manuscript into a file and then use the CMS facilities of EDIT and/or TYPE to process it.

The CENTERnn and NUMBERnn options should be used with small explicit nn values if the text output is in multiple column format. The standard line length for double columns is 89 characters. The default nn value for the NUMBER option is NUMBER30 and would print to the left of each column, requiring

$$89 + 2 \times 30 = 149$$

characters - too many for a printer line.

The FILE option produces an output file in either typewriter format (backspace characters and carriage return characters are used) or printer format (printer control codes are used). The default format is typewriter. The printer format can be specified by the combination of both the FILE and OFFLINE options. A printer format file may be later printed by the CMS command PRINT with the CC option.

The QUIET option can be especially useful when the SCRIPT command is issued by means of a CMS EXEC file. This will cause the processing to begin without any interruptions or printout (the NOWAIT option may be also needed for online terminal printout to surpress other interruption). If multiple SCRIPT commands are issued from the EXEC file (each with appropriate QUIET and NOWAIT options), the output will correctly start on a new page for each input file as needed for "form letters", for example.

The 2PASS option can be used, perhaps in conjunction with the CONTINUE option, to scan the entire input for possible control word errors before starting any actual printout. Furthermore, when used in conjunction with the SET-SYMBOL and SUBSTITUTE-SYMBOL control words, the 2PASS option allows references to be automatically inserted which are not physically defined until later in the input file (e.g. "This will be discussed again on page &PAGENUM.", where &PAGENUM is a set-symbol defined later in the input file).

HALTING OUTPUT

Once you have entered a Script command and pressed RETURN, your keyboard is locked, whether or not you have specified the OFFLINE option. If you decide that you do not want your output to continue, perhaps because of errors in the Script file, or because you have specified an incorrect filename or have omitted a desired option, you can halt output processing as follows

1. Press ATTN. The system will respond by typing an exclamation point (!).
2. type ht. The system will respond with the READY message.

If, after you press ATTN, you decide to continue with your output, press RETURN. SCRIPT/370 will resume processing your file, but the line you interrupted <u>will</u> <u>not</u> be processed.

SCRIPT/370 ERROR PROCESSING

As SCRIPT/370 processes your file it responds to errors (usually improperly-specified control word) by printing appropriate messages at the terminal. If you have specified the CONTINUE option of the SCRIPT command, SCRIPT/370 does not stop processing your file, but goes on if possible. If you did not specify the CONTINUE option, processing stops when the error is encountered, output up to that point is printed, and the system responds with the CMS READY message followed by the CMS error code. The following information, as appropriate, is printed for each error detected:

1. The SCRIPT error number and description of the error situation.

2. The control word line or parameter that caused the error.

3. The number of input lines that had been processed up to the point that the error was encountered.

4. The specific SCRIPT filename and record number that was last read.

5. The SCRIPT filename and record number that imbedded the error file, if the error was encountered within an imbedded file.

Sample error output:

```
SCRIPT ERROR 02: CONTROL LINE PARAMETER SHOULD BE A NUMBER.
.sp abc
ERROR OCCURRED AFTER READING 00145 LINES.
LAST LINE READ WAS FROM FILE: ERROR    , LINE: 00003.
WHICH WAS IMBEDDED FROM FILE: TESTR    , LINE: 00067.
```

ERROR MESSAGES

The messages typed by SCRIPT/370 when it encounters an error are shown below.

E(00001) OUTPUT LINE TOO LONG OR PRINTER ERROR.
An output line longer than 132 characters was created. This usually is caused by neglecting to set the format mode

resulting in very long lines which, when printed using the CENTER option, exceed 132 characters.

E(00002) CONTROL LINE PARAMETER SHOULD BE A NUMBER.
An alphabetic parameter was found for a SCRIPT control word that requires a numeric parameter.

E(00003) MORE THAN EIGHT ACTIVE FILES - REDUCE NESTING.
SCRIPT files have been imbedded to a depth greater than eight (see .IM).

E(00004) INVALID CONTROL WORD ENCOUNTERED.
A line was read that started with a period but could not be recognized as a valid control word.

E(00005) CONTROL LINE PARAMETER MISSING.
A required parameter for this SCRIPT control word was ommitted.

E(00006) STATUS STACK OVERFLOW/UNDERFLOW.
An attempt was made to stack status to a depth greater than 5 (see .SA) or to restore status more times than it was saved (see .RE).

E(00007) NEGATIVE PAGE NUMBER COMPUTED.
A negative page number was computed. This is usually caused by using a negative parameter with the .PA control word incorrectly.

E(00008) INVALID CONTROL LINE PARAMETER.
A parameter specified is not valid for this SCRIPT control word (e.g. only ON or OFF are valid parameters for certain control words).

E(00009) LINE LENGTH 0 OR GREATER THAN 132.
The parameter to the .LL control word is not within the range 1 to 132.

E(00010) UNDENT>INDENT.
The execution of a .IN, .UN, or .OF control word would cause the left hand margin to move to the left of column 1.

E(00011) PREVIOUSLY SET OFFSET HAS NOT BEEN TRIGGERED.
Two .OF control words were encountered without any intervening text lines.

E(00012) HEADING MARGIN>TOP MARGIN.
The execution of a .HM or .TM control word would violate the constraint that the heading margin must be less than the top margin.

E(00013) FOOTING MARGIN>BOTTOM MARGIN.

The execution of a .FM or .BM control word would violate the constraint that the footing margin must be less than the bottom margin.

E(00014) FIRST PARAMETER SHOULD BE A SINGLE DIGIT.
The first parameter to a .RC or .CS control word must be a digit.

E(00015) .RC MODE WAS ON/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 already OFF.

E(00016) INVALID .RC TERMINATION - NUMBER NOT DEFINED OR ALREADY TERMINATED.
An attempt was made to undefine a revision code that was not currently defined.

E(00017) NEGATIVE SPACE COUNT GENERATED.
This is a system error and should not occur.

E(00018) FILE SYSTEM ERROR ON INPUT.
An error code was returned from the file system while reading input.

E(00019) TEMP FILE "CMSUT1 SCRIPT" ALREADY EXIST, ERASE IT.
The temporary file CMSUT1 SCRIPT is normally erased automatically by SCRIPT; if this file is a user file alter its name, otherwise erase it.

E(00020) CORRECT FORM IS: "SCRIPT" FILENAME (OPTIONS); TYPE "SCRIPT ?" FOR MORE INFORMATION.
A filename was not specified in the SCRIPT command.

E(00021) INPUT FILE NOT FOUND (SYSTEM ERROR).
The file specified in a .IM or .AP control word cannot be found.

E(00022) FILE SPECIFIED ON SCRIPT COMMAND NOT FOUND.
The file specified in the SCRIPT command cannot be found.

E(00023) MESSAGE CODE NOT USED.

E(00024) INVALID SCRIPT COMMAND OPTION.
One of the options to the SCRIPT command is not valid.

E(00025) RC STACK OVERFLOW.
Revision codes can only be nested to a depth of 9, there was an attempt to nest further. This is usually caused by forgetting to use appropriate .RC n OFF control words.

E(00026) SYSTEM ERROR HAS OCCURRED, PLEASE SAVE YOUR SCRIPT

FILE.
This message indicates a system error. The appropriate personnel should be informed of the circumstances. Usually this condition can be bypassed by diagnosing the cause of the error and changing the SCRIPT file accordingly.

E(00027) EQUAL SIGN (=) NOT FOUND IN .SET.
An equal sign is required in the operand field of the SET-SYMBOL control line.

E(00028) INVALID SYNTAX ON LEFT OF EQUAL SIGN OF .SET.
The symbol-designator of the SET-SYMBOL control line is not in one of the four legal forms.

E(00029) INVALID SYNTAX ON RIGHT OF EQUAL SIGN ON .SET.
The symbol-value of the SET-SYMBOL control line is not in one of the legal forms.

E(00030) .SET SYMBOL TABLE OVERFLOW.
The maximum number of set-symbols has been exceeded, this limit is normally set at 1000 symbols.

E(00031) UNDEFINED SYMBOL USED AS INDEX OF .SET SYMBOL ON LEFT OF EQUAL SIGN.
A symbol-designator of the form symbol1(&symbol2) was used in a SET-SYMBOL control line where symbol2 was not a previously defined set-symbol.

E(00032) INVALID (NON-DECIMAL) NUMBER USED AS INDEX OF .SET SYMBOL ON LEFT OF EQUAL SIGN.
A symbol-designator of the form symbol1(n) was used in a SET-SYMBOL control line where n was not a valid decimal number.

E(00033) INVALID (NON-DECIMAL) NUMBER ENCOUNTERED IN EXPRESSION ON RIGHT SIDE OF .SET.
The symbol-value of a SET-SYMBOL control line is an arithmetic expression which has a term which is neither a set symbol (e.g. &symbol) nor a valid decimal number.

E(00034) UNDEFINED SYMBOL ENCOUNTERED IN EXPRESSION ON RIGHT SIDE OF .SET.
The symbol-value of a SET-SYMBOL control line is an arithmetic expression which has a term in the form of a set symbol, e.g. &SYMBOL, where SYMBOL is not a previously defined set symbol.

E(00035) A TOKEN LONGER THAN 14 CHARACTERS ENCOUNTERED IN .SET.
A string of more than 14 characters has been encountered in a SET-SYMBOL control line where there are no break characters (e.g. blank, +, -, etc.) within the 42 character

string. This can not be a legal control line.

E(00036) MORE THAN 10 TOKEN ENCOUNTERED IN .SET.
A maximum of 10 tokens (symbols, punctuation, numbers) are allowed in a SET-SYMBOL control line (e.g. ".SET-SYMBOL ALPHA = BETA * 2 - GAMMA + 13" has exactly 10 tokens).

E(00037) INFINITE LOOP OCCURRED AS A RESULT OF RECURSIVE .SET SYMBOL SUBSTITUTION.
While processing an input line under the specification of a SUBSTITUTE-SYMBOL control word, each time a set-symbol was substituted its value contained another set-symbol and never terminated.

E(00038) SUBSTITUTION FOR .SET SYMBOL CAUSES LINE TO EXCEED MAXIMUM ALLOWABLE SIZE.
While processing an input line under the specification of a SUBSTITUTE-SYMBOL control word, the substitution of a set-symbol causes the input line to exceed 132 characters.

E(00039) UNABLE TO ALLOCATE SPACE FOR .SET SYMBOL TABLE.
The space of the set-symbol table is allocated by means of an SVC GETMAIN. There was not enough storage space available to satisfy the requirements of the GETMAIN.

E900040) FILE SPECIFIED ON IMBED OR APPEND NOT FOUND.
The file named in an IMBED or APPEND control word was not found. Check the filename and insure that it is correct and that the file is available under your userid.

E(00041) NUMERIC CONTROL LINE IS OUTSIDE OF VALID RANGE.
A parameter specified for a LINE-LENGTH or similar control word was erroneously specified as too large. Check the control word and respecify the parameters in error.

E(00042) INCORRECT NUMBER OF PARAMETERS SPECIFIED.
An incorrect number of parameters has been specified, such as specifying more than one value for a PAGE-LENGTH or INDENT control word. Check the control word and respecify it omitting the incorrect parameters.

E(00043) UNABLE TO ALLOCATE BUFFER SPACE FOR MULTIPLE COLUMN PROCESSING.
Less than 4096 bytes of virtual storage were available when required for buffer space. Press ATTN twice, and issue a DEFINE STORAGE command to increase the storage size of the virtual machine. Then issue IPL CMS (or the equivalent) and reissue the SCRIPT command.

E(00044) INSUFFICIENT BUFFER SPACE FOR MULTIPLE COLUMN PROCESSING.
Insufficient buffer space exists to format multiple column

output. Do one of the following: 1) Reduce the PAGE-LENGTH size and reissue the SCRIPT command; 2) Press ATTN twice and issue a DEFINE STORAGE command to increase the storage size of the virtual machine. Then issue IPL CMS (or the equivalent) and reissue the SCRIPT command.

EXAMPLE

This section contains unformatted and formatted copies of one of the sample problems that are distributed with the IUP tape to illustrate some of the facilities of the Script text processor. Since only one of the files is included here, references to an imbedded file in the unformatted copy (and the associated CONDITIONAL-SECTION control words) should be ignored.

```
.cs 1 ignore
.tt ////;.cm  this will suppress printing of Page xxx at the top of the following pages
.sp 6
.ce on;.cm this command will center each of the next lines typed
A Virtual Machine System for the 360/40
.sp 2
.tr ¬ 00;.cm see next line, blank character needed to adjust spacing
R.J. Adair¬
R.U. Bayles
L.W. Comeau
R.J. Creasy
.sp 2
IBM Cambridge Scientific Center Report
.sp 30
International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts
.sp 2
May, 1966
.ce off;.cm this resets default format-mode and left margins
.pa
.ri 4;.cm the next four lines of text will be
.cm moved to be in line with the right margin
May, 1966
.cm .ri, like .ce, is an implicit no-format command
Scientific Center Report
.sp 3
A VIRTUAL MACHINE
SYSTEM FOR THE 360/40
.sp 3
Abstract
.sp
.in 5;.cm all text will begin in column 6
.ll 55;.cm this, with the .in 5, will center the following text
.cm on the page, as all text will end in column 55
A virtual machine system, which provides copies of a 360
computing system for concurrent use by separate operating systems,
has been implemented for the IBM 360 Model 40.  The user
at a terminal interface of a virtual 360 has all of the capability,
with minor restrictions, provided by a stand-alone system.  The system
was designed as a system evaluation tool and as such, CPU
efficiency or throughput improvement was not a
primary design goal.
.in 0;.cm this resets the default for the left margin
.ll 60;.cm this command resets the default line length
.sp 2
-------------------------------
.br;.cm this command inhibits formatting, and will force the
.cm following text to a separate line
NOTE:   Cambridge worked on virtual machine concepts throughout
1965 and 1966 and in January, 1967, put the modified Model
40 into internal use supporting a dozen virtual machines.
     Parallel to this development, part of the Cambridge group
```

```
that worked on CP/40 began to work on a software solution for the
Model 67 user. In the fall of 1967, it completed CP/67, a
product oriented system.
.cm in the above paragraph, as in the following,
.cm initial blanks acted as a break
     In Janurary of 1968, Cambridge described CP/67 and CMS
to SHARE and in July of that year it became available to Model 67
users.  This system was the forerunner of VM/370.
.sp
   The information contained in this document is of historical
interest, and should not be confused with any current
VM/370 documentation.
.su on;.cm this command is necessary if the page numbers are to be
.cm supplied by the text processor when the 2pass option is used on the
.cm SCRIPT command line.
.nf
.pa
.se afigno=1;.cm these symbolic figure numbers, when used, may be convenient
.cm for files where figures may be added or deleted, or used conditionally
.cs 1 on;.cm this indicates that the following lines will be included or ignored
.cm depending on the setting of '.cs' at the top of the file
.se xfigno=1;.cm this introduces a new figure into the text
.se afigno=&xfigno+1;.cm this introduces a new value for 'afigno'.
.cs 1 off;.cm this is the end of the conditional section at this point
.se bfigno=&afigno+1;.cm now, whatever the value of 'afigno' from above,
.cm 'bfigno' will be one greater.
.cm the SCRIPT processor, when substitute mode is on, wil supply the
.cm correct numbers
.tb 6 ./55
.cm the above setting for the tabs indicate that tab stops are at 6 and 55, and that
.cm tabs to column 55 are to contain periods instead of blanks
.sp 2
.ce;.cm CE, with no on or off parameters, indicates that just one line is to be centered
TABLE OF CONTENTS
.sp 5
  I. INTRODUCTION   &tintro
.cm these set symbols refer to corresponding symbols set within the text, in the form '.se tintro=&'
.sp
 II. HARDWARE IMPLEMENTATION  &thard
.sp
III. PROBLEM MODE OPERATION OF THE ASSOCIATIVE MEMORY  &tproblem
.sp
 IV. CONTROL PROGRAM STRUCTURE     &tcontrol
.sp
  V. INPUT-OUTPUT OPERATIONS  &tinput
.sp
 VI. LIMITATIONS    &tlimit
.fi
.su off;.cm substitute mode is turned off when it is not needed
.cm to save on CPU time
.ps +;.cm defines page numbering parameter for following command:
.tt //-+-//
.pn on;.cm this command will initialize page numbering
```

```
.pa 1;.cm this assures that page numbering will begin with page 1
.cm on the first page of text
.tb 5;.cm this is the default, but when files are embedded, the last tab setting is in effect, so
.cm the defaults (if desired) can be reset by this command (.tb 5 assumes 10 15, 20, etc.)
.ce
I. INTRODUCTION
.se tintro=&;.cm this substitution symbol will, when the 2pass option is used
.cm         in processing, result in the proper page number appearing in the Table of Contents
.sp 2
     Late in 1964, the IBM Scientific Center (formerly
Systems Research and Development Center) at
Cambridge, Massachusetts, undertook a project with a number of
objectives.  Among these were:
.sp
.in 5
.tb 5 8
.of 3;.cm offsets and tabs used in conjunction to justify left margins
.cm of offset paragraphs
.cm tab setting = indentation level + offset value
-    the development of means for obtaining data on the operational
characteristics of both system applications programs;
.of 3;.cm begins a new offset sequence
-    the analysis of this data with a view toward more efficient
machine structures and programming techniques, particularly for use
in interactive systems;
.of 3
-    the provision of a multiple-console computer system for the
Center's computing requirements;
.of 3
-    the investigation of the use of associative memories in the
control of multi-user systems.
.in 0;.cm any command to indent will clear both previous indentations and offsets
.sp
.cs 1 on;.cm more of the conditional section is here
.di on;.cm the following command (until .di off is reached) will not be
.cm executed until text processing begins on a new page
.sa;.cm all current format settings (tab, indent, etc.) are to be saved
.im scdelay;.cm this is an imbedded file, which will begin a new page
.re;.cm this command restores the saved settings of control words
.di off
.cs 1 off
A system was designed which we thought would satisfy these goals and, in
addition, provide other useful features.  Efficiency in CPU utilization
was not a primary design consideration.
.sp
.tr $ b1;.cm this command will result in all '$'s' to appear as superscript '1's' in
.tr ? b2;.cm printed copy, for printers equipped with a TN train
.cm infrequently used characters are used to avoid substitution when not desired
     Central to the idea of this system is the concept of the
"virtual machine" and, in our case, the "virtual 360"$.  Because of our
desire to be able to measure a broad spectrum of programs, it is important
that the imposition of a measuring system results in minimum alteration
of the characteristics of the subject program.  The "virtual 360"
```

```
concept effects this minimum while providing the flexibility also
required for the multi-user environment.  In this system, the
subject program interacts with the multi-user controller in the same
manner as with the physical machine, and not by specially designed
supervisor calls or subroutine calls as in currently implemented
multi-programming packages.  The program does not "see" the software
interface between it and the physical hardware.
.sp
     Within these "virtual 360's" (called 360's), programs
such as operating systems, which were initially designed to run on
a hardware machine, may be run without change.  In order to use
the available facilities more efficiently, the Control Program
supporting these multiple 360's performs the traditional multi-user tasks,
such as scheduling, resource allocation, and core management.
.sp
     We have created, therefore, a multi-user system where each user's
virtual machine can run the programming system of his choice.  None
of these programming systems need consciously make use of multi-tasking
facilities to improve machine utilization.  Two other advantages accrue
from this design - the ability to dynamically alter the virtual
machine's configuration (core, size, available input-output units and
paths), and the ability to assign more than one virtual 360 to a
problem in order to examine the applications of multiprocessing.
.sp
     We are providing sixteen virtual machines which may address
256K bytes of main storage, a maximum of one multiplexor and
two selector channels, and a console typewriter.  Some of the virtual
machines may have additional typewriters, tape units, and a 2250
display console assigned to them.  A user is normally supplied
with three disks -- one read-only disk is to secure for all
users access to a library of often-used systems and routines, while
providing the protection necessary in a multi-user system.  The permanent
disk provides continuing storage capability to the user.  The temporary
disk is available to a virtual machine for the duration of the
session only.  The user retains complete control over the format and use
of his permanent and temporary space.  Programs may be loaded into the
user's virtual machine by name from the read-only disk, or by location
from any device attached to the virtual machine.
.sp 3
.cp 7;.cm this command will force the text to a new page if fewer than 7 spaces
.cm remain on the current page, will keep headings and text together
II.  HARDWARE IMPLEMENTATION
.se thard=&
.sp 2
     To provide these virtual machines, the Center
obtained a modified System 360/40? with
a multiplexor and two selector channels, interval timer, storage protection
feature, universal instruction set, and 256K bytes of main storage.
Its input-output equipment includes a console typewriter, line printer,
card reader and punch, 2702 Transmission Control with remote
terminals, four 2311 disk drives on two control units, two 240-III tape
drives, and a 2250 display unit with a 4K buffer.
.sp
```

```
.su on;.cm  this command will invoke set symbol substitution for the figure
.cm number in the next paragraph, as per set commands in the Table of Contents
     The CPU has been modified to permit dynamic relocation
of storage addresses by the addition of a 64
word (one per 4096 byte page of core memory) by 16 bit associative
memory (Figure &afigno.).
.cm with '.su on' during processing, the proper figure no. will be printed
A privileged operation to load and interrogate the memory has been
added to the instruction set. (see Figure &afigno..)
.cm since a period is necessary following a 'figno' to invoke substitution
.cm two periods are needed here if one is to print
.sp 3
.cp 7
III.  PROBLEM MODE OPERATION OF THE ASSOCIATIVE MEMORY
.se tproblem=&
.sp 2
     When the CPU is in problem mode, each main storage address
presented to memory is mapped by the following method (see Figure &bfigno.):
.sp
.in 5
.su off
the high order six bits of the eighteen bit memory address plus the
user identification number (set by the Control Program) are
presented to the associative memory for a match:
.sp
.tr ! af
.cm the translate coordinates 'af' supply a bullet for lists
.un 2
.un 2;.cm when an indent is in effect, an 'undent' commmand will set the
.cm first line of subsequent text two spaces to the left
.cm and by using a tab character, proper line-up of indented text is assured.
.cm this is an alternate method of offsetting portions of text
!    if a single match is found, the address of the selected
row of the memory replaces the high order six bits on the memory
bus, and the memory select takes place;
.sp
.un 2
!    if a multiple match condition (an error which should
never occur) or no match (requested page is not in memory)
occurs, an interruption is generated and the Control Program must take
the appropriate steps to resume execution.
.in 0
.tr;.cm this will negate all previously set translate control words
.sp
This mapping take place with no degradation of Model 40 cycle time.
.sp
     Six bits of the associative word are provided
to assist the scheduling section of the Control Program in selecting
the least costly (i.e. least likely to be brought back) page to roll
out when additional main memory space is required.
.sp
.in 5
The used bit is set when a match condition is found for a row
```

of memory, indicating a reference to the corresponding
page, and is reset when all of the pages represented
by the entries in the memory have been either referenced or
are locked;
.sp
the active bit is set at the same time the used bit is set,
but is not, like the used bit, automatically reset;
.sp
the changed bit is set when the instruction causing
the match condition could result in alteration of the contents of
the corresponding page;
.sp
the lock bit is set by the control program and is interpreted
by it to mean that the page may not be removed from memory;
.sp
the transit bit is used by the control program to indicate that the
page is currently being brought in or dumped out;
one spare bit is provided for unspecified use.
.sp 3
.in 0
.cp 7
IV. CONTROL PROGRAM STRUCTURE
.se tcontrol=&
.sp 2
When the Control Program code is being executed, the machine
is in the supervisor state; at all other times it is in the
problem state. Any action of a virtual machine which could cause
a change of machine state results in an interrupt. The Control
Program, then, is an interrupt driven system whose components reference
a set of tables describing the state of users' virtual machines. For
each user, this table (UTABLE) contains a copy of the current
Program Status Word (PSW) and the user's general purpose and floating-point
registers, the locations of the user's virtual memory pages
(which are either core or disk resident), a description of the input-output
equipment and its status, a copy of the user's interrupt region,
and other similar information.
.sp
There are two basic types of interrupts handled
by the Control Program: 1) those which invoke a section of the
Control Program to perform some function for the
virtual machine, and 2) those which require no special action by
the Control Program and are "reflected" to the virtual
machine (such as supervisor call and most program interrupts caused
by overflow conditions, protection violations, addressing errors, etc.).
The reflection of interrupts to the virtual machine is performed by the
appropriate swapping of PSW's in UTABLE and setting the proper interruption
codes there.
.sp
If a virtual machine's current PSW contains the wait bit
or if its execution has been delayed due to the temporary unavailability
of a necessary resource, it is considered
not runnable. At the occurrence of an interrupt which could affect
the runnability of a machine, UTABLE is examined

for interrupts pending which are enabled. If an enabled pending interrupt is found, the appropriate "reflection"
of the interrupt takes place (moving of the current PSW to one of the old PSW's, and of the corresponding new PSW to the current PSW).
.sp
A privileged operation interruption (caused by an attempted execution of a privileged operation while in problem mode) results in one of two actions, depending on whether the virtual machine was in problem or supervisor mode. If the virtual machine was in problem mode, the interrupt is merely reflected to the virtual machine. If the virtual machine was in supervisor mode, the action of the privileged operation, with the exception of input-output operations which are handled separately and discussed in a later section, must be simulated by the Control Program by appropriate changes in UTABLE.
.sp
A timer-initiated external interrupt controls CPU scheduling among the virtual machines. Each machine is allotted a quantum of time to run (which may be sliced into smaller intervals for timer simulation purposes) and, at the completion of the interval, a round-robin scan is made of the users to ascertain if another virtual machine is runnable.
.sp
An interrupt from the associative memory, caused by an attempted reference to a page not core resident, invokes the core management and scheduling routines. The missing page is indicated by the interruption code and the paging routines must schedule a page to be rolled out (according to algorithms which will be a chief point of study), and the appropriate page retrieved from disk. For the duration of the "page turning" the virtual machine is placed in not runnable condition.
.sp 3
.cp 7
V. INPUT-OUTPUT OPERATIONS
.se tinput=&
.sp 2
The input-output equipment generally falls into two classes: high data rate devices on the selector channel, and low data rate devices on the multiplexor channel. These characeristics, together with the need to share unit record facilities, the expected programming mode of the typewriter devices, and the differences in the logical structure of the channels, make it desirable to handle selector channel and multiplexor channel input-output operations separately.
.sp
Much of the selector channel input-output is not conveniently interruptable; therefore the channel programs are prescanned and all referenced pages are brought into core and held there for the duration of the operation. During this scan, virtual data addresses are converted to real core addresses, eliminating the need for translation hardware associated with channels. Similarly, direct access storage addresses (bin, cylinder, and head numbers) specified within the channel program are modified to provide partitioning of these devices, thus sharing the units among the virtual machines. The modified copies of the channel programs thus produced are used to directly control the selector

```
channel devices.  Under this scheme much of the validity checking and
interruption sequencing can be performed by the hardware.  Tables
are provided to map device addresses, detect path conflicts, and stack
interruptions for the virtual machines.  An input-output scheduler
provides request queuing and facilities scheduling at the hardware interface.
.sp
     The sub-channel programs for the shared unit record equipment
(punch, reader, and printer) on the multiplexor channel are run interpretively
All data for these devices are buffered in core and on disk, thus operating in a
spool-like mode.  All interrupts must be software simulated.
.sp
     Since multiplexor subchannels servicing typewriter
devices are expected to spend most of the time in a read state
awaiting input, buffers are provided to reduce the core tied up by these
operations from one or two pages to a few hundred bytes.  The I/O status
at these devices is controlled by the subchannel program; no information is read
before being requested by the virtual machine.  A mapping program is provided
to simulate the on-line typewriter with remote typewriters, when
desired.
.sp
     By depressing the BREAK button (a special feature of our
modified remote terminals which roughly corresponds to the Attention feature
of the online 1052 and 2741 remote terminals), the user may break out of his
virtual machine and enter conversation activity with the Console Function
routines, which provide the simulation of the following hardware console
functions:
.sp
.in 10
.nf
.tr ! af;.cm since translate symbols were cancelled, this one  must be reset
! Address Stop
! System Reset
! Start
! Stop
! Load
! External Interrupt
! Display
! Store
.in 0
.fi
.sp 3
.cp 7
VI.  LIMITATIONS
.se tlimit=&
.sp 2
     Taking the above concepts into consideration,
the following limitations were accepted in the initial implementation
of the system:
.sp
.in 5
.of 3
.cm since no other tab settings have been specified, the '.tb 5 8' is still in effect
-    the result of dynamic alteration of channel control
```

```
programs, while they are in execution, is generally unpredictable.
.of 3
-    correct operation of input-output timing dependent programs
may not be assumed;
.of 3
-    no input-output operation which requires more core than
is available for virtual machine page residence is allowed;
.of 3
-    the interval timer will accurately reflect only CPU execution time.
.in 0
.pa
.ce
```
<u>References</u>
```
.sp 2
.tr $ b1
.tr * b2
.of 5
$    "On Virtual Systems", D. Sayre, IBM Watson Research Center
.sp
.of 5
*    "A Time-Sharing System Using an Associative Memory",
A. B. Lindquist and R. R. Seeber, IBM Systems Development Laboratory.
Unpublished paper submitted for December, 1966, issue of "Proceedings
of the IEEE".
.in 0
.pa
.nf
.tr * ac;.cm translate symbols may be redefined without first cancelling them
.tr $ bc;.cm so that other translate symbols will stay in effect
.tr ¬ bb;.cm this series of symbols supply corners for box figures
.tr = ab
.tr - bf
.sp 5
                                                          MATCH INDICATORS
                                                                     |
                PAGE         USER     U    A    C    L  SP           |
               *------------------------------------------------$    |
CONTROL        |              |        |    |    |    |    |    |    |
               |--------------|--------|----|----|----|----|----|    |
               |              |        |    |    |    |    |    |    |
               =------------------------------------------------¬    |
                                                                     V
               *------------------------------------------------$   *--$
               |              |        |    |    |    |    |    |   |  |   0
               |--------------|--------|----|----|----|----|----|   |--|
               |              |        |    |    |    |    |    |   |  |   1
               |--------------|--------|----|----|----|----|----|   |--|
               |              |        |    |    |    |    |    |   |  |   2
         M     |--------------|--------|----|----|----|----|----|   |--|
         E     |              |        |    |    |    |    |    |   |  |   3
         M     |--------------|--------|----|----|----|----|----|   |--|
         O     |              |        |    |    |    |    |    |   |  |   4
         R     |--------------|--------|----|----|----|----|----|   |--|
        Y     :            :      :   :   :   :   :    :  :  :   !
              :            :      :   :   :   :   :    :  :  :   !
              :            :      :   :   :   :   :    :  :  :   !
              :            :      :   :   :   :   :    :  :  :   !
              *---------------------------------------$  *--$
              |            |      |   |   |   |   |    |  |  |   62
              |------------|------|---|---|---|---|----|  |--|
              |            |      |   |   |   |   |    |  |  |   63
              =---------------------------------------¬  =--¬
```

```
.sp 5
.su on
.ce
FIGURE &afigno..
.pa
```

```
                                                       | FROM CPU
                                                       |
                                                       V
   USER NUMBER                               *------------------------$
          |                                  |         |              |
          |                                  |         |              |
          =---------$                        =------------------------¬
                    |                        <---- ---> <----- ------->
          *---------+-----------------------¬              |
          |         |                                      |
   *------V---------V-------------$    M                   |
   |          |      |            |    A                   |
   |----------|------|------------|    T                   |
   | 11 11 11 | 11 11| 0 0 0 0    |    C                   |
   =------------------------------¬    H                   |
                                                           |
   *------------------------------$  *--$                  |
   |                              |  |  |  $               |
   |------------------------------|  |--|  |               |
 1 |                              |  |  |  |               |
   |------------------------------|  |--|  |               |
 2 |                              |  |  |  |               |
   |------------------------------|  |--|  |               |
 3 |                              |  |  |  > --$           |
   =------------------------------¬  =--¬  |   |           |
 ! :                              :  :  :  |   |           |
   :                              :  :  :  |   |           |
 ! :                              :  :  :  |   |           |
   :                              :  :  :  |   |           |
 ! :                              :  :  :  |   |           |
   *------------------------------$  *--$  |   |           |
63 |                              |  |  |  ¬   |           |
   =------------------------------¬  =--¬      |           |
                                               |           |
                                    *-----------------------------------$
                                    |          V  |        V            |
                                    |             |                     |
                                    =-----------------------------------¬
                                    <--------------V------------------->
```

```
                                   |
                               TO MEMORY
.sp 4
.ce
FIGURE &bfigno..
.fi
.su off
.tr
```

A Virtual Machine System for the 360/40

R.J. Adair
R.U. Bayles
L.W. Comeau
R.J. Creasy

IBM Cambridge Scientific Center Report

International Business Machines Corporation
Cambridge Scientific Center
Cambridge, Massachusetts

May, 1966

May, 1966
Scientific Center Report

A VIRTUAL MACHINE SYSTEM FOR THE 360/40

Abstract

A virtual machine system, which provides copies of a 360 computing system for concurrent use by separate operating systems, has been implemented for the IBM 360 Model 40. The user at a terminal interface of a virtual 360 has all of the capability, with minor restrictions, provided by a stand-alone system. The system was designed as a system evaluation tool and as such, CPU efficiency or throughput improvement was not a primary design goal,

----

NOTE: Cambridge worked on virtual machine concepts throughout 1965 and 1966 and in January, 1967, put the modified Model 40 into internal use supporting a dozen virtual machines.

Parallel to this development, part of the Cambridge group that worked on CP/40 began to work on a software solution for the Model 67 user. In the fall of 1967, it completed CP/67, a product oriented system.

In Janurary of 1968, Cambridge described CP/67 and CMS to SHARE and in July of that year it became available to Model 67 users. This system was the forerunner of VM/370.

The information contained in this document is of historical interest, and should not be confused with any current VM/370 documentation.

TABLE OF CONTENTS

I. INTRODUCTION

Late in 1964, the IBM Scientific Center (formerly Systems Research and Development Center) at Cambridge, Massachusetts, undertook a project with a number of objectives. Among these were:

- the development of means for obtaining data on the operational characteristics of both system applications programs;
- the analysis of this data with a view toward more efficient machine structures and programming techniques, particularly for use in interactive systems;
- the provision of a multiple-console computer system for the Center's computing requirements;
- the investigation of the use of associative memories in the control of multi-user systems.

A system was designed which we thought would satisfy these goals and, in addition, provide other useful features. Efficiency in CPU utilization was not a primary design consideration.

Central to the idea of this system is the concept of the "virtual machine" and, in our case, the "virtual 360"[1]. Because of our desire to be able to measure a broad spectrum of programs, it is important that the imposition of a measuring system results in minimum alteration of the characteristics of the subject program. The "virtual 360" concept effects this minimum while providing the flexibility also required for the multi-user environment. In this system, the subject program interacts with the multi-user controller in the same manner as with the physical machine, and not by specially designed supervisor calls or subroutine calls as in currently implemented multi-programming packages. The program does not "see" the software interface between it and the physical hardware.

Within these "virtual 360's" (called 360's), programs such as operating systems, which were initially designed to run on a hardware machine, may be run without change. In order to use the available facilities more efficiently, the Control Program supporting these multiple 360's performs the traditional multi-user tasks, such as scheduling, resource allocation, and core management.

We have created, therefore, a multi-user system where each user's virtual machine can run the programming system of his choice. None of these programming systems need consciously make use of multi-tasking facilities to improve machine utilization. Two other advantages accrue from this design - the ability to dynamically alter the virtual machine's configuration (core, size, available input-output units and paths), and the ability to assign more than one virtual 360 to a problem in order to examine the applications of multiprocessing.

We are providing sixteen virtual machines which may address 256K bytes of main storage, a maximum of one multiplexor and two selector channels, and a console typewriter. Some of the virtual machines may have additional typewriters, tape units, and a 2250 display console assigned to them. A user is normally supplied with three disks -- one read-only disk is to secure for all users access to a library of often-used systems and routines, while providing the protection necessary in a multi-user system. The permanent disk provides continuing storage capability to the user. The temporary disk is available to a virtual machine for the duration of the session only. The user retains complete control over the format and use of his permanent and temporary space. Programs may be loaded into the user's virtual machine by name from the read-only disk, or by location from any device attached to the virtual machine.

II. HARDWARE IMPLEMENTATION

To provide these virtual machines, the Center obtained a modified System 360/40[2] with a multiplexor and two selector channels, interval timer, storage protection feature, universal instruction set, and 256K bytes of main storage. Its input-output equipment includes a console typewriter, line printer, card reader and punch, 2702 Transmission Control with remote terminals, four 2311 disk drives on two control units, two 240-III tape drives, and a 2250 display unit with a 4K buffer.

The CPU has been modified to permit dynamic relocation of storage addresses by the addition of a 64 word (one per 4096 byte page of core memory) by 16 bit associative memory (Figure 1). A privileged operation to load and interrogate the memory has been added to the instruction set. (see Figure 1.)

III. PROBLEM MODE OPERATION OF THE ASSOCIATIVE MEMORY

When the CPU is in problem mode, each main storage address presented to memory is mapped by the following method (see Figure 2):

the high order six bits of the eighteen bit memory address plus the user identification number (set by the Control Program) are presented to the associative memory for a match:

- if a single match is found, the address of the selected row of the memory replaces the high order six bits on the memory bus, and the memory select takes place;
- if a multiple match condition (an error which should never occur) or no match (requested page is not in memory) occurs, an interruption is generated and the Control Program must take the appropriate steps to resume execution.

This mapping take place with no degradation of Model 40 cycle time.

Six bits of the associative word are provided to assist the scheduling section of the Control Program in selecting the least costly (i.e. least likely to be brought back) page to roll out when additional main memory space is required.

The used bit is set when a match condition is found for a row of memory, indicating a reference to the corresponding page, and is reset when all of the pages represented by the entries in the memory have been either referenced or are locked;

the _active_ bit is set at the same time the used bit is set, but is not, like the used bit, automatically reset;

the _changed_ bit is set when the instruction causing the match condition could result in alteration of the contents of the corresponding page;

the _lock_ bit is set by the control program and is interpreted by it to mean that the page may not be removed from memory;

the _transit_ bit is used by the control program to indicate that the page is currently being brought in or dumped out; one spare bit is provided for unspecified use.

IV. CONTROL PROGRAM STRUCTURE

When the Control Program code is being executed, the machine is in the supervisor state; at all other times it is in the problem state. Any action of a virtual machine which could cause a change of machine state results in an interrupt. The Control Program, then, is an interrupt driven system whose components reference a set of tables describing the state of users' virtual machines. For each user, this table (UTABLE) contains a copy of the current Program Status Word (PSW) and the user's general purpose and floating-point registers, the locations of the user's virtual memory pages (which are either core or disk resident), a description of the input-output equipment and its status, a copy of the user's interrupt region, and other similar information.

There are two basic types of interrupts handled by the Control Program: 1) those which invoke a section of the Control Program to perform some function for the virtual machine, and 2) those which require no special action by the Control Program and are "reflected" to the virtual machine (such as supervisor call and most program interrupts caused by overflow conditions, protection violations, addressing errors, etc.). The reflection of interrupts to the virtual machine is performed by the appropriate swapping of PSW's in UTABLE and setting the proper interruption codes there.

If a virtual machine's current PSW contains the wait bit or if its execution has been delayed due to the temporary unavailability of a necessary resource, it is considered not runnable. At the occurrence of an interrupt which could affect the runnability of a machine, UTABLE is examined for interrupts pending which are enabled. If an enabled pending interrupt is found, the appropriate "reflection" of the interrupt takes place (moving of the current PSW to one of the old PSW's, and of the corresponding new PSW to the current PSW).

A privileged operation interruption (caused by an attempted execution of a privileged operation while in problem mode) results in one of two actions, depending on whether the virtual machine was in problem or supervisor mode. If the virtual machine was in problem mode, the interrupt is merely reflected to the virtual machine. If the virtual machine was in supervisor mode, the action of the privileged operation, with the exception of input-output operations which are handled separately and discussed in a later section, must be simulated by the Control Program by appropriate changes in UTABLE.

A timer-initiated external interrupt controls CPU scheduling among the virtual machines. Each machine is allotted a quantum of time to run (which may be sliced into smaller intervals for timer simulation purposes) and, at the completion of the interval, a round-robin scan is made of the users to ascertain if another virtual machine is runnable.

An interrupt from the associative memory, caused by an attempted reference to a page not core resident, invokes the core management and scheduling routines. The missing page is indicated by the interruption code and the paging routines must schedule a page to be rolled out (according to algorithms which will be a chief point of study), and the appropriate page retrieved from disk. For the duration of the "page turning" the virtual machine is placed in not runnable condition.

V. INPUT-OUTPUT OPERATIONS

The input-output equipment generally falls into two classes: high data rate devices on the selector channel, and low data rate devices on the multiplexor channel. These characeristics, together with the need to share unit record facilities, the expected programming mode of the typewriter devices, and the differences in the logical structure of the channels, make it desirable to handle selector channel and multiplexor channel input-output operations separately.

Much of the selector channel input-output is not conveniently interruptable; therefore the channel programs are prescanned and all referenced pages are brought into core and held there for the duration of the operation. During this scan, virtual data addresses are converted to real core addresses, eliminating the need for translation hardware associated with channels. Similarly, direct access storage addresses (bin, cylinder, and head numbers) specified within the channel program are modified to provide partitioning of these devices, thus sharing the units among the virtual machines. The modified copies of the channel programs thus produced are used to directly control the selector channel devices. Under this scheme much of the validity checking and interruption sequencing can be performed by the hardware. Tables are provided to map device addresses, detect path conflicts, and stack interruptions for the virtual machines. An input-output scheduler provides request queuing and facilities scheduling at the hardware interface.

The sub-channel programs for the shared unit record equipment (punch, reader, and printer) on the multiplexor channel are run interpretively All data for these devices are buffered in core and on disk, thus operating in a spool-like mode. All interrupts must be software simulated.

Since multiplexor subchannels servicing typewriter devices are expected to spend most of the time in a read state awaiting input, buffers are provided to reduce the core tied up by these operations from one or two pages to a few hundred bytes. The I/O status at these devices is controlled by the subchannel program; no information is read before being requested by the virtual machine. A mapping program is provided to simulate the on-line typewriter with remote typewriters, when desired.

By depressing the BREAK button (a special feature of our modified remote terminals which roughly corresponds to the Attention feature of the online 1052 and 2741 remote terminals), the user may break out of his virtual machine and enter conversation activity with the Console Function routines, which provide the simulation of the following hardware console functions:

- Address Stop
- System Reset
- Start
- Stop
- Load
- External Interrupt
- Display
- Store

VI. LIMITATIONS

Taking the above concepts into consideration, the following limitations were accepted in the initial implementation of the system:

- the result of dynamic alteration of channel control programs, while they are in execution, is generally unpredictable.
- correct operation of input-output timing dependent programs may not be assumed;
- no input-output operation which requires more core than is available for virtual machine page residence is allowed;
- the interval timer will accurately reflect only CPU execution time.

References

1 "On Virtual Systems", D. Sayre, IBM Watson Research Center

2 "A Time-Sharing System Using an Associative Memory", A. B. Lindquist and R. R. Seeber, IBM Systems Development Laboratory. Unpublished paper submitted for December, 1966, issue of "Proceedings of the IEEE".

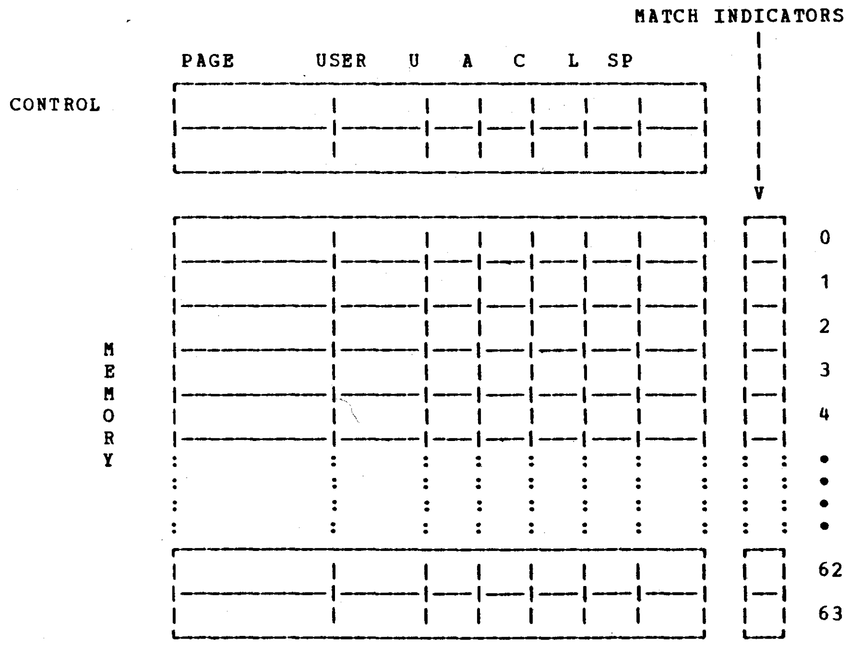

MATCH INDICATORS
PAGE
USER
U
A
C
L
SP
CONTROL
MEMORY
0
1
2
3
4
62
63

FIGURE 1.

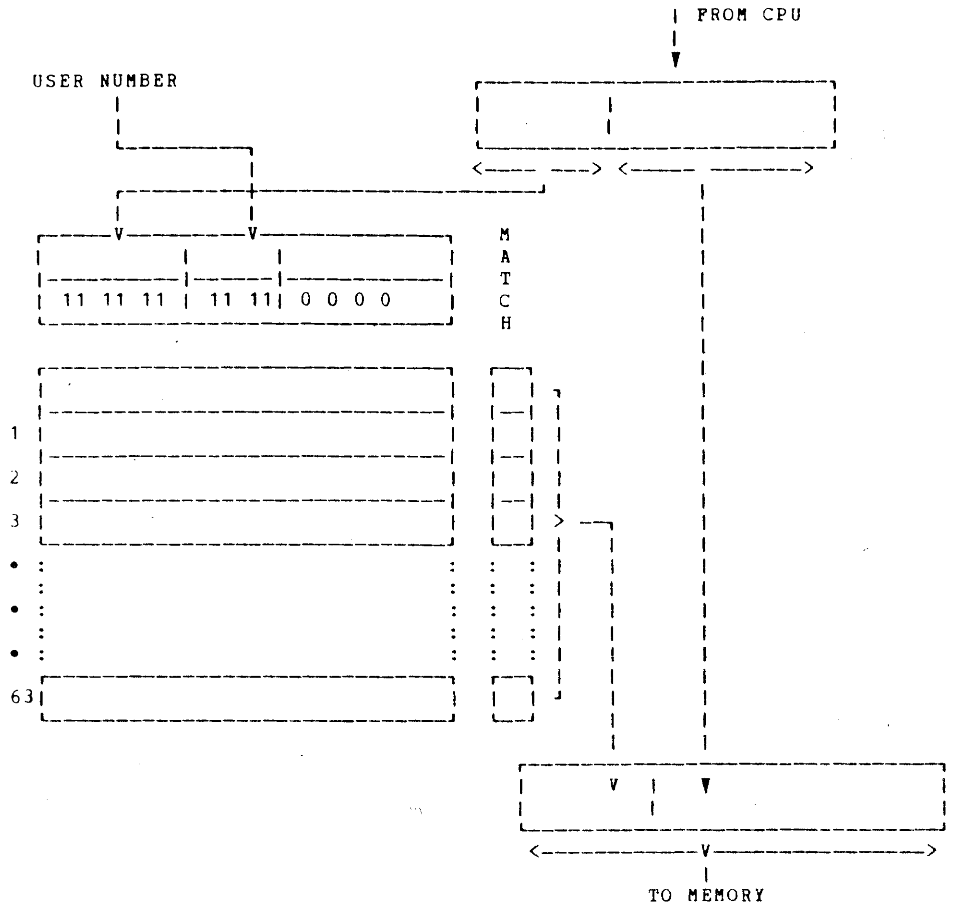


FIGURE 2.

End of EXAMPLE
This page intentionally left blank.

USING SCRIPT/370

All creation, modification, and manipulation of your Script files must be done using commands available to you as a user of CMS, the Conversational Monitor System. This section describes some commands which you must learn in order to work with Script files, and lists others which you may find helpful. Descriptions of each command available through CMS are contained in the following publications:

- *IBM Virtual Machine Facility/370: Terminal User's Guide*, Order No. GC20-1810
- *IBM Virtual Machine Facility/370: Command Language User's Guide*, Order No. GC20-1804
- *IBM Virtual Machine Facility/370: EDIT Guide*, Order No. GC20-1805

USING THE CMS EDITOR

Undoubtedly you will want to make modifications to your Script files. When the Editor is in Edit mode, you can insert, replace, or delete entire lines, and also change parts of a line. Edit mode is entered in either of two ways:

1. For an existing file, by using the command,

   ```
   edit 'filename' script
   ```

2. After you have initially developed part of a Script file in Input mode, by entering a null line.

Operation in Edit mode is by means of a pointer, which can be moved up and down through the file and can be interrogated about its position in the file at any time. In order to make corrections, you must know where the pointer is located.

If you have entered Edit by method 1 above, the pointer is at a blank line preceding the first line of your file; by method 2, it is at the last line of input.

POSITIONING THE POINTER

The TOP Subcommand (top): This command positions the pointer to the blank line preceding the first line of the file. It enables insertions to be made at the top of the file.

The NEXT Subcommand (next 'n' or n 'n'): If no number is specified, 1 is assumed. Pointer moves down the specified number of lines in the file. If the end of the file (EOF) is reached, NEXT positions the pointer at the last line of the file. This command functions as a BOTTOM subcommand (see below) if 'n' is greater than the number of lines between the pointer and the bottom of the file.

The UP Subcommand (up 'n' or u 'n'): If no number is specified, 1 is assumed. Moves the pointer up in the file the specified number of lines. This functions as a TOP subcommand if n is greater than the number of lines between pointer and top of file. The line at which the pointer is positioned is printed out.

The BOTTOM Subcommand (bottom or bo): This command positions the pointer at the last line of the file.

The TYPE Subcommand (type or t): If at any time you are uncertain about the position of the pointer, issue this command. The system responds by typing the line at which the pointer is directed.

The pointer can be moved either according to its line position (as described above), or by line context (searching for a line by its content). Ordinarily it is easier to do context editing, since the relative position of a file entry is not usually known.

The LOCATE Subcommand (locate /'string'/ or l /'string'/): This command searches for the first appearance of the specified string of characters. The search begins with the line after which the pointer is positioned, and the line in which the string occurs is typed.

The string is enclosed by any pair of characters (called <u>delimiters</u>) not occurring in it. Generally a slash is used as a delimiter except when the string includes a slash. The rule is that any character can be used as a string delimiter as long as it does not appear in the string, and as long as the same character is used to enclose both sides of the string. For example, the following are valid uses of string delimiters:

```
     locate 8string8

     locate .string.
```

The character string must be unique to the statement being searched, since the first occurrence of the string will be located. However, if it is not unique and is found first in the wrong line, the command can be issued again. Remember to include in the string any blanks or other unique characters that appear in the statement being searched: that is, the string to be searched must be typed exactly as it appears in the file. Delimiters are also used with the Change command (see below).

If you receive an end-of-file message (EOF:) rather than the text line you are searching for, try reissuing the LOCATE with fewer characters in the string. The reason for this is that the Editor can recognize only a character string that appears in the file entirely within one line as stored in the file. Therefore, if any part of your Locate string carries over to the next line, Edit will search through your entire file without finding it, and the pointer will be positioned at the end of your file.

<u>The FIND Subcommand</u> (find 'line' or f 'line'): The FIND subcommand, unlike Locate, is position dependent, and no delimiters are used. To find the line which, beginning in position 2, has the words "Section one", type

```
     f  Section one
```

The first blank after the command is part of the command format. The second blank indicates a blank in position 1 of the text line, and the search for nonblank characters begins in column 2. As in Locate, the line in which the characters occur is typed.

To use FIND for a tabbed line, you must press the tab key before typing the characters you are searching for. If tabs occur within the characters you are searching for, you must use the tab key to create either a LOCATE string or a FIND string.

CHANGING PART OF A LINE

The CHANGE Subcommand (c /'string1'/'string2'/): Change searches left to right in the current line for the first occurrence of string1 and replaces it with string2. The current line is expanded or compressed as required (string1 and string2 can be of different lengths), and the line is typed in its changed form. If no match is found, the line remains unchanged, the message FIELD NOT FOUND is typed, and the pointer (as above) remains positioned at the current line so that the line can be operated on again easily. As with the LOCATE subcommand, delimiters for the strings must not occur in either of the strings.

To delete, that is, not substitute new characters for string1, issue a null string2, as follows:

```
     c /'string1'//
```

The CHANGE subcommand can also be issued just once to make a correction in any number of lines--either on just the first, or on every occurrence of the string in those lines. This is done by adding parameters to CHANGE. For example:

```
     c /mony/money/ 10
```

In the next ten lines (starting with the one pointed to) "mony" will be searched for and the first occurrence of "mony" in each line will be changed to "money". If you want every occurrence of "mony" in those lines to be changed to "money" you need a second parameter, an asterisk:

```
     c /mony/money/ 10 *
```

The * denotes "in every case".

To make the specified change in every line in the file, use an asterisk as the first parameter instead of a number:

```
     c /mony/money/ * *
```

EXAMPLE OF USING LOCATE AND CHANGE COMMANDS

Suppose the second line of the file below is to be searched for and changed (note that the text begins in column 4).

```
   The quick brown fox jumps
   over the quiet stream.
```

If, by mistake, you issue:

```
     l /qui/
```

EDIT locates and types:

```
   The quick brown fox jumps
```

since that is the line in which the string "qui" first occurs.

Since you know that the desired line is the next line in the file, you issue:

```
     n
```

If you did not know this, you would reissue:

```
     l /qui/
```

In either case, the system types:

```
   over the quiet stream.
```

The pointer is now positioned at the desired line and you can use the CHANGE command to make your correction.

If you want the line to read:

```
   over the quiet street.
```

Issue:

```
     c /stream/street/
```

The system types the altered line:

```
   over the quiet street.
```

and the pointer remains positioned at that line.

Suppose you now want to change "street" to "streets". If you issue:

```
     c /et/ets/
```

EDIT locates the first occurrence of "et" (in "quiet") and prints the altered line:

```
   over the quiets street.
```

To correct this error, you can issue something like this:

```
     c /ts/t/
```

The system will respond:

```
   over the quiet street.
```

You have corrected your error. This time change "street" to "streets" by including more letters in the string than you did before. (The character string specified in the CHANGE string should be unique to the line.) Suppose you issue:

```
     c /eet/eets/
```

The first occurrence of "eet" is in "street". The desired altered line would then be typed out:

```
   over the quiet streets.
```

If you want to delete the word "quiet", you could use the CHANGE subcommand with no string2 specification:

```
     c /quiet//
```

The line would then read:

```
   over the  street.
```

If you decide to reinsert "quiet", issue:

```
     c /the/the quiet/
```

Using the FIND Subcommand: If you had used the FIND subcommand (instead of LOCATE) to search for the first line in the example above, you could have typed:

```
     f    The quick
```

The first blank is required by format rules; the next blanks indicate the occurrence of blanks in positions 1, 2, etc. of the line being searched for. These blanks must be specified if they exist in the file if you are using FIND.

CHANGING AN ENTIRE LINE

The RETYPE Subcommand (r 'line'): The current line is replaced with 'line'. If no 'line' is specified (that is, if only r is typed) the current line is deleted, and the Input mode is entered. This is useful when it is necessary to replace the current line with more than one line, or when many additions must be made to a file that has already been created (it is easier to use Input mode for this purpose). You must type a null line to reenter Edit mode, and issue the FILE subcommand to save your insertions.

REPLACE corresponds in format rules to FIND: 'line' is separated from the request by only one blank, and any other blanks are considered part of 'line'; no delimiters are used; and the request is position-dependent.

Example: We shall use the same two lines as in the example above:

```
The quick brown fox jumps
over the quiet stream.
```

To use REPLACE to change "stream" to "streets," position the pointer to the desired line, and issue:

```
r    over the quiet streets.
```

Note that the entire new line must be typed, since the old line is going to be overlaid by the new one. Also, the same number of blanks separate r from 'line' as were used in the FIND subcommand above. In general, CHANGE is easier to use for small changes within a line.

The DELETE Subcommand (del 'n'): Starting with the current line, the specified number of lines are deleted. If no number is specified, only the current line is deleted. The pointer is positioned at the line following the last deleted line.

Before making additional changes, it is helpful to issue t (for type) to ascertain the current line position following a DELETE subcommand.

ADDING A LINE IN EDIT MODE

The INSERT Subcommand (i 'line'): This subcommand allows a line to be added to a file without transferring to the Input mode. The line is inserted after the line at which the pointer is positioned, and the pointer is advanced to the inserted line. Thus, additional lines can be entered between existing lines. A blank line can be inserted by using one or more spaces for 'line' but, if 'line' is omitted and no spaces are inserted, the Input mode is entered. The same format rules as for FIND and REPLACE are observed for the INSERT subcommand.

LISTING PART OF A FILE

In the process of editing, it is easy to lose track of the changes that have been made. The TYPE subcommand lists one line, several lines, or the entire file within Edit mode. The command script 'filename' ('options') produces an entire listing, but in the CMS command environment rather than in Edit mode.

To type only one line, issue t 1, or simply t.

To type part of a file, use NEXT, FIND, or LOCATE to position the pointer at the first line to be printed, and issue t 'n' where 'n' is the number of lines to be printed.

In order to type the entire file on the terminal, issue top, then t 'n' where 'n' is greater than the number of lines in the file.

EXITING FROM EDIT MODE

There are two ways to exit from the Edit environment into CMS:

1. To save your edited text and enter CMS command environment, issue the "file" subcommand in the Edit mode.

2. If you decide not to save the changes made in Edit, and to enter CMS command environment, issue the "quit" subcommand.

USING OTHER VM/370-CMS COMMANDS

The commands listed in Figure 6 may be of use to you. They are a subset of the VM/370 commands which are often used by people working with SCRIPT/370. All of the commands shown can be issued in the CMS environment, although some are properly commands of the Control Program component (CP) of VM/370.

You should become familiar with the functions available to you as a SCRIPT user. If any function described in Figure 6 satisfies a requirement that you have, refer to *IBM VM/370: Command Language User's Guide* for instructions on how to use the command.

You should know that, as a user of SCRIPT/370, you have access to a virtual machine which is created for you by VM/370. It is the virtual machine environment which enables you to use SCRIPT/370 while other users at your installation pursue their own operating system and problem-solving requirements. You may wish to read *IBM VM/370: Introduction* to gain an appreciation of the VM/370 capabilities.

ACCESS	define additional direct access space to a CMS virtual machine.
COPYFILE	copy files according to specifications.
DEFINE	reconfigure a virtual machine.
DETACH	disconnect a real device from a virtual machine.
ERASE	delete files from user disks.
EXEC	process special procedures made up of frequently used sequences of commands.
IPL	initialize a virtual machine.
LINK	provide access to a specific disk.
LISTFILE	list information about CMS files.
LOGIN	provide access to VM/370.
LOGOUT	disable access to VM/370.
MSG	transmit messages from user to user.
PRINT	spool a specified file to the printer.
PUNCH	spool a specified file to the punch.
QUERY	request information about the virtual machine and system status.
SET	control various functions within the virtual system.
SORT	arrange a specified file in ascending order.
SPOOL	alter spooling control options.
TAPE	create a CMS file from data on tape, and vice versa.
TRANSFER	direct spooled printer or punch files to a specified user's virtual card reader.
TYPE	type all or part of a CMS file at the terminal.

Figure 6. VM/370-CMS Command Subset Summary

INSTALLING SCRIPT/370

This part of the manual describes the system requirements and procedures necessary to install SCRIPT/370. When properly installed, as verified by successful execution of the sample problem supplied with the SCRIPT/370 distribution tape, the SCRIPT/370 text processing facility is activated by entering the SCRIPT command under CMS.

SCRIPT/370 DISTRIBUTION TAPE

The SCRIPT/370 distribution tape is a nine-track tape recorded at either 800 or 1600 bpi, as requested by the recipient, containing three files.

1. A file in CMS tape dump format, which contains the SCRIPT/370 processor. The SCRIPT/370 processor accepts input files of filetype SCRIPT and produces formatted printed output from them.

2. A file in CMS tape dump format containg source, macros, and associated object modules for maintenance of SCRIPT/370.

3. A file in CMS tape dump format containing the sample problem and its associated EXEC file, required to execute the sample problem.

SYSTEM REQUIREMENTS

In addition to a correctly operating VM/370, SCRIPT/370 requires the following for installation.

1. A virtual machine with main storage of at least 256K. The CMS nucleus occupies main storage from location zero to location 20000(hex). SCRIPT/370 occupies main storage from location 20000(hex) to approximately location 30000(hex).

2. Approximately 50 blocks (800 bytes/block) on the system disk to contain the SCRIPT module. (A 2314 cylinder contains 150 such blocks, and a 3330 cylinder 266.)

3. Approximately four 3330 cylinders or eight 2314 cylinders on a disk other than the system disk, to contain the source files, object module files, sample problem, and sample problem EXEC file.

   Note: Space for the sample problem and its EXEC file need not be permanently allocated.

4. To maintain or modify SCRIPT/370, a temporary disk of ten 3330 cylinders is recommended. This can be allocated by issuing the command

   ```
   DEFINE T3330 19x CYL 10
   ```

5. A line printer equipped with the TN chain to provide upper- and lower-case alphabetic characters. If a TN chain is not available, the TRANSLATE option of the SCRIPT command must be specified. SCRIPT/370 output should be directed to the appropriate output class by means of the SPOOL command.

INSTALLATION PROCEDURE

To install SCRIPT/370, perform the following procedure.

Loading the SCRIPT/370 Distribution Tape

1. Mount the SCRIPT/370 distribution tape on a tape unit and ATTACH that unit to a virtual machine as device 181. The system disk should be ACCESSed as the A-disk, in read/write status.

2. Issue the CMS command TAPE LOAD which will load the file named SCRIPT MODULE.

Loading the Source and Object Modules

1. Keep the SCRIPT/370 distribution tape mounted, and its tape unit ATTACHed to the virtual machine as device 181.

2. ACCESS the disk that is to contain the source and object files as the A-disk.

3. To insure that the tape is positioned properly, issue the following CMS commands

```
TAPE REW

TAPE FSF 1

TAPE LOAD * *
```

This will load the following files

```
SCSFOR ASSEMBLE
SCSLIN ASSEMBLE
SCSLNK ASSEMBLE
SCSPRT ASSEMBLE

SCSFOR TEXT
SCSLIN TEXT
SCSLNK TEXT
SCSPRT TEXT
```

Loading the Sample Problem

1. Keep the SCRIPT/370 distribution tape mounted, and its tape unit ATTACHed to the virtual machine as device 181

2. ACCESS the disk that is to contain the sample problem and its EXEC file as the A-disk.

3. To insure that the tape is positioned properly, issue the following CMS commands

```
TAPE REW

TAPE FSF 2

TAPE LOAD * *
```

This will load the following files

```
SAMPLE EXEC

PROBLEM SCRIPT
PROBLEM2 SCRIPT
SCDELAY SCRIPT
```

Executing the Sample Problem

The sample problem EXEC file makes available a command called SAMPLE. Issuing the SAMPLE command causes the sample problem to be executed. Successful execution of the sample problem verifies correct installation and operation of the SCRIPT/370 processor contained in the SCRIPT module. To execute the sample problem, perform the following procedure.

1. Mount a TN chain on the printer that is to receive the output of the SCRIPT/370 processor. If no TN chain is available, the TRANSLATE option of the SCRIPT command must be specified.

2. Have the system operator START the printer as output class S.

3. ACCESS the disk that contains the sample problem EXEC file as an extension of the A-disk for the userid under which the test is being performed.

4. Issue the following CMS command

   ```
   SAMPLE
   ```

   The following files will be processed by SCRIPT/370 and directed to the line printer previously specified.

   ```
   PROBLEM Formatted
   PROBLEM2 Formatted

   PROBLEM Unformatted
   PROBLEM2 Unformatted
   SCDELAY Unformatted
   ```

SYSTEM MAINTENANCE

SCRIPT/370 has been written in the VM/370 Assembler Language and uses CMS macros. Maintenance of the system will be by CMS update files and requires that the SCRIPT/370 source code be disk resident when program maintenace is being performed. Update instructions will be supplied with the maintenance releases.

APPENDIX A. Compatibility with SCRIPT (CP-67/CMS)

SCRIPT/370 is fully compatible with files created for processing by SCRIPT (CP-67/CMS). Although the FILL-MODE, HEADING, and FOOTING control word of this earlier version have been superseded by the FORMAT-MODE and various titling control words in SCRIPT/370, Script files created under the CP-67/CMS version will be processed correctly by SCRIPT/370.

APPENDIX B. Control Word Summary

The table which follows summarizes SCRIPT/370 control words and their characteristics. It can be used as a convenient reference sheet at the terminal.

Control Word	Function	Page Reference	Implicit Break	Standard Setting or Default Value
.ap (APPEND)	Allows an additional file to be appended to the one just printed or typed.	49		
.bc (BALANCE-COLUMNS)	Causes subsequent text to be placed starting at the next column or page.	66		Balanced Column Mode
.bm (BOTTOM-MARGIN)	Specifies the number of lines in the bottom margin.	21	y	6
.br (BREAK)	Prevents the concatenation of the following text lines with preceding text lines.	32	y	
.bt (BOTTOM-TITLE)	Specifies a title line for the bottom of the current and each subsequent page.	25		
.cb (COLUMN-BEGIN)	Causes subsequent text to be placed starting at the next column or page.	65	y	
.cc (CONDITIONAL-COLUMN-BEGIN)	Causes a column eject if fewer than n lines remain in the column.	65	y	
.cd (COLUMN-DEFINITION)	Specifies the number of columns on a page and the leftmost position of each.	64	y	
.ce (CENTER)	Specifies the centering of the following text line(s).	36	y	n=1

Control Word	Function	Page Reference	Implicit Break	Standard Setting or Default Value
.cl (COLUMN-LENGTH)	Specifies the number of characters in each line of a column.	64		Line Length
.cm (COMMENT)	Allows comments to be stored in the file for future reference.	60		
.co (CONCATENATE-MODE)	Causes output lines to be formed by concatenating input lines.	32	y	Concatenate-Justify Mode
.cp (CONDITIONAL-PAGE-EJECT)	Causes a page eject if fewer than n lines remain on the page.	40		
.cs (CONDITIONAL-SECTION)	Allows conditional inclusion of input in the formatted output.	55		INCLUDE
.cw (CONTROL-WORD-SEPARATOR)	Specifies the character used for separation of control words on a single input line.	59	y	;
.di (DELAY-IMBED)	Delays the inclusion of a portion of the input file until the next page eject occurs.	47	y	
.ds (DOUBLE-SPACE-MODE)	Specifies that subsequent formatted output will be double spaced.	38	y	Single Space Mode
.eb (EVEN-PAGE-BOTTOM-TITLE)	Specifies that a title line for the bottom of the current page, if it is even-numbered, and all subsequent even-numbered pages.	25		
.ef (END-OF-FILE)	Simulates an end of file condition.	49		

Control Word	Function	Page Reference	Impli-cit Break	Standard Set-ting or Default Value
.ep (EVEN-PAGE-EJECT)	Causes one or two page ejects such that the next page will be even numbered.	40	y	
.et (EVEN-PAGE-TOP-TITLE)	Specifies a title line for the top of each subsequent even-numbered page.	24		
.fo (FORMAT-MODE)	Causes concatenation of input lines, and left and right justification of output. (Also called Concatenate-Justify).	31	y	Concatenate-Justify Mode
.fm (FOOTING-MARGIN)	Specifies the number of blank lines between the last line of text and the bottom title.	22	y	2
.hm (HEADING-MARGIN)	Specifies the number of blank lines between the top title and the first line of text.	22	y	2
.im (IMBED)	Inserts a file of text and/or control words into the one being processed by the SCRIPT command.	46 79		
.in (INDENT)	Specifies the number of spaces subsequent text is to be indented when printed or typed.	41	y	Left margin. n=1
.ju (JUSTIFY-MODE)	Causes left and right justification of output.	31	y	Concatenate-Justify Mode
.li (LITERAL)	Insures that the next input line(s) is read as a text line by SCRIPT/370.	60		n=1
.ll (LINE-LENGTH)	Specifies the number of characters, including blanks, in each subsequent line.	20	y	60

Control Word	Function	Page Reference	Implicit Break	Standard Setting or Default Value
.ls (LINE-SPACING)	Specifies the number of blank lines to be inserted after each subsequent output text line.	38	y	Single Space Mode
.nb (NO-BALANCED-COLUMNS)	Causes columns of lines forced out by the equivalent of a page eject to be unbalanced.	66		Balanced Column Mode
.nc (NO-CONCATENATE-MODE)	Prevents concatenation of input lines.	32	y	Concatenate-Justify Mode
.nf (NO-FORMAT-MODE)	Permits "as-is" output text by preventing concatenation and left-and-right justification.	34	y	Concatenate-Justify Mode
.nj (NO-JUSTIFY-MODE)	Prevents padding between words of input text lines with blank characters.	33	y	Concatenate-Justify Mode
.ob (ODD-PAGE-BOTTOM-TITLE)	Specifies a title line for the bottom of the current page, if it is odd numbered, and all subsequent odd numbered pages.	25		
.of (OFFSET)	Provides a technique for indenting all but the first line of a section.	42	y	Left Margin. n=1
.op (ODD-PAGE-EJECT)	Causes one or two page ejects such that the next page will be odd numbered.	40	y	
.ot (ODD-PAGE-TOP-TITLE)	Specifies a title line for the top of each subsequent odd numbered page.	24		
.pa (PAGE-EJECT)	Causes a page eject and optionally alters the internal and external page numbers.	39	y	

Control Word	Function	Page Reference	Implicit Break	Standard Setting or Default Value
.pl (PAGE-LENGTH)	Specifies the number of lines, including margins, on each output page.	21	y	66
.pn (PAGE-NUMBERING-MODE)	Permits control of both external and internal page numbering.	26		Arabic numeral
.ps (PAGE-NUMBER-SYMBOL)	Specifies the character used for the page symbol in the top and bottom titles.	25		&
.qu (QUIT)	Causes SCRIPT/370 processing to be terminated immediately.	50		
.rc (REVISION-CODE)	Designates a revision code marker to be printed at specified places along the left margin.	57		
.rd (READ-TERMINAL)	Permits one or more data lines to be entered from the terminal during SCRIPT/370 output.	70	y	n=1
.re (RESTORE-STATUS)	Restores status of SCRIPT variables from a push down stack created by SAVE-STATUS.	53		
.ri (RIGHT-ADJUST	Causes lines to be printed flush with the right margin	34	y	
.sa (SAVE-STATUS)	Saves the status of SCRIPT variables.	52		
.se (SET-SYMBOL)	Defines and assigns values to symbolic names, interfacing with the macro capabilities of SCRIPT/370.	73		

Control Word	Function	Page Reference	Implicit Break	Standard Setting or Default Value
.sp (SPACE-LINES)	Specifies the number of blank lines to be inserted before the next output line.	37		n=1
.ss (SINGLE-SPACE-MODE)	Specifies that subsequent formatted output will be single spaced.	37	y	Single-Space Mode
.su (SUBSTITUTE-SYMBOL)	Enables the macro capabilities of SCRIPT/370 by selectively causing substitution of defined set symbols.	75		OFF
.tb (TAB-SETTING)	Specifies the "logical" tabs used when the document is printed or typed by SCRIPT/370.	44	y	5,10,15...75
.te (TERMINAL INPUT)	Permits one or more text lines to be entered from the terminal during SCRIPT/370 processing.	70		n=1
.tm (TOP-MARGIN)	Specifies the number of lines in the top margin.	21	y	6
.tr (TRANSLATE CHARACTER)	Specifies the final output representation of any input character.	61		
.tt (TOP-TITLE)	Specifies a title for the top of each subsequent output page.	70		
.ty (TYPE-ON-TERMINAL)	Types one line of information on the user's terminal during SCRIPT/370 processing.	72		
.un (UNDENT)	Causes the following text line to be printed or typed farther to the left than the current indent.	43		

APPENDIX C. Additional Information Regarding SET-SYMBOL and SUBSTITUTE-SYMBOL Control Words

RESERVED SYMBOLS

There are 8 special reserved symbols that are automatically initialized, each time SCRIPT is used, to the current values for: year (SYSYEAR), month (SYSMONTH), day of year (SYSDAYOFY), day of month (SYSDAYOFM), day of week (SYSDAYOFW), hour of day (SYSHOUR), minute of hour (SYSMINUTE), and seconds of minute (SYSSECOND). These symbols are tabulated below:

Symbol	Value	Meaning	Example of value
SYSYEAR	XX	year	If this is 1971, SYSYEAR has the value 71.
SYSMONTH	XX	month	If this is Feb. 8, 1971, SYSMONTH has the value 02.
SYSDAYOFY	XXX	day of year	If this is Feb. 8, 1971 (39th day of the year), SYSDAYOFY has the value 039.
SYSDAYOFM	XX	day of month	If this is Feb. 8, 1971, SYSDAYOFM has the value 08.
SYSDAYOFW	X	day of week	If this is Monday, Feb. 8, 1971, SYSDAYOFW has the value 2 (Sunday is considered 1st day of week).
SYSHOUR	XX	hour of day	If it is 7:30 P.M., SYSHOUR has the value 19 (24 hour clock, 7:30 A.M. would have value 07).
SYSMINUTE	XX	minute of hour	If it is 7:30 P.M., SYSMINUTE has the value 30.
SYSSECOND	XX	seconds of minute	If it is 7:30:15, SYSSECOND has the value 15.

The table above only indicates the initial values of the symbols. Although they may be changed and used exactly as regular set symbols, it is not recommended.

Each symbol has a fixed size value field. If its current value is less that the field size, leading zeroes are provided (e.g., SYSDAYOFY was 039 above not just 39). The leading zeroes can be easily removed by a SCRIPT control word such as

```
     .SET SYSDAYOFY = &SYSDAYOFY + 0
```

since leading zeroes are automatically deleted after an arithmetic evaluation.

The numeric values of the special symbols can be converted to other forms as shown in the following example:

```
     .set m01='January';.set m02='February'
     .set m03='March'; ...;.set m12='December'
     .set alphamonth = &m&sysmonth
     .sub;This is now &alphamonth already.
```

This above sequence would result in the line: "This is now February already" if the current date was Feb. 8, 1971. The beginning of the sequence defines a sequence of set-symbols, m01, m02, ..., m12, whose values correspond to the alphabetic representation of the 12 months. The expression "&m&SYSMONTH" involves a double substitution; first it becomes "&m02" and then "February". A very similar technique can be used to convert the days of the week to their alphabetic equivalents (e.g., Monday instead of 2).

EXAMPLES USING SET SYMBOLS

EXAMPLE 1

```
     .set version1 = 'March 1971'
     .set version2 = 'June 1971'
     .set number=2
     ...
     .set vname = 'version&number'
     .set vdate1 = &version&number
     .set vdate2 = &&vname
```

The symbol vname will end up with the value "version2," the symbols vdate1 and vdate2 will end up with the value "June 1971."

EXAMPLE 2

```
     .set < = '&left.&n&n'
     .set > = 'n..&right'
     .set left = '(';.set right = ')'
     .set n1='i';.set n2='ii';.set n3='iii';.set n4='iv';...
     .set nn = 0
     ...
     .set nn = &nn + 1
     .su 1; Reference &<.&> has more data.
```

The expanded line will be: "Reference (i) has more data."

This result involves 6 substitutions, as follows:

```
     &<.&>
     &left.&n&n>
     (&n&n&>
     (&n&nn..&right
     (&n1.&right
     (i&right
     (i)
```

This may seem to be a particularly obscure way of accomplishing a simple task, but it does illustrate the flexibility possible. In fact, this type of multiple substitution can be very useful in conjunction with the CONDITIONAL-SECTION and TERMINAL-INPUT control words for altering specific parts, such as changing the (i) form to be <i> instead.

EXAMPLE 3

```
     .SET code_word=0;.SET code_word()=&        (page 2)
     ...
     .set code_word()='*&.*'                    (page 6)
     ...
     .set code_word()=&                         (page 14)
```

The successive elements of the array code_word are assigned the values of the page numbers that they occurred on. In particular, code_word(0) = 3, code_word(1) = 2, code_word(2) = *6*, and code_word(3) = 14.

```
     .sub on
     .of 20
     Code_word references (&code_word.) ...... &code_word(*)
     ...
```

would become

```
     Code_word references (3) ...... 2, *6*, 14
```

as a result of array substitions.

SH20-1114-0

SCRIPT/370 Text Processing Facility Under Virtual Machine Facility/370 PD & OM Printed in U.S.A. SH20-1114-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)