ABSTRACT

The University of Waterloo version of SCRIPT is a powerful text formatting program. SCRIPT is very easy to use for "simple" documents such as memos and small papers; with just a few SCRIPT commands (called "control words"), you can satisfy the formatting requirements of most documents.

SCRIPT is also extremely useful for the preparation of large documents such as manuals, theses, and other research and instructional publications. By using some of the "programming language" capabilities of SCRIPT, it is possible to make SCRIPT generate a table of contents and an index, or to write your own control words. This makes SCRIPT particularly useful for the construction and maintenance of large documents, where you wish to be able to standardize the layout and make significant changes to a component of the document (add a chapter, delete a chapter, or re-order large sections of text) without affecting the rest of the document.

To use SCRIPT, you must use a text editor to create the input file. There are a number of text editors available to you: WITS, WYLBUR, the VM/370 CMS Editor, or CMS WYLBUR. If you are not familiar with any of these editors, introductory documentation may be obtained from any of the Consulting Offices, and reference manuals are available through the Computer Reference Room.

In all systems, it is possible to direct the output to a high-speed line printer. This is usually desirable for "proofreading" copies; it is also often desirable for the "final copy" of a large document. It is possible for you to request that the document be printed on white paper. Special mylar printer ribbons may be purchased which result in a very high-quality printout acceptable for reproduction by photo-offset printing; or, if original copies are required, a heavy-bond paper is also available. Information on purchasing and using mylar ribbons and/or special paper can be obtained by contacting the I/O Services Office (MC 1070, extension 2686).

Under CMS, the formatted document may be typed at your terminal, stopping at the bottom of each page for the next sheet of paper.

This "Introduction to SCRIPT" is intended to introduce you to SCRIPT, and therefore will not discuss all of the features of SCRIPT. Most of the SCRIPT control words discussed in this Introduction have additional capabilities; they are described herein in the ways in which they are likely to be used most often.

The version of SCRIPT in existence at the time this Introduction was printed was:

UOW SCRIPT - VERSION(3.2) 78JAN13

1. INTRODUCTION

SCRIPT is a computer program which accepts input, processes it, and produces output.

SCRIPT recognizes only two different types of input: text and control words. The processing which SCRIPT performs is called text formatting. The output produced by SCRIPT is a document in which all of the text has been formatted according to the layout instructions specified by the control words.

2. CREATING THE INPUT

The input to SCRIPT is a collection of text and control words which you type at a typewriter terminal or visual-display terminal by using a text editor. A text editor is a computer program which allows you to enter, correct, delete, add, and save the text and control words.

When you are entering the input, it is important that you start each sentence of the text on a new input line. This makes subsequent insertions or deletions of sentences much easier to perform.

2.1 CONTROL WORDS AND TEXT

Each time SCRIPT reads an input line, it looks at the character in column 1. If that character is a ".", SCRIPT considers the input to be a control word. Otherwise, the input line is treated as text.

Each SCRIPT control word is of the form

```
.xx operand(s)
```

The "." is called the control word indicator literal, and "xx" is a 2-character control word. Some SCRIPT control words have one or more operands. The first operand begins in the second position after the control word; each operand is separated by a blank.

2.2 PUTTING MORE THAN ONE CONTROL WORD ON THE SAME INPUT LINE

This may be done by separating each control word with a semicolon (;). SCRIPT refers to the semicolon as the control word separator. You can also put text on this same input line. In general, then, an input line which contains control words is entered as follows:

```
.xx operands;.yy operands;.zz operands;text
```

where "xx", "yy", and "zz" are control words (with or without operands). You cannot put SCRIPT control words after the text on an input line.

3. THE FORMATTING PROCESS

The basic step in the text formatting process is the creation of an output line.

3.1 THE LINE LENGTH

The SCRIPT ".ll" LINE LENGTH control word specifies the number of character positions in the output line. The line length for this Introduction was defined with the control word

```
.ll 60
```

The operand can be an absolute numeric value, as above, or a signed numeric, such as

```
.ll +20
```

which adds 20 to the current LINE LENGTH value. If no operand is specified, the default of sixty will be assumed. Normally, you will specify the value for ".ll" at the beginning of the document and then not change it.

3.2 SINGLE- AND DOUBLE-SPACING

By default, SCRIPT single-spaces the output lines. The SCRIPT ".ds" DOUBLE SPACING control word will cause a blank line to be generated between each output line. You can get SCRIPT to revert to single-spacing by issuing the ".ss" SINGLE SPACING control word.

3.3 STARTING A NEW PARAGRAPH

The easiest way to define the start of a new paragraph is to use the SCRIPT ".pp" PARAGRAPH START control word. For example,

```
.pp;This is the start of a new paragraph ...
```

produces:

This is the start of a new paragraph. The ".pp" control word skips a line and indents the line of text 3 spaces.

3.4 UNDERSCORING AND CAPITALIZING

The SCRIPT ".us" UNDERSCORE control word, ".up" UPPERCASE control word, and ".uc" UNDERSCORE AND CAPITALIZE control word allow you to underscore and/or capitalize a single line of text. They are of the form

```
.us line of text to be underscored
.up line of text to be capitalized
.uc line of text to be underscored (and capitalized)
```

and produce the following effects:

line of text to be underscored
LINE OF TEXT TO BE CAPITALIZED
LINE OF TEXT TO BE UNDERSCORED (AND CAPITALIZED)

With ".us" and ".uc", it is possible to underscore only selected parts of the line. This requires that the SCRIPT ".ud" UNDERSCORE DEFINITION control word be used to define a special "escape" character which will indicate that underscoring is to be turned off and on again. For example, if

```
.ud set |
```

has already been specified, then

```
.us line |of text |to be |under|scored
.uc line |of text |to be |under|scored and capitalized
```

will produce

line of text to be underscored
LINE OF TEXT TO BE UNDERSCORED AND CAPITALIZED

By default, these control words will underscore all characters except those shown in Table 1. If you want to add any of these characters to the list to be underscored auto-

matically, you can use the "on" operand of ".ud" to do it. For example, if you want SCRIPT to include parentheses in the list of characters which will be underscored if they appear in the operand string of a ".uc" or ".us", then specify:

```
.ud on ( )
```

TABLE 1

Characters Not Underscored (by default)

Character	Graphic Symbol	Hex Code
backspace	none	16
blank		40
comma	,	6B
colon	:	7A
double quote	"	7F
exclamation mark	!	5A
left brace	{	8B
left bracket	[	AD
left parenthesis	(	4D
period	.	4B
question mark	?	6F
right brace	}	9B
right bracket	]	BD
right parenthesis	)	5D
semicolon	;	5E
tab	none	05
underscore	_	6D

3.5 KEEPING A BLOCK OF TEXT TOGETHER ON THE SAME PAGE

In many instances, it is necessary to ensure that a block of text not be split across a page boundary -- for example, the text (or spaces) for charts, figures, or tables. There are two methods which may be used to prevent output lines from being split over two pages.

The SCRIPT ".cp" CONDITIONAL PAGE control word can be used to start a new page if there is not enough room on the current page. For example, when SCRIPT encounters the input sequence

```
.cp begin
input text and other SCRIPT control words
.cp end
```

it formats the "input text and other SCRIPT control words" in an internal "work area" so that it knows how many lines of output this input sequence has created. If there is not sufficient room for these output lines on the current page, SCRIPT leaves the rest of the current page blank, and prints these already-formatted output lines at the top of the next page.

If you know precisely how many output lines are needed, it is more efficient to use ".cp" with a numeric operand. For example,

```
.cp 10
```

will start a new page if there are less than 10 lines available on the current page.

The ".cp" control word, as mentioned above, will cause the remainder of the current page to be left blank if a new page must be started. This is often highly undesirable. If you use the SCRIPT ".fk" FLOATING KEEP control word

```
.fk begin
input text and other SCRIPT control words
.fk end
```

SCRIPT will format the "input text and other SCRIPT control words" internally, and print the resulting output lines on the current page if there is room for them. If there is insufficient room, SCRIPT will save them for printing on the next page, and will fill the current page with the output created by the text and control words which follow the FLOATING KEEP begin/end sequence.

3.6 STARTING A NEW PAGE

You can force the output to start on a new page by using the SCRIPT ".pa" PAGE control word.

```
.pa
.pa +5
.pa 1
```

The first example skips to the top of the next page; you might wish to do this, for example, to cause a new chapter to start on a new page. The second example skips to the top of the next page and adds five to the page number. This may

be useful if you intend to insert five pages which are to be produced in some other fashion. The third example skips to the top of the next page and resets the page number to "1".

3.7 BLANK LINES (SPACES)

The SCRIPT ".sp" SPACE control word creates blank output lines. The operand specifies the number of blank lines desired.

```
.sp 5
.sp
```

The first example produces five blank lines, the second example the default of one. If double-spacing were in effect, the first example would produce 10 blank lines and the second would produce 2. If you want to leave a specific number of blank lines independent of whether you may decide later on to switch the document from single- to double-spacing (or vice versa), then use the "absolute" operand, as in

```
.sp 5 a
```

In this form, ".sp" will produce the specified number of "absolute" blank lines, independent of the line-spacing mode.

3.8 BLANK LINES AT THE TOP OF A PAGE

By default, SCRIPT prevents the printing of blank lines at the top of a page. If you want to allow this, use the ".le" LEADING SPACES control word. For example, you might begin each new chapter with the input sequence

```
.pa;.le yes;.sp 5 a;.le no
```

The ".le yes" allows the printing of blank lines at the top of the page, and the ".le no" suppresses them again.

3.9 PRINTING EMPTY PAGES

By default, SCRIPT does not print pages which are empty. You can use the SCRIPT ".em" EMPTY PAGES control word to force SCRIPT to print empty pages. For example,

```
.pa;.em yes;.pa;.pa;.em no
```

leaves the rest of the current page blank, tells SCRIPT to print empty pages, creates two empty pages, and then suppresses the printing of empty pages again.

3.10 BREAKS

A "break" in the output text is a place where the current output line has been printed, no matter how full it may be, and a new output line begun. There are four ways in which you can specify or cause a "break" in the output text to occur. The first way is to use the SCRIPT control word BREAK,

```
.br
```

The second way is to use a SCRIPT control word which, in addition to doing other things, also implies a break (such as the ".sp" control word).

The third way is to start an input line with one or more blanks. This starts a new output line, indented by the number of blanks with which the input line starts.

The fourth way is to start an input line with a tab character. This starts a new output line, indented to the first defined tab stop.

4. THE DEFINITION AND USE OF TABS

4.1 DEFINING TAB-STOP POSITIONS

SCRIPT does not know about any physical tab-stop settings you may have on your output device. Instead, you must use the SCRIPT ".tb" TAB control word to define the positions in the output line which are to be used as tab-stop settings. By default, tab-stop settings are defined in positions 5, 10, 15, 20, ..., 80, as in

```
.tb 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80
```

When SCRIPT encounters a tab character while putting words onto the output line, it inserts the number of blanks necessary to move over to the right so that the character after the TAB will appear in the next tab position in the output line.

For example, with the default tab-stop positions, a TAB in the first column of an input line will create a new output line in which the first 4 positions are blank.

4.2 ENTERING TAB CHARACTERS IN THE INPUT

If your input device has a TAB key, you can use it to enter the TAB character simply by depressing the TAB key. If your input device does not have a TAB key, then you can make use of the SCRIPT ".tb set" TAB SET control word to define a special character which will be treated as the TAB character whenever it is encountered in the input. For example, if you were to include the control word

```
.tb set $
```

at the beginning of your input file, you would be defining the "$" to be the "TAB" character -- any time SCRIPT encounters an "$" in an input line, it will automatically insert the number of blanks necessary to move over to the next defined tab-stop position.

4.3 TABS WITH A "FILL STRING"

In addition to inserting blanks as it moves over to the next tab-stop position, SCRIPT also provides the ability for you to specify a "fill string" (one or more non-blank characters) to be used instead of blanks. For example, if you had defined a tab-stop position with

```
.tb '. '50
```

then the input sequence

```
.tb set $
.sp
hello $ there
.br
how are $ you
```

would produce

```
hello . . . . . there
how are . . . . . you
```

4.4 RIGHT-ALIGNING TABS

SCRIPT also gives you the ability to specify a tab-stop position to which text will be right-aligned. For example, if you had defined the above tab-stop position as

```
.tb '. '60r
```

then the exact same input sequence would produce

```
hello . . . . . . . . . . . . . . . . . . . . . . . . . there
how are . . . . . . . . . . . . . . . . . . . . . . . . you
```

The TAB between the words "hello" and "there" causes SCRIPT to "push" the word "there" over to the right until its last character is in column 60.

4.5 CENTERING TABS

If the tab-stop position is followed by the letter "C", such as

```
.tb 30c
```

then SCRIPT will align the text following the TAB so that it is centered at that position. For example,

```
.tb set ¬;.tb 30c
the value is¬123456789
.br
and the next is¬123
```

will cause the number "123456789" to print so that the "5" is in column 30, as shown below.

```
the value is             123456789
and the next is             123
```

4.6 CHARACTER-ALIGNING TABS

If the tab-stop position is followed by a delimited single character, such as

```
.tb 30'.'
```

then SCRIPT will align the text following the TAB to the first occurrence of that character. For example,

```
.tb set ¬;.tb 30'.'
the value is¬12.456789
.br
and the next is¬123.56
```

will cause the number "12.456789" to print so that the "." is in column 30, as shown below.

```
the value is              12.456789
and the next is          123.56
```

5. THE OUTPUT PAGE

5.1 THE LEFT AND RIGHT MARGINS

The SCRIPT ".ad" ADJUST control word may be used to define the starting position of the output line with respect to the physical left margin (the physical left margin of this Introduction is actually 10 spaces to the left of this paragraph). The term "physical left margin" refers to the leftmost possible print position of the output device (terminal or printer). The term "left margin" by itself is used in this Introduction to refer to the starting position of the output line on the output device, as defined by the ADJUST value. Since the initial ADJUST value is zero, the left margin and the physical left margin are therefore identical at the start of SCRIPT processing. Specifying a nonzero ADJUST value moves the left margin the specified number of spaces to the right of the physical left margin. The length of the output line, as defined by ".ll", is not affected by the ADJUST value.

For example, this document contains the SCRIPT control words

```
.ad 10;.ll 60
```

at the beginning of the input file; this creates an output line which begins in print position 10+1 and ends in print position 10+60. This ADJUST value was specified in order to allow a sufficient margin on the left of the page for binding.

The right margin is one position to the right of the position defined by the sum of the ADJUST and LINE LENGTH values.

5.2 THE TOP AND BOTTOM MARGINS

The SCRIPT ".tm" TOP MARGIN control word may be used to define the number of lines to be left at the top of each output page. The top margin for this Introduction was created with the control word

```
.tm 6
```

The SCRIPT ".bm" BOTTOM MARGIN control word may be used to define the number of lines to be left at the bottom of each output page. The bottom margin for this Introduction was created with the control word

```
.bm 6
```

5.3 THE TEXT AREA

The text area is that area of the output page bounded on the left by the left margin (as defined by the ADJUST value), on the right by the right margin (as defined by the sum of the ADJUST and LINE LENGTH values plus 1), on the top by the top margin (as defined by the TOP MARGIN value), and on the bottom by the bottom margin (as defined by the BOTTOM MARGIN value).

It is this "text area" into which SCRIPT places its output lines. Whenever the text area becomes full, SCRIPT automatically skips to a new page.

6. THE FORMATTING MODES

The criterion which SCRIPT uses in attempting to "fill" the output line is called the "formatting mode". To show the effects of the various formatting modes, the text in the following sections has been formatted by the same control words which the text is describing.

6.1 UNFORMATTED MODE

To begin, we will start with "unformatted mode", by issuing the SCRIPT control word

```
.fo no
```

In the "unformatted mode", SCRIPT does no "text formatting"; it
merely copies the text exactly as it appeared in the input lines.
We are now in "unformatted mode".
This is useful when we want to enter the text for a table or figure or example, and we
want the output to appear exactly as we entered it.

In "unformatted" mode,
".ll" has no effect on the length of the output line.
For text in sentence-and-paragraph form, we want something else.

6.2 CONCATENATED MODE

If we now specify the SCRIPT control word CONCATENATE,

```
.co
```

then SCRIPT puts as many words as possible on an output line. Note that, although no output line is longer than the ".ll" value, the right margin is still "ragged". We can get even more text onto each output line if we also add "hyphenation" to the "concatenation" process.

6.3 HYPHENATION

SCRIPT has three modes of hyphenation. The hyphenation mode is specified with the HYPHENATE control word:

```
.hy off
.hy user
.hy on
```

The "off" operand causes SCRIPT to perform no hyphenation.

The hyphenation mode specified by the "user" operand is the default hyphenating mode. If SCRIPT encounters a hyphenated word and it cannot get the whole word on the current output line but can get the first part and the hyphen, it will do so, and put the rest of the word on the next output line.

The "on" operand causes SCRIPT to perform automatic hyphenation. SCRIPT attempts to detect all permissible hyphenation points in a word*, and then puts as much of the word as possible on the current line, followed by a hyphen. The rest of the hyphenated word is used to start the next output line.

SCRIPT's hyphenation attempts do not always result in a hyphenation point which is aesthetically pleasing. Unfortunately, the human's rules for deciding which hyphenation points "don't look right" are not sufficiently well-defined to be programmed. To overcome this, there is an "exception dictionary" which you may choose to imbed (using the SCRIPT ".im" control word) if you are formatting your document with ".hy on", by specifying

---

*The part of Waterloo SCRIPT which performs hyphenation was adapted from the IBM "HYPHENATION/360" computer program.

```
.im syshyph
```

at the beginning of your document. The "syshyph" file contains several thousand "exception" words. When hyphenation is "on", SCRIPT first checks a word it wants to hyphenate to see if it is in the exception dictionary. If it is, SCRIPT considers only the hyphenation points defined for that word in the exception dictionary (which may, of course, be none). This Introduction was formatted with the exception dictionary.

The SCRIPT ".hy" control word has several additional operands for modifying the hyphenation process and the exception dictionary. It is beyond the intended scope of this Introduction to discuss them here; you can learn more about the ".hy" control word (and other control words) in the SCRIPT Reference Manual mentioned later in this document.

6.4 CONCATENATED-AND-JUSTIFIED (FORMATTED) MODE

In the "concatenated-and-justified" mode, SCRIPT adds another step beyond that of concatenation -- after it realizes that it cannot get any more on the output line, it then inserts extra blanks between words. We can add justification to the concatenation process by using the SCRIPT control word JUSTIFY

```
.ju
```

and SCRIPT will include justification from this point on. This is the default formatting mode. SCRIPT refers to this "concatenated-and-justified" mode as the "formatted" mode. Rather than specifying both ".co" and ".ju", we could create the same effect by specifying the SCRIPT control word FORMAT

```
.fo
```

7. POSITIONING BLOCKS OF TEXT IN THE TEXT AREA

7.1 CENTERING LINES OF TEXT

The SCRIPT ".ce" CENTRE control word, as used in the following input sequence

```
.ce begin
input text and other SCRIPT control words
.ce end
```

will cause each text line in the range of the "begin/end" to be centered on a separate output line. If only one input line is to be centered, it can be entered as

```
.ce line of text to be centered
```

A numeric operand can be used, for a specific number of input lines of text.

7.2 RIGHT-ADJUSTING LINES OF TEXT

The SCRIPT ".ra" RIGHT ADJUST control word causes input text to be right-adjusted on output. It is used in the same manner as ".ce" above. For example,

```
.ra this line of text
```

produces:

this line of text

7.3 INDENTING BLOCKS OF TEXT

The output line normally starts at the left margin (as defined by the ADJUST value). To indent output lines a specific number of spaces, use the SCRIPT ".in" INDENT control word. This indentation will remain in effect until another ".in" control word is encountered. Consider the following examples:

```
.in 10
```

This causes all output lines to be indented ten spaces from the left margin. The control word

```
.in +5
```

adds five to the current INDENT value, so that output lines are now indented 15 spaces from the left margin. The control word

```
.in
```

used without an operand causes the INDENT value to be reset to zero, so that the output lines now begin at the left margin again.

Note that the INDENT control word, when used in the above ways, does not alter the positioning of the right margin. If that is desired, the control word may be used in the form

```
.in 10 -5
```

which causes the left margin to be indented 10 spaces from the ADJUST value and the right margin to be indented five spaces to the left of the LINE LENGTH value.

In no case does the INDENT value ever alter the LINE LENGTH value.

7.4 INDENTING A SINGLE LINE OF TEXT

To indent only the next line of text, use the SCRIPT ".il" INDENT LINE control word. For example,

```
.il +3;will cause the next line of ...
```

produces:

will cause the next line of text to be indented 3 more than the value of ".in" currently in effect.

7.5 OFFSETTING LINES OF TEXT

To offset all output lines after the next one is printed, use the SCRIPT ".of" OFFSET control word. The ".of" value is relative to the current ".in" INDENT value, and remains in effect until the next ".of" or ".in" control word is used. A list of points could be formatted by using OFFSET as follows.

```
.of 4
(1) first text input line for point
second text input line for point
.of 4
(2) first text input line for second point
second text input line for second point
.of
```

The explanation below was formatted using the above technique.

(a) The next output line after the ".of 4" is encountered will begin at the INDENT value, but output lines thereafter, until the next ".of" is encountered, will be offset 4 spaces from the INDENT value.

(b) The second ".of 4" is necessary to cancel the effect of the previous ".of 4", so that the "(2)" will be printed beginning at the INDENT value, with output lines thereafter being offset 4 spaces to the right.

(c) The ".of" after the text for the second point re-sets the current OFFSET value to zero.

7.6 OUTPUT OVERLAY PATTERNS

The SCRIPT ".oo" OUTPUT OVERLAY control word replaces blanks in output columns with a specified overlay pattern. For example, the input sequence

```
.in 10
.oo 1 OUTPUT
.oo 1 OVERLAY
.oo 1 Example
causes the output to be indented
10 spaces ...
```

produces:

OUTPUT
OVERLAY
Example

causes the output to be indented 10 spaces from the left margin and defines a sequence of three "output overlay" patterns, each of which is to be printed "1" time, as indicated by the "1" following the ".oo". The "output overlay" patterns are printed in the order in which they are defined.

The ".oo" control word can be used to create marginal notes in indented text, as shown below.

```
.in 10
The following lines of text will be indented 10 spaces
and the ".oo" OUTPUT OVERLAY control word will
be used to position the word "here"
.oo 1 here==>
on the line in which the word appears.
Note that ".oo" does not create a break in the
formatting of the text.
```

produces:

here==>

The following lines of text will be indented 10 spaces and the ".oo" OUTPUT OVERLAY control word will be used to position the word "here" on the line in which the word appears. Note that ".oo" does not create a break in the formatting of the text.

7.7 THE "SIGNIFICANT BLANK": KEEPING THINGS ON THE SAME LINE

When it is necessary to ensure that SCRIPT keeps a string of characters which includes blanks together on the same output line, the SCRIPT ".tr" TRANSLATE ON OUTPUT control word must be used. For example, the input sequence

```
It is absolutely essential that the characters
.tr $ 40
X$=$a$+$b$+$c$+$d appear on the same
line of output, and that SCRIPT not insert
any extra blanks in the string.
.tr
```

produces:

> It is absolutely essential that the characters X = a + b + c + d appear on the same line of output, and that SCRIPT not insert any extra blanks in the string.

The ".tr $ 40" causes SCRIPT to translate all occurrences of "$" to a blank (40 is the machine-code representation for a blank) after each line has been formatted and just before it is printed. The ".tr" all by itself terminates this "translate on output" process. Be sure not to terminate it right after specifying the string to which you want it to apply, because it probably hasn't printed yet; wait a few lines, as was done above.

7.8 PUTTING A BOX AROUND TEXT

The SCRIPT ".bx" BOX control word allows you to create boxes. For example, the input sequence

```
.cp begin
.bx 1 60
.sp;.in +3 -3
This text will have a box around it,
in columns 1 and 60,
compliments of the ".bx" control word.
.in -3 +3;.sp
.bx off
.cp end
```

produces:

This text will have a box around it, in columns 1 and 60, compliments of the ".bx" control word.

7.9 WIDOWING

If you want SCRIPT to attempt to prevent the first line of a block of text from being the last line on a page, or prevent the last line of a block of text from being the first line on a page, then specify the SCRIPT ".wd" WIDOW control word

```
.wd
```

8. FOOTNOTES

Many documents require the use of footnotes. Using SCRIPT, it is possible to enter the text of the footnote immediately after the word to which the footnote refers. SCRIPT formats the footnote text into an internal "work area", counts the number of output lines which result, and leaves sufficient room at the bottom of the text area to ensure that the footnote text is printed at the bottom of the same* page. For example, the text for the footnote at the bottom of this page was created by the following sequence of text and control words:

```
at the bottom of the same*
.fn begin
.of 1
*It is possible that some or all of the
footnote may spill onto the next page or
for the line containing the reference to get
pushed over.
.fn end
page.  For example, the text for the footnote ...
```

----

*It is possible that some or all of the footnote may spill onto the next page or for the line containing the reference to get pushed over.

The SCRIPT ".fn begin" FOOTNOTE BEGIN control word defines the start of the footnote. The output resulting from the input text and SCRIPT control words thereafter is saved in SCRIPT's internal "work area". The SCRIPT ".fn end" FOOTNOTE END control word terminates this "internal formatting" of the footnote and restores the formatting mode and the values of control words such as ".in" and ".of" to whatever their values were before the ".fn begin" was encountered.

By default, SCRIPT separates the last line of text from the first line of the footnotes by a blank line followed by a line with a centered row of dashes, followed by another blank line. You can change this by using the "set" operand of the ".fn" control word. For example, the following input sequence

```
.fn set 4
.fn set 1 /--------------------///
.fn set 2 ////
.fn set 3 ////
.fn set 4 ////
```

sets the number of "footnote separator" lines to 4, the first of which contains 20 left-justified dashes. The 2nd, 3rd, and 4th separator lines will be entirely blank. Whatever you place between the first and second slash will be left-justified on that separator line. Whatever you place between the second and third slash will be centered. Whatever you place between the third and fourth slash will be right-justified.

9. PAGE NUMBERING: TOP AND BOTTOM TITLES

9.1 TITLE LINES IN THE TOP MARGIN

The SCRIPT ".hs" HEADING SPACE control word defines the number of heading lines which may be printed in the TOP MARGIN area. The HEADING SPACE for this Introduction was defined by

```
.hs 0
```

The SCRIPT ".hm" HEADING MARGIN control word is used to define the number of blank lines to be left between the HEADING SPACE area and the first line of the text area. The HEADING MARGIN for this Introduction was defined with

```
.hm 1
```

Note that the sum of the ".hs" and ".hm" values may not be greater that the ".tm" value. If you currently have a ".tm" value of 5 and you want to define

```
.hm 3;.hs 3;.tm 7
```

it must be done in the order

```
.tm 7;.hm 3;.hs 3
```

The SCRIPT ".tt" TOP TITLE control word allows you to define lines of text to be printed in the HEADING SPACE of each page. This control word is of the form

```
.tt n /left string/centre string/right string/
```

The value "n" specifies the line number in the HEADING SPACE on which this heading line is to be printed; if a value is not specified, "1" is assumed. The characters specified in place of the three "strings" will appear left-justified, centered, and right-justified on that output line.

9.2 TITLE LINES IN THE BOTTOM MARGIN

Three similar control words exist for the BOTTOM MARGIN area of each page. The SCRIPT ".fs" FOOTING SPACE control word specifies the number of lines to be used for bottom titles. The ".fm" FOOTING MARGIN control word specifies the number of blank lines to be left between the last line of the text area and the first line of the FOOTING SPACE area. The ".bt" BOTTOM TITLE control word is used to define the title lines to be printed in the FOOTING SPACE area.

A word of caution is in order. Do not confuse footnotes and footings. Footnotes are printed at the bottom of the text area; footing (bottom title) lines are printed in the FOOTING SPACE area.

9.3 PAGE NUMBERING

If page numbering is required, then place a percent sign (%) into one of the three strings of characters in the top and/or bottom titles. The "%" will be replaced with the current page number when SCRIPT prints the title line. For example,

```
.tt ////
.bt //- % -//
```

were used to define the top and bottom titles for this Introduction. By default, SCRIPT uses

```
.tt ///PAGE %/;.bt ////
```

and arabic numerals for the page numbers. You can specify the numbering style for the page numbers by specifying any of:

```
.pn arabic
.pn roman
.pn roman upper
```

10. MULTIPLE-COLUMN OUTPUT

The text you read in newspapers and magazines is formatted in multiple columns per page. A number of journals also use this style, and some of them require that authors submit camera-ready copy which is already formatted in multiple columns (some IEEE publications, for example). SCRIPT can produce output in up to nine columns on a page.

10.1 DEFINING THE PAGE LAYOUT

To use this multiple-column facility, you must specify the following:

1. the ".cl" COLUMN LENGTH control word, to define the number of characters to be placed on each line in a column,

2. the ".cd" COLUMN DEFINITION control word, to define the number of columns on the page and the displacement of each relative to the left margin, and

3. the ".ll" LINE LENGTH control word, to define the total number of character positions on the resulting multiple-column output line.

As an example, imagine that you wish to format output in two 60-character columns per page, with a 10-character "gutter" between them, resulting in a total line length of 130 characters. This can be achieved by the SCRIPT control words

```
.cl 60;.cd 2 0 70;.ll 130
```

which define two 60-character columns starting at displacements 0 and 70 on the 130-character line. The first column occupies positions 1 through 60, the gutter occupies positions 61 through 70, and the second column occupies position 71 through 130.

10.2 STARTING A NEW COLUMN

Working in multiple-column mode requires a change in the approach taken to formatting a document. As in single-column mode, the ".pa" PAGE EJECT and ".cp" CONDITIONAL PAGE control words operate on a page basis. However, there are also ".cb" COLUMN BEGIN and ".cc" CONDITIONAL COLUMN control words which operate on a column basis, and you should use them instead of ".pa" and ".cp" (unless you really want to start a new page). In multiple-column mode, text defined with a ".fk" FLOATING KEEP sequence will be promoted to the next column if it won't fit in the current column, and footnotes are printed at the bottoms of columns instead of pages.

10.3 BALANCING COLUMNS

By default, SCRIPT attempts to "balance" the columns on a page before it prints the page. For example, if the first column on a page is full and the second is only half full, then SCRIPT will move some lines from the bottom of the first column to the top of the second column, to create two columns which are each three-quarters full, before it prints the page.

You can use the ".bc" BALANCE COLUMNS control word (with an ON or OFF operand) to explicitly prevent (OFF) or allow (ON) column-balancing.

There are some things which prevent SCRIPT from attempting to balance the columns on a page even when column-balancing is "ON". For example, if a column contains a footnote or was terminated with a ".cb" COLUMN BEGIN, then that column cannot be balanced.

10.4 SWITCHING BETWEEN SINGLE AND MULTIPLE COLUMNS

The ".sc" SINGLE COLUMN control word allows you to switch from multiple-column mode to single-column mode (for example, to create a table or figure the full width of the page), and the ".mc" MULTIPLE COLUMN con-

trol word will switch you back to the previously-defined multiple-column mode. For more complicated effects you can use the ".cd" control word instead. For example, you can go from two columns to three columns and back to two columns again.

If ".bc on" is in effect, then any time you switch from a multiple-column mode to a different number of columns (by using either ".sc" or ".cd"), SCRIPT attempts to balance columns before switching to the new column mode.

11. USING SYSTEM REFERENCE VARIABLES

System Reference Variables are maintained by SCRIPT to make it possible to get at information that may be useful in formatting a document. For example, the SCRIPT ".ur" USE REFERENCE control word

```
.ur &sysdate
```

results in the characters

```
July 3, 2012
```

which is the date on which this document was printed. This is extremely useful in preparing letters, because it can be used in conjunction with ".ra" (RIGHT ADJUST), as in

```
.ur .ra &sysdate
```

to produce the following effect:

July 3, 2012

12. CREATING YOUR OWN SCRIPT CONTROL WORDS USING MACROS

A MACRO is a "named" sequence of SCRIPT control words and text delimited with ".dm" DEFINE MACRO control words. SCRIPT saves these input lines and invokes them automatically whenever the name of the MACRO is used as a SCRIPT control word. For example,

```
.dm para begin
.sp a;.il +8
.dm para end
```

defines a MACRO named "para" which you can invoke by specifying

```
.para
```

and, as a result, one blank line will be generated and the next line of text will be indented 8 positions to the right of the current indent value.

13. THE CONTROL WORD INDICATOR AND SEPARATOR

13.1 IGNORING THE CONTROL WORD INDICATOR

There will be instances in which it is necessary to force SCRIPT to treat an input line as text even though it starts with the control word indicator (.) mentioned on page 1. This can be done for a specific number of input lines by using the SCRIPT ".li" LITERAL control word. For example,

```
.li 2
... This line of text starts with a "."
... and so does this one
```

causes SCRIPT to take the next 2 input lines as text even though they start with the control word indicator.

13.2 CHANGING THE CONTROL WORD INDICATOR

If you have a large number of input text lines which start with the control word indicator, and you don't want to have to bother counting them and specifying the ".li" control word as shown in the previous section, then you can change the control word indicator to some other character. For example,

```
.li ?
.These are lines of text which may or may not
start with "."s, but will be treated as text anyway
.       ...
.       ...
?li .
```

causes SCRIPT to change the control word indicator from a "." to a "?", and then back to a "." later on.

13.3 DEFINING A "NO-BREAK" CONTROL WORD INDICATOR

Many of the SCRIPT control words described in this Introduction cause a "break" (see Section 3.10). There are some instances in which this is not desirable. You can define a "no-break" control word indicator by using the ".li" control word. For example,

```
.li , nobreak
```

defines the "," as the "no-break" control word indicator. This is very useful whenever you want to use a control word which would normally create a break without having it do so, as shown below:

```
The following lines of text will be formatted without
creating any breaks, and, at the same time, we will
indent the text 10 spaces, starting with the line
in which the word "here"
,in 10
appears.  If ".in" were used instead, the line
containing this word would be printed, no matter
how full it was, and the indent would start with
the next line after it.
.in
```

produces:

The following lines of text will be formatted without creating any breaks, and, at the same time, we will indent the text 10 spaces, starting with the line in which the word "here" appears. If ".in" were used instead, the line containing this word would be printed, no matter how full it was, and the indent would start with the next line after it.

If you want to remove the "no-break" indicator, so that it is no longer in effect, then specify

```
.li * nobreak
```

13.4 THE CONTROL WORD SEPARATOR

If you ever find it necessary to change the control word separator character, the SCRIPT ".cw" CONTROL WORD SEPARATOR control word will do it for you. For example,

```
.cw <
```

causes the control word separator character (the semicolon, ";") to be replaced by the character "<"; you can restore it to the semicolon later on by specifying

```
.cw ;
```

14. USING SCRIPT

14.1 VM/370 CMS

In CMS, you may use either EDIT or WYLBUR to create a file of type "script", enter the input records, and save the file. To SCRIPT the file, specify the CMS command

```
SCript filename (options
```

where "filename" is the name of your input file. A list of the options available may be obtained by issuing the CMS command

```
help script
```

(or "$help script" from within EDIT, or "@help script" from within WYLBUR).

It is also possible to use the "help" command to get reference information on a particular SCRIPT control word, by specifying the control word instead of "script" (for example, "help .in" will tell you about INDENT).

14.2 THE BATCH SERVICE

If you are using WITS or WYLBUR to create your file, then the following Job Control Language (JCL) should be included:

```
//jobname JOB '#,F=5200,accounting options'
//  EXEC  SCRIPT
//SCRIPT.SYSIN  DD  *
input file
/*
```

The "accounting options" are described in the "BATCH Job Card Options" document available from any of the Consulting Offices. If you wish to specify any SCRIPT options, then change the EXEC card to

```
//  EXEC SCRIPT,SCRIPT='options'
```

14.3 THE DEBUG SERVICE

SCRIPT may be accessed through the DEBUG Service reader/printer stations or the WIDJET system as follows:

```
$ACC acct# name
$JOB  SCRIPT
input file
```

Unfortunately, keypunches do not have lower-case characters.

14.4 THE UTILITY SERVICE

If you are using WITS or WYLBUR, you can submit a SCRIPT job to the DEBUG Service, but the time- and page-limits are restrictive. Instead, it is more appropriate to submit a SCRIPT job to the CLASS=Y UTILITY Service, using the following control cards:

```
//jobname JOB '#,F=5200,accounting options',CLASS=Y
$JOB  SCRIPT
input file
/*
```

14.5 SOME VERY HELPFUL SCRIPT OPTIONS

There are a great many options you can specify when invoking SCRIPT, but most of them are beyond the scope of this Introduction. The "MARK" and "NUM" options below will make it considerably easier for you to relate the output produced by SCRIPT to the corresponding lines in your input file, and hence make it easier for you to edit your input file.

EIGHT — SCRIPT assumes that the output is being produced at SIX lines to the vertical inch on a page which is 11 inches long. Specifying the EIGHT option causes SCRIPT to count lines at eight per inch instead of six to decide when the 11-inch page is "full".

MARK — The "MARK" option causes SCRIPT to underscore the first character of each input text line in the output. This makes it easier to determine exactly which input line you are looking at, in conjunction with the "NUM=" option.

NUM=n — The "NUM=" option causes SCRIPT to print the input line numbers and the name of the input file from which they came, starting in output column "n".

For example, "NUM=85" is convenient, because it puts this information off to the right of the text area. Take care that the value you specify for "nn" is greater than the sum of the ADJUST and LINE LENGTH values for your document.

PA=m:n — Causes SCRIPT to print only pages "m" through "n". If ":n" is omitted, only pages from "m" to the end of the document are printed.

PL=n — If your output is not being produced on paper that is 11 inches long, you will need to tell SCRIPT how many lines there are on the page. For example, if your output page is 12.5 inches long, then specify "PL=75" for 6 lines to the inch, or "PL=100" for 8 lines to the inch.

PR
DISK
SCREen — These options are only useful in CMS, where the output is sent to your terminal by default (in all other systems mentioned above, output is always sent to the high-speed line printer). The "PR" option causes SCRIPT to send the output directly to the high-speed printer. The "DISK" option causes SCRIPT to put the output on your disk, in a file with the same name as the input file, but with a file type of LISTING. The "SCREen" option is useful for visual-display devices which do not support "backspace and underscore" output.

STOP — If your output is being produced at a hard-copy terminal, this option will cause SCRIPT to stop at the bottom of each page so that you can put in the next sheet of paper.

example

```
SPOOL PRINTER FORM 5200
SC filename (PR NUM=85 MARK
```

15. WHERE TO GO FROM HERE?

Before you embark on using SCRIPT for a document of any great complexity, it is recommended that you try SCRIPT on a few small documents, to build up some experience.

If you need help with SCRIPT, contact one of the User Services Consultants, in any of the Consulting Offices.

15.1 SCRIPT REFERENCE CARD

A copy of the most current SCRIPT Reference Card may be obtained from any of the Consulting Offices.

15.2 SYSPUB

SYSPUB is a set of MACROs which simplify the task of preparing and formatting documents which fall into the category of papers, theses, books, and other research publications.

SYSPUB allows you to prepare your publication using a set of commands which perform all of the numbering of chapters, sections, subsections, tables, figures, and footnotes automatically. These commands also format the text to conform to a widely-accepted set of layout standards (including those required for theses at this university). SYSPUB will also generate a Table of Contents for your publication.

SYSPUB allows you to prepare your document without you having to learn any more about SCRIPT than is presented in this Introduction (in fact, with SYSPUB, you don't need most of what is in this Introduction, but it's a good idea to understand it anyway). This Introduction was prepared using the SYSPUB commands.

A copy of the SYSPUB User's Guide many be obtained from any of the Consulting Offices.

15.3 OBTAINING A SCRIPT REFERENCE MANUAL

The material presented in this Introduction is the basis for solving most of the common formatting problems. When you need more information on SCRIPT, then submit the following job:

```
//jobname JOB  '#,TIME=2,PAGES=300,F=5200'
//     EXEC     SCRIPTMN
```

15.4 UPDATING YOUR SCRIPT REFERENCE MANUAL

SCRIPT is under on-going development at the University of Waterloo. Updates to your SCRIPT Reference Manual may be produced by submitting the following job:

```
//jobname JOB '#,T=1,P=200,F=5200'
//     EXEC     SCRIPTMN,SCRIPT='+UPDATE=yymmdd'
```

where "yymmdd" is replaced by the date on which your SCRIPT Reference Manual was printed (see the title page in the SCRIPT Reference Manual). Each time you submit this job, you obtain a new title page, and replacement pages for your Reference Manual for any component which has been changed since the date specified by "yymmdd".

You can submit either of the above jobs through WITS, WYLBUR, or CMS. If you are submitting a job through CMS, then type "help submit" for more information before you attempt to submit the job. If you are submitting a job on cards at a BATCH Service card-reader, then you must replace the "#" with your account number and lock.

Index of SCRIPT Control Words and Options

ad (ADJUST) ... 10

bc (BALANCE COLUMNS) ... 22
bm (BOTTOM MARGIN) ... 10
br (BREAK) ... 7
bt (BOTTOM TITLE) ... 20
bx (BOX) ... 17

cb (COLUMN BEGIN) ... 22
cc (CONDITIONAL COLUMN) ... 22
cd (COLUMN DEFINITION) ... 21
ce (CENTRE) ... 13
cl (COLUMN LENGTH) ... 21
co (CONCATENATE) ... 12
cp (CONDITIONAL PAGE) ... 4
cw (CONTROL WORD SEPARATOR) ... 2, 25

dm (DEFINE MACRO) ... 23
ds (DOUBLE SPACING) ... 2

em (EMPTY PAGES) ... 6

fk (FLOATING KEEP) ... 5
fm (FOOTING MARGIN) ... 20
fn (FOOTNOTE) ... 18
   set ... 19
fo (FORMAT) ... 11, 13
fs (FOOTING SPACE) ... 20

hm (HEADING MARGIN) ... 19
hs (HEADING SPACE) ... 19
hy (HYPHENATE) ... 12
   imbedding the exception dictionary ... 12

il (INDENT LINE) ... 15
im (IMBED) ... 12
in (INDENT) ... 14
   without creating a break ... 25

ju (JUSTIFY) ... 13

le (LEADING SPACES) ... 6
li (LITERAL) ... 1, 24, 25
ll (LINE LENGTH) ... 2

mc (MULTIPLE COLUMN) ... 23

of (OFFSET) ... 15
oo (OUTPUT OVERLAY) ... 16
options
   DISK ... 28
   EIGHT ... 27
   MARK ... 27
   NUM= ... 27
   PA= (PAGE) ... 28
   PL= (PAGE LENGTH) ... 28
   PR ... 28
   SCREEN ... 28
   STOP ... 28

pa (PAGE) ... 5
   see also "em" ... 5
pn (PAGE NUMBERING) ... 21
pp (PARAGRAPH START) ... 3

ra (RIGHT ADJUST) ... 14, 23

sc (SINGLE COLUMN) ... 22
sp (SPACE) ... 6
   absolute ... 6
   at the top of a page ... see also "le"
ss (SINGLE SPACING) ... 2

tb (TAB) ... 7, 8
   set ... 8
tm (TOP MARGIN) ... 10
tr (TRANSLATE ON OUTPUT) ... 17
tt (TOP TITLE) ... 20

uc (UNDERSCORE AND CAPITALIZE) ... 3
ud (UNDERSCORE DEFININTION) ... 3
up (UPPERCASE) ... 3
ur (USE REFERENCE) ... 23

us (UNDERSCORE) ... 3 wd (WIDOW) ... 18

TABLE OF CONTENTS

Abstract i

page

1. INTRODUCTION 1

2. CREATING THE INPUT 1

   Control Words and Text 1
   Putting More Than One Control Word on the Same Input Line 2

3. THE FORMATTING PROCESS 2

   The Line Length 2
   Single- and Double-Spacing 2
   Starting a New Paragraph 3
   Underscoring and Capitalizing 3
   Keeping a Block of Text Together on the Same Page 4
   Starting a New Page 5
   Blank Lines (Spaces) 6
   Blank Lines at the Top of a Page 6
   Printing Empty Pages 6
   Breaks 7

4. THE DEFINITION AND USE OF TABS 7

   Defining Tab-Stop Positions 7
   Entering TAB Characters in the Input 8
   Tabs with a "Fill String" 8
   Right-Aligning Tabs 8
   Centering Tabs 9
   Character-Aligning Tabs 9

5. THE OUTPUT PAGE 10

   The Left and Right Margins 10
   The Top and Bottom Margins 10
   The Text Area 11

6. THE FORMATTING MODES 11

   Unformatted Mode 11
   Concatenated Mode 12

Hyphenation 12
Concatenated-and-Justified (Formatted) Mode . . . 13

7. POSITIONING BLOCKS OF TEXT IN THE TEXT AREA 13

Centering Lines of Text 13
Right-Adjusting Lines of Text 14
Indenting Blocks of Text 14
Indenting a Single Line of Text 15
Offsetting Lines of Text 15
Output Overlay Patterns 16
The "Significant Blank": Keeping Things on the Same Line 17
Putting a Box Around Text 17
Widowing 18

8. FOOTNOTES 18

9. PAGE NUMBERING: TOP AND BOTTOM TITLES 19

Title Lines in the Top Margin 19
Title Lines in the Bottom Margin 20
Page Numbering 20

10. MULTIPLE-COLUMN OUTPUT 21

Defining the Page Layout 21
Starting a New Column 22
Balancing Columns 22
Switching Between Single and Multiple Columns . . 22

11. USING SYSTEM REFERENCE VARIABLES 23

12. CREATING YOUR OWN SCRIPT CONTROL WORDS USING MACROS 23

13. THE CONTROL WORD INDICATOR AND SEPARATOR 24

Ignoring the Control Word Indicator 24
Changing the Control Word Indicator 24
Defining a "No-Break" Control Word Indicator . . 25
The Control Word Separator 25

14. USING SCRIPT 26

VM/370 CMS 26
The BATCH Service 26
The DEBUG Service 27
The UTILITY Service 27
Some Very Helpful SCRIPT Options 27

15. WHERE TO GO FROM HERE? 28

SCRIPT Reference Card 29
SYSPUB 29

Obtaining a SCRIPT Reference Manual 29
Updating Your SCRIPT Reference Manual 29

Index of SCRIPT Control Words and Options 31