ABSTRACT

The University of Waterloo version of SCRIPT (referred to as Waterloo SCRIPT, or simply as SCRIPT) is a powerful text formatting program. A general introduction to SCRIPT is given in the "Introduction to SCRIPT" document available from any of the Consulting Offices. It is assumed that you are familiar with and experienced at using the SCRIPT control words presented in that document.

This "SCRIPT Techniques" document illustrates some of the programming-language capabilities of SCRIPT which will enable you to automate the preparation and maintenance of documentation.

There are two good reasons for automating the production of documentation:

1. to eliminate the "unnecessary re-work" impact of making changes, so that you can add a chapter, delete a section, interchange large sections of text, etc., without affecting the rest of the document;

2. to make it easier for someone else to learn how to use the computer to prepare documenation, and also reduce the amount of "human time" needed to produce a document.

A number of the techniques presented in the following pages can be used to create a set of "higher-level" formatting commands so that repetitive tasks can be automated, standardized, and simplified.

This document is continually being modified and extended to include additional techniques of potential wide-spread interest.

1. IMBEDDING OTHER INPUT FILES

With any large document, it is to your advantage to create a separate input file (collection of input lines) for each of its constituent parts, and then construct a small "driver" file to imbed each file, in the desired order. In this fashion, small sections of text may be developed independently and may easily be interchanged by changing the order in which they are imbedded.

You can instruct SCRIPT to imbed another input file by using the ".im" IMBED control word. For example, if an input file consisted only of

```
.im chap1;.im chap2;.im chap3
```

the three "chapters" could be maintained as three separate files. Moreover, the first file, "chap1", could consist of

```
.im sect1;.im sect2
```

which would, in turn, cause two more files to be imbedded during the formatting process.

2. SCRIPT AS A PROGRAMMING LANGUAGE

2.1 SYSTEM REFERENCE VARIABLES

SCRIPT maintains a large number of "status indicators" and "system information values" in special variables called "System Reference Variables". In the examples listed below, the name of the System Reference Variable is shown on the right, followed by a brief explanation, followed in parentheses by it's value at the time this document was formatted.

SYSDATE contains today's date (July 3, 2012)

SYSLINE the current line number on the page (49)

SYSLL the current LINE LENGTH (60)

SYSPAGE contains the current page number, as a number (1)

SYSPPAGE contains the current page number, as a character string (1)
(if we were on page 2 and page numbers were being printed as lower-case roman numerals, SYSPPAGE would have the value "ii")

SYSOUT a "TERM" if output is on-line, otherwise "PRINT" (PRINT)

System Reference Variables are used in conjunction with the ".ur" control word, as shown in the following example which prints today's date.

```
.ur Today is &sysdate..
```

The first period after "&sysdate" is treated as a "concatenation indicator" by SCRIPT. Thus, "&sysdate.." means "today's date, concatenated with a period", and the result is:

```
Today is July 3, 2012.
```

2.2 USER-DEFINED REFERENCE VARIABLES

You can define Reference Variables of your own (just don't pick names which start with the letters "sys") and assign values to them by using the ".sr" control word.

```
.sr name = numeric integer value
.sr name = string
.sr name = 'a delimited string'
```

The name of a Reference Variable may contain up to ten alphanumeric characters. A Reference Variable may optionally be subscripted from (-32767) to (+32767). The value assigned to the Reference Variable may be an integer (such as 3), an undelimited string (such as the word dog), or a delimited string (such as 'hello there').

If you wish to use other Reference Variables on the left- or right-hand side of the "=", then you must use the alternate form of SET REFERENCE, ".se", which does symbolic substitution as many times as is needed to eliminate Reference Variables from both the "name" and "value" fields. For example,

```
.se i = &i + 1
```

adds 1 to the previous value of Reference Variable "i". If "i" has not previously been defined, the above is equivalent to

```
.se i =  + 1
```

since the "&i" results in the null string.

If Reference Variable "dog" contained the string "cat", then

```
.se &dog = 1
```

would create a Reference Variable "cat" with the value "1".

2.3 AUTOMATIC SUBSTITUTION OF REFERENCE VARIABLES

Whenever there are a large number of lines containing Reference Variables, it is often easier to use the SUBSTITUTE REFERENCE control word ".su" to cause symbolic substitution, rather than numerous occurrences of ".ur". For example, the following

```
.su on
.sr fignum = &fignum + 1
Figure &fignum ....... &sysppage
.su off
```

produces:

```
Figure 2 ....... 3
```

2.4 THE "PERFORM" CONTROL WORD

The SCRIPT ".pe" control word is used in one of the two forms

```
.pe n;operand line          .pe n
                      (or)  operand line
```

It causes the "operand line", which may contain both SCRIPT control words and text, to be "performed" the specified "n" times. For example, it is sometimes desirable to construct a dashed line which will stretch across the full width of the page (that is, has the same number of "-"s as the LINE LENGTH value). This can be done as follows:

```
.sr dash = ''
.ur .pe &sysll
.se dash = '-&dash'
```

The first line sets "dash" to the null string. The second line results in a "perform counter" equal to the ".ll" value. The third line adds another "-" to the value of "dash". The Reference Variable "dash" can then be used to create a dashed line across the page, as follows:

```
.br;.ur &dash
```

which results in

2.5 TESTING VARIOUS CONDITIONS: THE "IF" CONTROL WORD

The full power of Reference Variables is realized in conjunction with the ability to test the value of a Reference Variable, and then conditionally do something. For example, the input line

```
.ur .if &syspage ne 1;.pa
```

says "if the value of the current page number is not equal to 1, then eject to the top of the next page". The ".if" control word has two forms:

```
.if <condition>;<operand>
```

and

```
.if <condition>
.th <operand1>
.el <operand2>
```

In both of the above forms, the "operand" may be either a single SCRIPT control word, or it may be a text line.

2.5.1 "DO" Groups

The full power of the ".if" control word is realized when it is used in conjunction with the ".do" and ".do end" control words.

```
.if <condition>;.th .do
<operand1>
.do end;.el .do
<operand2>
.do end
```

The operands can now be any number of SCRIPT control words and/or text lines.

2.6 LENGTH, TYPE, AND SUBSTRING OPERATIONS ON REFERENCE VARIABLES

SCRIPT provides you with the ability to determine the number of characters in a Reference Variable or whether a

Reference Variable contains an integer or a character string, and to take a substring of the contents of a Reference Variable, as shown below:

L'&refvar	the value of this expression is the number of characters in the variable "refvar"
T'&refvar	the value of this expression is the character "N" if "refvar" has an integer value, or "C" otherwise
&refvar(m:n)	the value of this expression is a string consisting of all the characters from positions "m" through "n" inclusive in the value of "refvar"

For example, let us assume that at the start of each major section of a reference manual, the section number is contained in the Reference Variable "number" and the section title in the Reference Variable "title". At the start of each section, we wish to eject to a new page, reset the page number to "1", set the page number prefix to the first 5 characters of the section title (in upper case) and followed by a dash, add one to the section number, and print the section number and title such that the title is underscored but the number is not. If the title takes more than one line to print, we want the second and subsequent lines offset ten spaces more than the length of the section number. The following sequence of SCRIPT control words accomplishes that:

```
.pa 1
.ur .up .pn prefix &title(1:5).-
.se number = &number + 1
.ur .of 10+L'&number
.ud set │
.ur .us │&number   │&title
.of;.sp
```

3. WRITING YOUR OWN HIGHER-LEVEL FORMATTING COMMANDS

3.1 MACROS

A MACRO is a "named" sequence of SCRIPT control words and text delimited with ".dm" control words. SCRIPT saves these input lines and invokes them automatically whenever the name of the MACRO is used as a SCRIPT control word. MACROs are of the form:

```
.dm name begin
text and/or additional SCRIPT control words
.dm name end
```

and are invoked by specifying:

```
.name operands
```

The operands can be a mixture of two forms: positional and keyword. When the MACRO is invoked, SCRIPT processes the keyword operands as though they were of the form

```
.sr keyword=value
```

and assigns the values of the positional operands to the Reference Variables "1", "2", "3", ..., and assigns the number of positional operands to the Reference Variable "0", and then executes the body of the MACRO.

For example, in Section 2.6 on page 5, we illustrated a sequence of SCRIPT control words which were to be executed at the start of each section of a reference manual. By using MACROs, we could write a higher-level formatting command, called "section", to do the same thing:

```
.dm section begin
.pa 1
.ur .up .pn prefix &1(1:5).-
.se number = &number + 1
.ur .of 10+L'&number
.ud set |
.ur .us |&number   |&1
.of;.sp
.dm section end
```

Now, all that is necessary to start a new section is to use this "section" MACRO just like a SCRIPT control word:

```
.section 'title of section'
```

SCRIPT will assign the positional operand "title of section" to Reference Variable "1", and execute the body of the MACRO.

To give a simpler example, we can write a "paragraph" MACRO and use it to start each paragraph of text in our document:

```
.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.

3.2 REMOTES

A REMOTE is a special sequence of SCRIPT control words and/or text delimited by ".rm" control words. SCRIPT saves these input lines and invokes them automatically when a particular event occurs. For example,

```
.rm 10
text and/or other SCRIPT control words
.rm
```

defines a REMOTE which will be invoked the next time SCRIPT attempts to output something on line number 10 (if we're already past line 10 on the current page, then it will be invoked at line 10 of the next page). If we wanted this sequence invoked for the next 5 occurrences of line 10 (that is, on the next 5 pages), then changing the first line to

```
.rm 10 5
```

would accomplish that. If we wanted this sequence invoked on all subsequent pages, then

```
.rm 10 save
```

would prevent the REMOTE from being deleted after being invoked a specific number of times.

If we wanted to invoke this REMOTE ourselves, whether we were on line 10 or not, we could do so by using the SCRIPT ".si" (SIGNAL REMOTE) control word:

```
.si 10
```

If we wanted to delete the REMOTE, to prevent it from being invoked again, we could do so by specifying:

```
.rm 10 delete
```

Rather than deleting a REMOTE, it may be desirable to "deactivate" it temporarily. This can be done as follows:

```
.rm 10 save
.ur &efcm
text and/or control words
.rm
```

Setting the Reference Variable "efcm" to the value ".cm" results in a "comment" line when the REMOTE is invoked, so that the "text and/or control words" which follow it will get executed as normal. When we want to "deactivate" the REMOTE, however, re-defining the Reference Variable "efcm" to have the value ".ef" (END FILE) will cause SCRIPT to terminate processing of the REMOTE when it executes the ".ef" control word. The REMOTE will still be invoked every time SCRIPT tries to print on line 10, but it won't do anything. You can re-define "efcm" back to ".cm" when you want to reactivate the REMOTE later.

4. FLOATING BLOCKS

SCRIPT provides several control words which cause text to be formatted into an "internal workarea" and then printed; they are of the form

```
.xx begin
text and/or additional control words
.xx end n
```

where ".xx" is one of the control words

```
.cc   (conditional column)
.cp   (conditional page)
.fk   (floating keep)
.fb   (floating block)
```

SCRIPT formats the text in this begin/end sequence into an internal workarea, and keeps a count of the number of lines of "output" which resulted. If an integer "n" has been specified after the "end" operand, SCRIPT adds the value of "n" to this count. SCRIPT uses the count in determining where the block of "output" lines should be printed, as follows:

.cc if the number of lines remaining in the text area in this column is less than the count of lines in the block, SCRIPT begins a new column and then prints the block

.cp if the number of lines remaining in the text area on this page is less than the count of lines in the block, SCRIPT begins a new page and then prints the block

.fk if the number of lines remaining in the text area in this column is less than the count of lines in the block, SCRIPT attempts to fill the remainder of this column by formatting the input which follows the block until the column is full, and then prints the block at the top of the next column

The ".fb" (FLOATING BLOCK) control word functions in a slightly different manner; the block doesn't print until you tell SCRIPT to print it, and this makes ".fb" a useful programming tool. SCRIPT provides two Reference Variables related to ".fb"s:

SYSFBC the count of the total number of output lines waiting to print, in all ".fb" blocks

SYSFBF the count of the number of output lines in the FIRST block waiting to print

To print a ".fb" block, you must specify the SCRIPT control word

```
.fb dump n
```

If used without specifying an "n", all existing ".fb" blocks will be printed; if you just want to print the first one, specify a value of "1" for "n"; if you want to print the first 3, then specify a value of "3".

5. CASE-STUDY EXAMPLES

5.1 AUTOMATING PAPERS FOR IEEE TWO-COLUMN PHOTO-OFFSET PUBLICATIONS

The style requirements and layout form for papers submitted to the Institute of Electrical and Electronics Engineers provide an excellent case-study example in the use of SCRIPT to automate the preparation of a document.

---

[1]IEEE Editorial Department, 345 East 47 Street, New York, N.Y. 10017

The layout grid provided by IEEE[1] is a 9.5"x13" area with two columns with a 1/4" gutter between them. On the first page, the top 2.5" of the page is to contain a full-page-width title area, and the bottom 2.5" of the left column is to be left blank for use by the IEEE. There is a two-line bottom margin; page numbers are centered on the second of these two lines. The style requirements dictate three levels of headings, which we shall refer to as "chapters", "sections", and "subsections". Chapter titles are to be centered, in capitals, and underscored. Section titles are to be flush with the left margin, upper- and lower-case, and underscored. Subsection titles are to be "paragraph" indented, upper- and lower-case, underscored, followed by a period, and run in with the text of the paragraph. The body of the paper is to begin with a short "ABSTRACT" in the same style as a chapter. For purposes of simplification, we are going to assume that the output is being produced at 6 lines to the vertical inch and 10 characters to the horizontal inch.

Our objective in this example is to write a file called "ieee" which the author will imbed at the start of the input file for the paper being produced. This "ieee" file will contain all the SCRIPT control words needed to set up the formatting environment so that the author's task in producing the paper can be simplified to the point of producing an input file as shown in Figure 1.

The author imbeds the "ieee" file, enters the text for the page-one title area without worrrying about the vertical or horizontal centering of that text in the title area, and then uses the commands we are going to write as MACROs to create the paper. To begin, we will write the SCRIPT control words which define the formatting environment for the paper, as shown in Figure 2.

Note that, after "ieee" is imbedded, we are in "centered" mode, and the input text lines are being formatted in the internal ".fb" workarea. Also, there is a REMOTE waiting to be invoked at line 61 (page length minus bottom margin minus 15) which will issue a ".cb" COLUMN BEGIN and cause the last 15 lines (2.5") of the first column to be left blank. The ".abstract" MACRO (see Figure 3), must terminate the ".fb", centre the floating-block lines at the top of the page, go into two-column mode, print the "ABSTRACT" title, and start formatting the text.

The ".sk c" produces a "skip conditional": a blank line which will only exist if it is not followed immediately by something which produces another blank line (otherwise, SCRIPT will throw the conditional blank line away). We can now write the MACROs for ".chapter", ".section", and ".subsection", as shown in Figure 4.

```
.im ieee
TITLE OF PAPER
.sp
Name of Author
Author's Business Affiliation
City, Province/State, Country
.abstract
.pp;text of abstract ...
.chapter 'title'
.pp;text ...
.section 'title'
.pp;.text ...
.subsection 'title'
text ...
   .
   .
.chapter 'References'
   .
   .
```

Figure 1: Input File Prepared by Author for IEEE Layout

The ".in"s issued by each of these MACROs are just a "safety precaution" in case the author left an INDENT or OFFSET outstanding -- it depends on how much you want to try to protect people from themselves.

```
.cm --- define page for terminal or printer
.hs 0;.hm 0;.tm 0;.bt //%//
.ur .if &sysout eq TERM
.th .do;.pl 78;.bm 2;.do end
.el .do;.pl 132;.bm 56;.do end
.cm --- define hyphenation environment
.hy on;.hy set thresh 4;.hy set minpt 2
.hy set endpt 3;.hy set sup 2
.cm --- define two-column environment
.ll 95;.cl 45;.cd 2 0 50
.cm --- define a remote to leave bottom 15 lines
.cm     of first column blank
.ur .rm &syspl-&sysbm-15 1
.cb
.rm
.cm --- go into single-column mode, collect title area
.sc;.ce yes;.fb begin
```

Figure 2: The "ieee" Formatting Environment

```
.dm abstract begin
.cm --- print the title area centered vertically
.fb end;.ur .sp (15-&sysfbc)/2;.fb dump
.cm --- go to line 26, print "ABSTRACT"
.ln 26;.mc;.ce 1;.uc abstract
.fo;.sk c
.dm abstract end
```

Figure 3: The Abstract MACRO

```
.dm chapter begin
.sp 2;.cc 6;.in;.ce 1;.ur .uc &1
.fo;.sk c
.dm chapter end
.dm section begin
.sp;.cc 4;.in;.co;.ju no;.hy sup
.ur .us &1
.ju;.sk c
.dm section end
.dm subsection begin
.sp;.cc 2;.in;.fo
.il 3;.ur .us &1..
.dm subsection end
```

Figure 4: The Chapter, Section, and Subsection MACROs

5.2 AUTOMATING FOOTNOTES

The SCRIPT ".fn" (FOOTNOTE) control word handles the placement of footnote text for you, but it does no more than that. In this example, we will write a set of MACROs based on the ".fn" control word to automate footnotes completely. We assume that footnotes are to be numbered consecutively starting at "1", and that if the document is being produced on the printer, a print train containing superscript numbers is being used, but if the document is being produced at an on-line terminal, then such superscripts are not available. The MACROs we are going to write will be called ".footnote" and ".footend", and we will use them in the same way that ".fn begin" and ".fn end" are used. In order to translate footnote numbers to superscripts, we need the control words shown in Figure 5.

The control words shown in Figure 5 define 10 Reference Variables called "super1" through "super0" whose values are the machine-code representations for the superscript digits 1 through 0. The MACROs are shown in Figure 6. They are used as follows:

```
text, up to word to be
.footnote referenced
text of footnote
.footend
followed by rest of sentence
```

and produce the following result:

```
.cm --- define the "_" to be the backspace character
.cm     and define the hex codes for the superscripts
.bs _
.sr super1 = 'B_1'
.sr super2 = 'B_2'
.sr super3 = 'B_3'
.sr super4 = 'B_4'
.sr super5 = 'B_5'
.sr super6 = 'B_6'
.sr super7 = 'B_7'
.sr super8 = 'B_8'
.sr super9 = 'B_9'
.sr super0 = 'B_0'
.bs
```

Figure 5: Translating Footnote Numbers to Superscripts

text, up to word to be referenced[2] followed by rest of sentence

The ".footnote" MACRO adds one to the footnote number, and if output is being produced to the terminal, creates a footnote symbol consisting of the footnote number in parentheses. If output is to the printer, it creates a footnote symbol by converting each of the digits in the footnote number to its corresponding superscript representation. The input line

```
.se x = '&x.&&&&super&&footnum(&i:&i)'
```

requires an explanation of SCRIPT's "substitution scan" algorithm. Each time SCRIPT encounters "&&", it replaces it by a single "&" and then continues scanning the input line. If the footnote number were "25" and SCRIPT were on the first time through the ".pe", then, after one pass of the substitution scan algorithm, the above line would have been reduced to

```
.se x = '&&super&footnum(1:1)'
```

and after the second pass, would have been reduced to

---

[2] text of footnote

```
.dm footnote begin
.cm --- add 1 to the previous footnote number
.se footnum = &footnum + 1
.cm --- are we online or offline?
.ur .if &sysout eq TERM;.th .se x = '(&footnum)'
.el .do
.sr x = '';.sr i = 0;.up .pe L'&footnum
.se i = &i + 1;.se x = '&x.&&&&super&&footnum(&i:&i)'
.do end
.cm --- output the word and the footnote number
.ur &1.&x
.fn begin
.cm --- if already some footnotes on this page,
.cm     leave blank line between footnotes
.ur .if &sysfnc gt 0 .sp
.cm --- indent and output the footnote symbol
.ur .in L'&x
.ur .oo 1 &x
.cm --- start collecting footnote text
.dm footnote end
.dm footend begin
.fn end
.dm footend end
```

Figure 6: The Footnote MACROs

```
.se x = '&super2'
```

which would give "x" the machine-code value "B2" on the next pass. On the second time through the ".pe", "x" would end up containing the machine-code value "B2B5", the superscript representation of the number "25".

5.3 AUTOMATING BOXES

In this example, we will write MACROs called ".boxon" and ".boxoff" to automate the placement of boxes around text (see Figure 7).

```
.dm boxon begin
.sk;.ur .bx &sysin+1 &syscl+(&sysinr)
.in +3 -3;.sk
.dm boxon end
.dm boxoff begin
.sk;.in -3 +3;.bx off;.sk
.dm boxoff end
```

Figure 7: The Boxon and Boxoff MACROs

INDEX OF SCRIPT CONTROL WORDS

TABLE OF CONTENTS