Applications Program Series
AP-41 Revision 1
November 15, 1981

MULTICS WORD PROCESSING USING RUNOFF

This two-part memo tells how to use runoff and other word-processing programs on Multics. Part 1 is a primer, intended to introduce a person who has little or no knowledge of computerized text-formatting to the basic material needed to produce documents with the computer. (If you have had no experience using a time-sharing system, you should first read Section I of IPS Memo MS-1 and Sections 1 through 4 of New Users' Introduction to Multics--Part I, cited below.) Part 2 is a complete reference description of the runoff text formatter.

This memo combines the previous AP-41 and AP-42 memos.

The following documents are referenced in this memo and are useful for further information.

VENDOR DOCUMENTATION

New Users' Introduction to Multics--Part I (CH24)
Emacs Text Editor Users' Guide (CH27)
qedx Text Editor Users' Guide (CG40)
Multics Programmers' Manual: Commands and Active Functions (AG92)
Multics WORDPRO Reference Guide (AZ98)

IPS MEMOS

IPS User's Guide
MS-1 Multics at IPS
AP-40 Word Processing at IPS (forthcoming)
AP-52 Author-Maintained Library: Application Software

AVAILABLE ON: Multics

CONTENTS

PART 1: RUNOFF PRIMER

I. INTRODUCTION

Word processing is the use of computer programs to help you produce memos, reports, letters, and other documents. On IPS's Multics system, the main word-processing program is runoff. Using runoff to turn out a document can be much easier than using a typewriter. This is because runoff automatically formats and types text. Repetitious typing is eliminated (such as typing the body of a form letter many times). You can correct errors or make changes without manually retyping large parts of the document. In addition, you need not concern yourself with the pagination of the document. Runoff automatically prints what it can on each line (and on each page), and continues on a new line (or new page) automatically. Unless you specify otherwise, runoff enforces top, bottom, and side margins, and smoothes out ("justifies") the right margin.

Although you do not have to carefully type the formatted pages that will make up the final version of your document, you do have to enter the input file from which runoff produces the final output. The input file contains text lines (the sentences of your document) and formatting directions. To create an input file for runoff, you use a text editor. On Multics, the two main text editors are qedx and Emacs. Either can be used to create, change, and store files in the computer's memory.

The process, then, of producing documents using runoff involves the following steps:

- using a text editor to enter text lines and formatting directions into an input file;
- issuing the runoff command to format the text of the input file.
- repeating the above two steps in sequence--i.e., using the text editor to change or correct the input file, and reissuing the runoff command to format the corrected or revised document. You repeat this process as many times as needed. Once you have created the input file, you enter only changes or additions into it. You do not retype parts of the document; runoff itself does the reformatting required to accommodate changes.

The part of the process that requires the most work and attention on your part is the placing of text lines and formatting

directions in the input file. The formatting directions you put in this file are called control lines. A control line begins with a control word, composed of a period (.) followed by two or three lowercase letters. For example, the control line used to center a line of text is ".ce".

You intersperse control lines throughout the text. (Each usually appears alone on a line of the input file just before the text it is to affect.) Control lines do not appear in the final document (i.e., the output produced by runoff).

Text lines contain material that will actually appear in the finished document. Text lines may be up to 256 characters in length. It is convenient, when using the text editor to create an input file for runoff, to end input-file text lines at the ends of sentences where possible. (This makes inserting new sentences easier.) You should not "measure" text lines, divide words, or worry about text "fitting" onto a page; runoff does all such formatting automatically. Normally a text line may not begin with a period.

Runoff control lines permit extremely sophisticated formatting. Equations can be formatted; space can be allowed for diagrams; footnotes can be automatically generated, numbered, and positioned. You can control indentation, pagination, and margins. If you wish, you can have formatted text printed page-by-page. (Runoff will wait to allow you to insert and position separate sheets of paper.) You can direct the output text to a file and have it printed on the high-speed printer. Runoff can produce right-justified text (like that you are reading) or a "ragged" right margin, and can be told to hyphenate words automatically. Many other features provide extensive control over the processing of text.

Part 1 of this memo introduces the simplest methods of using runoff, and refers (where possible) to the description of the runoff command in Part 2 for information on runoff's more sophisticated capabilities. Part 2 contains more detailed information about the use of (and restrictions on) control lines described here.

II. CREATING INPUT FILES FOR RUNOFF

Before you can use runoff, you must create an input file by using a text editor. The two main text editors on Multics are Emacs and qedx. Emacs is a screen editor. It can only be used with video terminals. Many people find it the most natural editor to use: you see portion of your file before you at all times; changes and additions are immediately reflected on the screen. Perhaps the only drawback of Emacs is that it is greedy in its use of system resources and therefore responds very sluggishly

when a lot of people are using Multics at the same time. Qedx, on the other hand, is a line editor. You type requests that operate on a line (or group of lines). To see the effect of any change you make to a line, you must explicitly ask qedx to print the line. Qedx can be used with any sort of terminal.

Although a text editor is a necessary adjunct to the use of runoff, we can only touch briefly on the two editors in this memo. To learn more about Emacs, read the Emacs Text Editor Users Guide (Honeywell document CH27). Qedx is described in the qedx Text Editor Users Guide (CG40).

There are, by the way, other editors you can use on Multics. To find out what they are and how they compare with Emacs and qedx, give the Multics command "help editing.gi".

LOGGING IN

Before you can use the word-processing software on Multics (or any of the software on Multics), you must "log in". See Section I of IPS Memo MS-1 and Section 2 of the New Users' Introduction to Multics--Part I for information on how to log into Multics.

CREATING FILES WITH EMACS

To begin creating a file with Emacs, type:

```
emacs
```

The command "emacs" invokes the Emacs text editor. Wait for the screen to clear. When it does, Emacs is ready for you to type in your input file. Type in text or control lines, followed by carriage returns.

After you've typed in your file, you have to tell Emacs to "write" your file, i.e., make a permanent copy of the file in the Multics storage system. To do this, you issue an Emacs request. However, since anything you type in the conventional manner becomes part of the file, there are special ways of entering Emacs requests. One way is hold down the key marked CONTROL (or CTRL) while you press another key. (The usual notation for this is a circumflex character "^", signifying the CONTROL key, followed by the character associated with the other key.) For instance, you issue ^X^W to write a file--that is you hold down the CONTROL key while you press the "X" key once, and then hold down CONTROL and press the "W" key.

When you issue ^X^W, Emacs prompts you for a name for the file. Type a name, followed by a carriage return; the name must end in ".runoff" so that you can use runoff to format the file.

After you've written the file, exitf from the editor. Type the ^X^C request. Multics prints a "ready" message. The computer is again "ready" to accept commands. This is called command level.

CREATING FILES WITH QEDX

To begin creating a file with qedx, type the lines:

```
qedx
a
```

The command "qedx" invokes the qedx text editor. Multics does not print a response. The request "a" tells qedx you want to append lines--i.e., to enter input mode.

The qedx editor has two modes: input and edit. When you add lines to a file (or when you first begin to create a file), you use input mode. At other times (e.g., when you change, print, delete, or manipulate lines, or when you save files or prepare to leave the editor), you use edit mode.

You may now enter a text or control line, followed by a carriage return. You may then enter another text or control line, and continue in this way until you have entered all of the lines of your document--or enough so that you feel it's time to rest. When you want to stop, type "backslash f" (\f), to tell qedx you are "\finished" entering lines. The "\f" tells qedx to leave input mode and enter edit mode. For example:

```
qedx
a
These are a few lines we are typing into a file.
Normally we might make some mistakes,
but we'll assume we're perfect for now,
and get into that later.
\f
```

Now type a "w" (write) request to qedx, to tell the computer to store the document in a file. Give the file a name on the "w" line; the name must end in ".runoff" so that you can use the runoff command to format the document:

```
w littlewords.runoff
```

If you have not finished entering the document, give another "a" (append) request to reenter input mode and begin entering lines again. If you have finished entering the document, type a "q" (quit) request, to tell the computer you are done editing and want to leave the editor. Multics prints a "ready" message. The computer is again "ready" to accept commands. This is called command level.

```
q
r 1651.4 0.053  2.704
```

Typing the "w" (write) request tells Multics to save the work you have done in the computer's storage. Always type a "w" just before "quitting" from the editor; if you do not do this you will lose the work you have done since you last typed "w". When creating or editing long documents, it is a good idea to "write" often--say, every 10 minutes--to protect additions or corrections that you make from being lost in case you forget the final "write" or the system should crash unexpectedly. After you have done your first "write" on a file, you can (and should) "write" subsequent times by typing just the "w" (in edit mode) on a line by itself. (Do not retype the file name.) For example:

```
qedx
a
  <---(Line of text)
  <---(Line of text)
  <---(Line of text)
\f
w junk.runoff
a
  <---(Line of text)
  <---(Line of text)
  <---(etc.)
\f
w
a
  <---(Still more text)
  <---(etc.)
\f
w
  <---(And so on.)
```

III. INVOKING RUNOFF

To format a document, you invoke runoff by typing the command's name (runoff), followed by the name of the file to be formatted. If, for example, you gave the name "goodgrief.runoff" to the input file when you created it using Emacs or qedx, you could type:

```
runoff goodgrief
```

Note that you can omit the file's ".runoff" suffix; runoff searches for a file with the suffix ".runoff".

Runoff formats the text of the input file according to the directions (control lines) in the input file, and prints each output line at the terminal. If you have given runoff a control line it cannot understand, an error message about it is usually printed after the document is finished.

There are a number of options you can give along with the name of the file when you give your runoff command. These are called control arguments. They are optional, and are used to modify the way runoff formats the file. A few runoff control arguments are discussed in Part 1 of this memo (see page); complete information is given in Part 2. While you are first learning to use runoff, do not worry about using command options.

IV. AN ANNOTATED EXAMPLE

The previous two sections have introduced the two basic steps involved in word processing using runoff: you use an editor to prepare an input file and then you give the runoff command to generate formatted output. This section comprises two sample terminal sessions (one featuring Emacs, the other qedx) that show how you can use a text editor and runoff together to produce the following short memo:

Notice:

To: Mr. Elmer Shorthair
Date: May 1, 1982

As of today I quit your beastly job. Working for you has been pure torture. I wish you'd go back to Acme Skunk Farm or wherever you came from.

Best wishes,
Dinky Doppler

EXAMPLE OF USING EMACS AND RUNOFF

After you log in, enter the Emacs editor:

```
emacs
```

When the screen clears, begin entering runoff control lines and the text of the memo. For text line, put as many words on a line as you want; runoff will rearrange them evenly later.

```
.ce
Notice:
.sp
To:    Mr. Elmer Shorthair
.br
Date:  May 1, 1982
.sp
As of today I quit your beastly job.
Working for
you has been pure torture.
I wish you'd go back to Acme Skunk Farm or wherever you
came from.
.sp
.in 40
Love,
.br
Dinky Doppler
```

At this point, you save what you've just typed by giving the Emacs "write-file" request (^X^W). When Emacs prompts you for a name at the bottom of the screen, you type:

```
emancipation.runoff
```

Before you leave the editor, you decide some of the language is a little too fawning, so you make some changes. You move the cursor, which was positioned at the end of the memo, up two lines by entering ^P twice, and then move to the beginning of the line reading "Love," by entering ^A. You delete everything on the line with ^K, and then insert your correction by typing:

```
Best wishes,
```

Since you want the permanent, "saved" version of the file to reflect the changes you just made, you save the file again, but this time, to prevent Emacs from prompting you again for the name you've already assigned, you use the "save-same-file" request (^X^S). Then you enter ^X^C to leave the editor. Multics prints a ready message when you're back at Multics command level.

Now you give the "runoff" command:

runoff emancipation

The formatted version of your memo appears at your terminal:

<---runoff provides a top margin of six lines

Notice:

To: Mr. Elmer Shorthair
Date: May 1, 1982

As of today I quit your beastly job. Working for you has been pure torture. I wish you'd go back to Acme Skunk Farm or wherever you came from.

Best wishes,
Dinky Doppler

<---runoff prints blank lines to fill out a 66-line page

r 15:39 1.134 933

Of course, as an Emacs user, you've just watched the final product scroll across your video screen. Probably you want the formatted memo on paper. One way to do this is to log in on a printing terminal and give the runoff command. Alternatively, you could have the formatted output printed on one of IPS's high-speed printers (see page).

EXAMPLE OF USING QEDX AND RUNOFF

In the example that follows, lines that you type are preceded by an arrow (=>). Do not type the arrow.

After you log in, invoke qedx, type "a" to enter input mode, and type in runoff control lines and the text of the memo. For text lines, put as may words on a line as you want; runoff will rearrange them evenly later.

```
=> qedx
=> a
=> .ce
=> Notice:
=> .sp
=> To:    Mr. Elmer Shorthair
=> .br
=> Date:  May 1, 1982
=> .sp
=> As of today I quit your beastly job.
=> Working for
=> you has been pure torture.
=> I wish you'd go back to Acme Skunk Farm or wherever you
```

```
=> came from.
=> .sp
=> .in 40
=> Love,
=> .br
=> Dinky Doppler
```

Now return to edit mode and write the file into a segment. Specify "emancipation.runoff" as the filename.

```
=> \f
=> w emancipation.runoff
```

Now you can print all or part of the file and make changes. To print the all the lines, type "1,$p". Let's say that after printing the file and reading through it, you decide that you don't like the closing, so you locate the offensive line, use the qedx request "s" (substitute) to alter the line, and use the "p" request (print) to verify the change:

```
=> /Love/
   Love,
=> s/Love/Best wishes/p
   Best wishes,
```

Now you write the file again. You omit the name of the file because you want qedx to use the same name you specified last time. Then you give the "q" request to leave the editor. Multics returns you to command level and prints a ready message.

```
=> w
=> q
   r 1:34 2.21 102
```

Now you give the "runoff" command:

```
=> runoff emancipation
```

But before you press the carriage return after the name of the file, you align the paper in your terminal at the top of a page. Then press return, and watch the formatted memo print at your terminal:

<---By default, runoff provides a 6-line top margin

Notice:

To: Mr. Elmer Shorthair
Date: May 1, 1982

As of today I quit your beastly job. Working for you has been pure torture. I wish you'd go back to Acme Skunk Farm or wherever you came from.

Best wishes,
Dinky Doppler

<---Enough blank lines to fill out a 66-line page

r 15:39 1.33 111

SUMMARY OF CONTROL LINES USED IN THIS SECTION

CONTROL LINE	MEANING TO RUNOFF
.br	(break) causes the preceding text line to be printed without filling in words from the next text line.
.ce N	(center) centers the next N lines. If N is not given, ".ce 1" is assumed. A .ce control line also acts as a break. (See the description of the .br control line.)
.sp N	(space) prints N blank lines. If N is not given, ".sp 1" is assumed. You can also input a blank line (i.e., a line consisting only of a carriage return) instead of a .sp control line. A .sp control line causes a break.
.in N	(indent) indents all following text lines N spaces until the next ".in N" control line. If N is not given, ".in 0" is assumed--i.e., indentation returns to the left margin. A .in control line causes a break.

Part 2 of this memo lists the most commonly-used runoff control lines with a brief explanation of their functions.

****
** **
** Try the example on your terminal. Use either Emacs or **
** qedx. Enter the text and control lines; name and store **
** your file; correct andy errors; "write" your changes; **
** and leave the editor and return to command level. **
** Then, when you have a ready message, invoke runoff to **
** format your file. **
** **
** The remainder of this memo describes more sophisticated **
** runoff control lines and introduces some runoff command **
** options. If you do not understand the example, reread **
** Sections 2 through 5 of the New Users' Introduction to **
** Multics--Part I (CH24), and reread this memo up to **
** here. If you are still confused, do not continue; **
** contact an IPS consultant for help. **
** **
****

V. FILLING

Unless you tell it otherwise, runoff formats text by moving the words of the input file from line to line to even the right margin, and pads the line with blanks to justify it exactly. In fill mode, runoff attempts to completely fill the current output line by inserting words from the next input-file text line(s), and avoids overshooting the right margin by moving words from the current input-file text line onto the next output line. When the current output line will not hold the next word to be formatted, runoff justifies the output lines (inserts extra blanks between words) unless you have used .na to prevent it.

Sometimes you will not want runoff to perform "filling"--e.g., when formatting a table or a long list of one-line items. To turn off filling, use the .nf (no-fill) control line before such a block of text.[1] To restore filling afterward, use a .fi (fill) control line. For example, the lines:

```
Shopping List---
.sp
.in 5
.nf
milk
bread
peanuts
junk food
broccoli
```

---

[1] NOTE: Turning off filling automatically turns off right-margin justification.

```
carrots
mangoes
paper towels
.in 0
.fi
```

produce output like this:

```
    Shopping List---

         milk
         bread
         peanuts
         junk food
         broccoli
         carrots
         mangoes
         paper towels
```

See also the description of runoff tabs on page and in Part 2.

VI. CONTROL LINES FOR PAGE LAYOUT

DOUBLE- AND SINGLE-SPACING

Unless you tell it otherwise, runoff prints text single-spaced. To double-space text, use the control line:

```
.ds
```

before the text to be double-spaced. (If the whole document is to double-spaced, put the .ds control line at the top of the input file.) Runoff will continue double-spacing the text until it finds a .ss (single space) control line.

BEGINNING OUTPUT ON A NEW PAGE

Runoff normally fills each page with text before going to the next page. If you want to continue printing on a new page, even if the current page is not full, use a .pa (page advance) control line to eject the current page.[2] This is useful in a document

---

[2] You can also control page ejection "conditionally", according to how much space is left on the current page. (See the description of the .ne control line in Part 2.) Or you can eject

that has several sections, each of which must begin on a new page. For example, the input-file lines:

```
.ce
Section I
.sp
The information in this document is
for internal use only.
.pa
.ce
Section II
      <---(text and control lines for Section II)
```

produces two page of runoff output--the first containing only the title "Section I" and one sentence, and the second containing the title "Section II" and the remaining text.

RIGHT-MARGIN JUSTIFICATION

Unless you tell it otherwise, runoff automatically right-justifies text lines by inserting extra blanks between words until the text is aligned at the right margin. (The text of this memo is right-justified.) A line printed just before a control line that causes a break (i.e., .br, .sp, .ce) is <u>not</u> right-justified.

To turn <u>off</u> justification, use the .na (<u>no</u> adjust) control line. (If the whole document is to be printed without right-margin justification, put the .na control line at the top of the input file.) Runoff still "fills" text lines, but does not pad lines with extra blanks to justify exactly. This produces a "ragged" right margin.

If you have turned off justification and want to restart it, use the .ad control line.

---

the current page and force the new page to be odd-numbered. (See the description of the .op control line in Part 2.)

VII. OTHER CONTROL LINES FOR FORMATTING TEXT

INDENTING

To indent text, use the runoff control line:

```
.in N
```

This control line indents all following lines N spaces from the left margin until a new ".in N" is encountered or until an indent-zero control line (".in 0", or simply ".in").

Consider the following example, part of an input file for a document in which a quotation is set off (indented five spaces) from the main body of the text:

```
Since the following material is a quotation,
it is indented five spaces.
.in 5
.sp
Four score and seven years ago, our forefathers
brought forth on this continent a new nation,
conceived in liberty...
.sp
.in 0
After the quote, the text should
resume at the left margin.
```

Runoff would format the text like this:

```
Since the following material is a quotation, it is
indented five spaces.

    Four score and seven years ago, our forefathers
    brought forth on this continent a new nation,
    conceived in liberty...

After the quote, the text should resume at the left
margin.
```

Because .in 0 (turning off indentation) acts as a break (preventing filling), the only way to indent the first line of a paragraph is to begin the text line with the required number of spaces. NOTE: A text line beginning with one or more spaces causes a break.

OFFSET OUTPUT LINES

Runoff has a special control line:

```
.un N
```

which can be used to create offset ("hanging-indent") or tabular output. The ".un N" control line causes the next output line, and only the next output line, to begin N spaces to the left of the current indentation.

For example, the lines:

```
.ce 2
THINGS TO BRING WITH YOU
WHEN YOU MOVE TO CAMBRIDGE
.sp
.in 6
.un 3
a) three rooms' worth of furniture for a
studio apartment
.sp
.un 3
b) enough police locks to repel
the entire population of Australia
.sp
.un 3
c) a small fortune to spend on
rent, groceries, and insurance
```

produce output like this:

THINGS TO BRING WITH YOU
WHEN YOU MOVE TO CAMBRIDGE

a) three rooms' worth of furniture for a studio apartment

b) enough police locks to repel the entire population of Australia

c) a small fortune to spend on rent, groceries, and insurance

TABS

Use the .htd (horizontal tab define) control line to create and name a pattern of tab stops--e.g.,

```
.htd  tableA  10 25 45
```

Turn on this tab pattern, and define a character you will input instead of a real tab (and which runoff will turn into a tab during formatting) with the .htn (horizontal tab on) control line:

```
.htn  tableA  $
```

Then enter tabs into the input file with this special character. For example, the control lines:

```
.nf
.htd  tableA  10 25 45
.htn  tableA  $
$NAME$RESIDENCE$AGE
.sp
$Joe Shmo$Boston$19
$Mary Nerd$New York$32
$Peter Pan$Neverneverland$90
.fi
.htf tableA
```

would produce the following output:

NAME	RESIDENCE	AGE
Joe Shmo	Boston	19
Mary Nerd	New York	32
Peter Pan	Neverneverland	90

As shown, you should precede lines used to produce a table with a .nf (no-fill) control line, to tell runoff not to move words from line to line to even the right margin. After the table, restore "filling" with a .fi control line and turn the tab pattern off with a .htf (horizontal tab off) control line, as shown in the example. Note that, since the actual tabstops are specified only in the .htd control line, you can experiment with the spacing of the table just by changing that one line (using qedx) and running off successive versions (using runoff). For more information on using tabs with runoff, see Part 2.

VIII. UNDERSCORING

If you are using qedx, enter text containing underscores in the same way you would on a typewriter. However, remember that when you use qedx to change underscored text, the underscores are considered part of the text (i.e., you must give both the text and the underscores in a "s" [substitute] or "//" [locate] request).

If you are using Emacs, use the "underline-word" request (press the key marked ESCAPE and then the "_" key). Since Emacs assumes your video terminal cannot show an "_" and the character it is "on top of", Emacs provides a visible indication of the backspace characters which are part of underscored words. The sequence "\010" represents a single backspace character. Thus, a word that would look like this printed on paper:

immature

would be displayed by Emacs as:

_\010i_\010m_\010m_\010a_\010t_\010u_\010r_\010e

IX. FOOTNOTES

Use the .ft control line to create footnotes in a document. When you come to a point in the text where a footnote is to appear, enter a .ft control line. Then enter the text of the footnote on the next line(s). End the footnote with another .ft line. That is, two .ft control lines delimit a footnote; .ft control lines must appear in the input file before and after the text of a footnote.

Runoff automatically saves room for each footnote and prints it at the bottom of the page.[3]

For example, the following lines:

```
This is an example of using footnotes.
Lincoln said that "four score and seven
years ago our forefathers brought forth upon
this continent a new nation...."
.ft
```

---

[3] If there is not enough room on the page for all the footnotes that are "waiting" to be printed, they are continued at the bottom of the next page. If a footnote would be generated on the last or next-to-last line of the page, runoff automatically ejects that page and begins a new one before printing the line that generates the footnote.

```
Abraham Lincoln, Gettysburg Address, 1863.
.ft
The next now resumes with no break.
```

produce output like this:

```
This is an example of using footnotes.  Lincoln said that
"four score and seven years ago, our forefathers  brought
forth  upon  this  continent  a new nation...." (1)  The
text now resumes with no break....


(1) Abraham Lincoln, Gettysburg Address, 1863.
```

The .ft control line does not create a break. Note that runoff numbers footnotes automatically; you do not enter a number yourself. You can change the format of the footnote reference symbol from the default (for instance, in this memo we use numbers in brackets instead of the usual parentheses.

X. HEADERS AND FOOTERS

Use the .he (header) or .fo (footer) control lines to put headers or footers in a document. The control line is specified with four apostrophes[4] which delimit the text to be printed left-adjusted, centered, and right-adjusted at the top (for headers) or bottom (for footers) of each page:[5]

```
.he 'left-text'center-text'right-text'
```

or

```
.fo 'left-text'center-text'right-text'
```

For example, to print the heading "XYZ Transistor Corporation" at the top left of each page, use the control line:

---

[4] If an apostrophe appears in the text to be included, you can use any other character not part of the text of the header/footer to delimit the three parts.

[5] You can also specify different headers or footers for odd- and even-numbered pages. For more information, see the descriptions of the .oh, .eh, .of, and .ef control lines in Part 2. In addition, you can specify multiple header or footer lines (.he 1, .he 2, etc.).

```
.he 'XYZ Transistor Corporation'''
```

We used the control line:

```
.he 'AP-41-1''Page %'
```

to create the header for each page of this memo. The "%" in a header/footer control line is replaced by the page number.

XI. GETTING SPECIAL EFFECTS WITH CONTROL ARGUMENTS

PRINTING DOCUMENTS ON TYPEWRITER PAPER

You can have runoff print a document on separate sheets of paper (instead of continuous-form terminal paper) by giving the "-stop" option when you issue the runoff command:

```
runoff  filename  -stop
```

This tells runoff to stop before each page (including the first) so that you can insert a single sheet of paper in the terminal. When the sheet is aligned, type a single carriage return; runoff will print one page of output and stop, and the process is repeated. Runoff will also stop after the last page; type one more carriage return to get a ready message.

RUNNING OFF PORTIONS OF DOCUMENTS

Sometimes you will want to see only part of a document. Give the page numbers with the "-from" or "-to" options in the runoff command. Examples:

```
runoff  filename  -from 5
runoff  filename  -from 16 -to 18
runoff  filename  -to 4
```

This is useful when you have to stop runoff (by pressing the BREAK key on your terminal) and restart the document in the middle--e.g., if your terminal ribbon runs out, you misalign your paper, or you discover a bad mistake and want to fix it before continuing.

HYPHENATING

To tell runoff to hyphenate a document, specify the "-hph" control argument when you give the runoff command:

```
runoff  filename  -hph
```

Hyphenation adds to the cost of using runoff, but may improve the appearance of the document.

Runoff hyphenates by looking up words in a computer "dictionary". You can create other dictionaries to override the standard hyphenation or add words specific to your project (which may not be in the standard dictionary). See page for more information.

NOTE: You can combine most of the runoff command options (called control arguments in Multics documentation) discussed above, depending on what you want runoff to do to your document. Examples:

```
runoff  filename  -hph  -segment
runoff  filename  -stop  -hph
runoff  filename  -hph  -to 17  -segment
```

Runoff has many other options, described fully in Part 2.

XII. PRODUCING A DOCUMENT ON THE OFF-LINE PRINTER

Using runoff to print a short document at the terminal is convenient, but if a document is very large, printing it at the terminal can be time-consuming. Instead of getting runoff's output at your terminal, you can have it stored in a file ("segment") and then print it on the high-speed printer in Building 39.

To have the formatted text of a document put in a segment, specify the "-segment" option when you give the runoff command. For example, if your document is named "fatmemo.runoff", give the runoff command like this:

```
runoff  fatmemo  -segment
```

The "-segment" option stores the printer-ready text in a special (formatted) file instead of printing it at the terminal. This special file has the same name as that of the input file, but with a suffix of ".runout". In this case, therefore, the formatted text would be stored in the file "fatmemo.runout".

Multics will print a ready message when the runoff command above has finished formatting.[6] You can then have Multics print the

---

[6] NOTE: Any error messages produced by runoff will be printed before the ready message. If you get any such error messages, go back and "repair" the input file (using qedx) and give the runoff

".runout" file on the high-speed printer by using the "dprint" command:

```
dprint  fatmemo.runout
```

Multics will respond with a message saying that the file is waiting to be printed, and will then print a ready line:

```
1 request signalled, 27 already in printer queue 3.
r 1726.4 43.324  2.344
```

In most cases, you will not want to save the ".runout" file, since it occupies additional storage and can be easily reproduced again from the input (".runoff") file, and since any changes you make the the input file will make the ".runout" file obsolete. Therefore, you can tell the dprint command to delete the ".runout" segment automatically after it has been printed, by typing:

```
dprint  -delete  fatmemo.runout
```

instead of the simpler dprint command shown previously.

Output from the IPS high-speed printer (such as the ".runout" file above) can be picked up at the Output Dispatch Counter (Room 39-260). Ask for it by your Multics "ProjectID".[7]

You can send a formatted ".runout" file to other printers too, including some intended to produce high-quality output. For more information, see the descriptions of dprint request types in IPS Memo MS-1.

---

command again (as above) before proceeding. Normally you should not go on to the next step until your runoff command executes with no errors.

[7] To have printer output delivered by courier to the IPS East Campus Computing Facility (Room E52-083), give the dprint command with the destination "EAST-CMPS":

```
dprint  -delete  -ds EAST-CMPS  fatmemo.runout
```

XIII. DETECTING SPELLING ERRORS

LOOKING FOR TYPOS

Although runoff itself cannot check your document for spelling mistakes, you can use the Multics dictionary commands to scan a runout segment for words that are possibly misspelled. The process involves four steps:

- Issue the "runoff" command with the "-segment" control argument to generate a ".runout" file. Do not include the "-hyphenate" control argument because the pieces of the words that were broken across lines will show up later as misspelled "words".

- Give the "create_wordlist" command to produce a list of the unique words in the ".runout" file. The list of words is created in a segment whose name is the same as the ".runout" file's with the suffix ".wl" appended (for example, "nickleby.runout.wl").

- Use the "trim_wordlist" command to check the words in this list against words in one or more "dictionary" segments and remove from the list all words thereby verified. The dictionaries are the same ones runoff uses to hyphenate words; by default, the publicly accessible "standard" Multics dictionary is used.

- Use the "print_wordlist" command to see what misspelled or other suspicious words are left. You can then use an editor to locate and fix the errors in the runoff input segment.

The dictionary commands are part of the Honeywell WORDPRO package. For full information on them, see Multics WORDPRO Reference Guide (AZ98).

A sample session demonstrating the "proofreading" of an input file named "celibacy.runoff" is given below. As usual, commands you type are distinguished from Multics output by an arrow (=>); do not type the arrow.

```
=> runoff celibacy -sm
   r 11:13 4.378 309

=> create_wordlist celibacy.runout
   total number of words = 33
   number of unique words = 31
   r 11:14 1.378 311

=> trim_wordlist celibacy.runout
   number of words trimmed = 25
   number of words remaining = 6
```

```
    r 11:15 2.378 32

  => print_wordlist celibacy.runout
     feeling.*       over-powering       1.B
     frustratation   tht
     gerontocracy    1.A
     r 11:13 1.234 109
```

Two words in the list are actual misspellings, "frustratation" and "tht". Notice, however, that certain things that are correct also show up, such as sections numbers (like "1.B"), words with funny punctuation (like the footnoted "feeling.*"), and real words that are not in the dictionary (like "gerontocracy"). If you find yourself continually wading through wordlists full of correctly spelled technical terms or other specialized words, you may want to create your own private dictionary and use it in conjuction with the standard one when you use the dictionary commands. Creating private dictionaries is discussed in the section below.

USING DICTIONARIES

As we've said in the discussions of hyphenation and spelling-error detection, you do not have to do anything special to gain access to the standard Multics dictionary. Both runoff and the dictionary commands locate dictionaries by using the "dict" search path, which includes ">unb>standard.dict" by default. You can, however, modify your "dict" search path to include other dictionaries, including ones you create.

Suppose you want runoff and the dictionary commands to look up words in another dictionary as well as the standard one. Include the pathname of that dictionary in your "dict" search path. For instance, to add "english.dict", a dictionary in the Author-Maintained Library, type:

```
add_search_paths >aml>english.dict
```

(For a description of "english.dict", see IPS Memo AP-52. For more information on search paths, see the description of the "add_search_paths" command in the Multics Programmer's Manual: Commands and Active Functions, AG92.)

You can also maintain your own dictionary by using the family of commands "add_dict_word", "delete_dict_word", and "list_dict_word". These are also part of the WORDPRO package. For information on them, see Multics WORDPRO Reference Guide (AZ98).

PART 2: RUNOFF REFERENCE DESCRIPTION

A complete reference description of runoff is given on the following pages. This reference section conforms to the format of the Multics Programmers' Manual: Commands and Active Functions (AG92) and may be inserted into that document.

NAME: runoff, rf

FUNCTION: The runoff command produces output in manuscript form from an input file containing lines of text and control words.

SUPPORT LEVEL:

The support level for runoff is ONE. See the IPS User's Guide for an explanation of support levels.

SYNTAX AS A COMMAND:

```
runoff paths {-control_args}
```

where:

1. paths
   are the pathnames of input segments or multisegment files named entryname.runoff. The runoff suffix must be the last component of each entryname; however, the suffix need not be supplied in the command line. If two or more pathnames are specified, they are treated as if runoff had been invoked separately for each one. The segments are printed in the order in which they occur in the invocation of the command.

2. control_args
   can be chosen from the following list. Any control argument specified anywhere in the command invocation applies to all segments; control arguments can be intermixed arbitrarily with segment names. Control arguments must be preceded by a minus sign.

   -ball N, -bl N
   converts output to a form suitable for an N typeball on a unit equipped with a selectric-type typing element. Acceptable ball numbers are 041, 012, 015, and 963. The default is the form of the terminal device being used. Use of this control argument overrides any specification set by the -device control argument (below).

-character, -ch
flags certain key characters in the output by putting the line containing the key character in a segment named entryname.chars. The normal output is not affected. Page and line numbers referring to the normal output appear with each flagged line, and reminder characters, enclosed by color-shift characters, are substituted for the key characters. The default set of key and reminder characters corresponds to those unavailable with a 963 typeball, as follows:

Key	Reminder
left square bracket	<
right square bracket	>
left brace	(
right brace	)
tilde	t
grave accent	'

The key and reminder characters can be changed by use of the .ch control line; specifying a blank reminder character removes the associated key character from the set of key characters. If a key character would print normally in the output, it should also appear in a .tr control line to turn it into a blank in the output.

-device N, -dv N
prepares output compatible with the device specified. This is usually used when the output is stored in a segment to be printed elsewhere. Suitable devices are terminals 2741, 1050, 37, and the bulk output printers, 202 or 300. Use of this control argument overrides any specification set by use of the -ball control argument; if both are used in one invocation of runoff, the last one encountered prevails.

If neither -device nor -ball was specified, the default device type is that from which the user is logged in; any unrecognized device type is assumed to support the entire ASCII character set.

-from N, -fm N
starts printing at the page numbered N. If the -page control argument is used, printing starts at the renumbered page N.

-hyphenate, -hph
When this control argument is used, a procedure named hyphenate_word_ is invoked to perform hyphenation when the next word to be output does not fit in the space remaining in a line (see "Hyphenation Procedure Calling Sequence", page). Otherwise, no attempt is made to hyphenate words.

-indent N, -in N
indents output N spaces from the left margin (default indentation is 0; see also -number below). This space is in addition to whatever indentation is established by use of the .in control word.

-no_pagination, -npgn
suppresses page breaks in the output.

-number, -nb, -map
prints source line numbers in the left margin of the output; minimum indentation of 10 is forced.

-page N, -pg N
changes the initial page number to N. All subsequent pages are similarly renumbered. If the control line .pa is used within the segment, the -page control argument is overridden and the page is numbered according to the .pa control line.

-parameter arg, -pm arg
assigns the argument arg as a string to the internal variable "Parameter". (If arg contains blanks, enclose it in quotation marks.)

-pass N
processes the source segments N times to permit proper evaluation of expressions containing symbols that are defined at a subsequent point in the input. No output is produced until the last pass.

-segment, -sm
directs output to the segment or multisegment file named entryname.runout. This control argument assumes by default that the material is to be dprinted, so the segment is prepared compatible with device 202 unless another device is specified.

-stop, -sp
waits for a carriage return from the user before beginning typing and after each page of output

(including after the last page of output).

-to N
ends printing after the page numbered N.

-wait, -wt
waits for a carriage return from the user before starting output, but not between pages.

NOTES:

A runoff input segment contains two types of lines: control lines and text lines. A control line begins with a period; all other lines are considered text lines. A two- or three-character control word appears in the second and third character positions of each control line. The control word can take a parameter that is separated from the control word by one or more spaces.

Text lines contain the material to be printed. If an input line is too short or too long to fill an output line, material is taken from or deferred to the next text line. A line beginning with white space is interpreted as a break in the text (e.g., the beginning of a new paragraph) and the previous line is printed as is. Lines that are entirely blank are treated similarly to .sp 1 control lines (see the description of the .sp control word).

Nonprinting control characters in the input segment are discarded in the output segment. The .tr control word can be used to print these control characters in the output segment.

When an input text line ends with any of the characters ".", "?", "!", ";", or ":", or with ".", "?", or "!" followed by a double quote or ")", two blanks precede the following word (if it is placed on the same output line), instead of the normal single blank.

The maximum number of characters per input or output line is 361; this permits 120 underlined characters plus the newline character.

TERMINOLOGY:

The following paragraphs describe various terms that are used throughout the runoff description.

Fill and Adjust Modes

Two separate concepts are relevant to understanding how runoff formats output: fill mode and adjust mode. In fill mode, text is moved from line to line when the input either exceeds or cannot fill an output line. Adjust mode right justifies the text by inserting extra spaces in the output line, with successive lines being padded alternately from the right and from the left. Initial spaces on a line are not subject to adjustment. Fill mode can be used without adjust, but in order for adjust to work, fill mode must be in effect.

Line Length

The line length is the maximum number of print positions in an output line, including all spaces and indentations, but not including margins set or implied by the -indent or -number control arguments.

Break

A break ensures that the text that follows is not run together with the text before the break. The previous line is printed out as is, without padding.

Spacing Between Lines

Vertical spacing within the body of the text is controlled by the three control words: .ss, .ds, and .ms (for single, double, and multiple spacing respectively). Single spacing is the default. Multiple spacing is set by the control line .ms N where N-1 is the number of blank lines between text lines.

Page Eject

A page eject ensures that no text after the control line causing the page eject (e.g., .pa for "page") is printed on the current page. The current page is finished with only footers and footnotes at the bottom, and the next text line begins the

following page.

Margins

There are four margins on the page vertically. The first margin on the page is the number of blank lines between the top of the page and the first header; this margin is set by the .m1 control word. The second, set by .m2, specifies the number of lines between the last header and the first line of text. The third (.m3) is between the last line of text and the first footer. The fourth (.m4) is between the last footer and the bottom of the page. The default for the first and fourth margins is four lines; for the second and third, two lines.

Page Numbers

As the output is being prepared, a page number counter is kept. This counter can be incremented or set by the user. The current value of the counter can be used in a header or footer through the use of the symbol "%". A page is called odd (even) if the current value of the counter is an odd (even) number.

The page numbers can be output as either arabic (the default) or roman (using the .ro control word).

Headers and Footers

A header is a line printed at the top of each page. A footer is a line printed at the bottom of each page. A page can have up to 20 headers and 20 footers. Headers are numbered from the top down, footers from the bottom up. The two groups are completely independent of each other. Provision is made for different headers and footers for odd and even numbered pages. Both odd and even headers (footers) can be set together by using the .he (.fo) control words. They are set separately by using the .eh, .oh, .ef, and .of control words.

A header/footer control line has two arguments, the line number (denoted in the control line descriptions as "#"), and the title.

The line number parameter of the control line determines which header or footer line is being set. If the number is omitted, it is assumed to be 1, and all previously defined headers or footers of the type specified (odd or even) are cancelled. Once set, a line is printed on each page until reset or cancelled.

The title part of the control line begins at the first nonblank character after the line number. This character is taken to be the delimiting character, and can be any character not used in the rest of the title. If the delimiting character appears less than four times, the missing last parts of the title are taken to be blank. The three parts of the title are printed left justified, centered, and right justified, respectively, on the line. Any or all parts of the title can be null. Justification and centering of a header or footer line are derived from the line length and indentation in effect at the time of the definition of the header or footer, and are used whenever that line is output, regardless of the values at the time of use. Any occurrence of the special character "%" within a title is replaced by the current value of the page counter whenever the title is printed. To cause a percent character to be printed, "%%" must be written in the title. (The special character can be changed; see the .cc control word.) Moreover, symbols also are automatically expanded--you do not need a .ur control word preceding header or footer control words.

Omitting the title in the control line cancels the header or footer with that number, including its space on the page (e.g., ".he 4" cancels the fourth header). A blank line in the header or footer can be achieved by a title consisting entirely of one delimiting character (e.g., ".fo 3 $" makes the third footer a blank line). Omitting both number and title of a header (footer) cancels all headers (footers) of the type specified (e.g., ".oh" cancels all odd-page headers that were specified by any previous .oh or .he control lines).

EXPRESSIONS AND EXPRESSION EVALUATION:

An expression can be either arithmetic or string, and consists of numbers and operators in appropriate combinations. All operations are performed in integer format, except that string comparisons are performed on the full lengths of the strings.

The order of precedence for the operators is:

^ (bit-wise negation), - (unary)
*, /, \ (remainder)
+, - (binary)
=, <, >, ≠, ≤, ≥ (all are comparison operators that yield -1 for true or 0 for false)
& (bit-wise AND)
| (bit-wise OR), ≡ (bit-wise equivalence)

Other guidelines in the use of expressions are as follows:

1. Parentheses can be used for grouping.

2. Blanks are ignored outside of constants.

3. Octal numbers consist of "#" followed by a sequence of octal digits. Hexadecimal numbers consist of "@" followed by a sequence of hexadecimal digits.

4. String constants are surrounded by the double quote character; certain special characters are defined by multiple-character sequences that begin with the * character, as follows:

**	asterisk character
*"	double-quote character
*b	backspace character
*n	newline character
*t	horizontal-tab character
*s	space character
*cN	character whose decimal value is N (1 to 3 digits)

5. Concatenation of strings is performed by the juxtaposition of the strings involved, in order, left to right.

6. For positive i, k,

   string_expression(i)

   and

   string_expression(i, k)

   are equivalent to the PL/I substr builtin function references

   substr(string_expression, i)

   and

   substr(string_expression, i, k)

   respectively.

7. For negative i, the substring is defined as starting -i characters from the rightmost end of the string; for

negative k, the substring ends -k characters from the end of the string.

8. Evaluation of substrings takes place after any indicated concatenations; string operations have higher precedence than all the binary operations.

9. In any context other than a .sr control line or in a string comparison, a string expression is converted to an integer in such a way that a one-character string results in the ASCII numeric value of the character.

Expression evaluation takes place under the following conditions:

1. In .sr and .ts control lines

2. In all control lines that accept an "N" or "±N" argument

DEFINITION AND SUBSTITUTION OF VARIABLES:

Variables can be defined by the use of the .sr control line; their values can be retrieved thereafter by a symbolic reference. Names of the variables are composed of the uppercase and lowercase alphabetic characters, decimal digits, and "_", with a maximum length of 361 characters. When a variable is defined, it is given a type based on the type of the expression that is to be its value, either arithmetic or string. Variables that are undefined at the time of reference yield the null string, which is equivalent to an arithmetic 0.

In substitution of variables, the name of the variable is enclosed by "%"; other occurrences of the character "%" encountered during substitution of variables are replaced by the value of the page counter; if a "%" character is to occur in the resulting output, it must be coded as "%%" (but see also the .cc control word).

Substitution of variables can occur:

1. In control lines that take an expression argument if a "%" is found as either the first or second character of the argument (substitution of variables takes place before expression evaluation)

2. In .ur control lines

3. In all titles ('part1'part2'part3'), whether in header/footer control lines or as equation lines

Many of the variables internal to runoff are available to the user (a complete list is given at the end of this description). These variables include control argument values (or their defaults), values of switches and counters, and certain special functions. However, the user need not worry about naming conflicts, since an attempt to redefine an internal variable that is not explicitly modifiable is ignored; while the variable is reset to the user's value and no longer reflects internal information, the operation of the command is unaffected.

Two special built-in symbols in runoff are provided for use in footnote and equation numbering: "Foot" contains the value of the next footnote number available (or the current footnote if referred to from within the text of the footnote) and "Eqcnt" is provided for equation numbering. The value of "Foot" is incremented by one when the closing .ft of a footnote is encountered. Any reference to "Eqcnt" provides the current value and causes its value to be incremented by one automatically; thus its value should be assigned to a variable, and the variable should then be used in all further references to that equation number.

ACTIVE STRINGS:

Multics command-language active strings (see Multics Programmer's Manual: Reference Guide) may appear in a runoff input file in any context which permits a symbolic reference. An active string is identified by a combination of the syntax of a runoff symbolic reference and the syntax of a Multics active string: the active string begins with the character sequence "%[" and ends with a similar sequence, "%]". (If the .cc control word is used to change the symbol control character, the new character must be used for active strings also.) But, if an active string contains other active strings nested within it, only the outer brackets require the symbol identifier character. For example:

```
The deadline for this
.ur issue is %[underline [long_date Monday 2weeks] %].
```

The runoff command passes the entire active string to the Multics system for evaluation; thus, within the active string the syntax is that of a Multics command line. The runoff command uses the value string which the system returns to replace the active string in the runoff input, just as if it were the value of a runoff variable.

The active string may contain symbolic references to runoff variables. The runoff command substitutes values for such variables before evaluating the active string.

HYPHENATION:

You may have runoff attempt hyphenation when the space available on a line is less than the length of the next word (including the attached punctuation, if any). Normally, you enable hyphenation by giving the "-hph" control argument on the runoff command line. You may also control hyphenation line by line from within the input file by inserting .hyn, .hyf, and .hy control lines.

The .hyn (hyphenation on) control line instructs runoff to attempt to hyphenate words so as to minimize output-line white space (padding) or the unevenness of the right margin (in no-adjust mode). The .hyn control line may also be used to specify the minimum space which runoff should attempt to fill with a hyphenated word part. The .hyf (hyphenation off) control line tells runoff to stop hyphenating. This is particularly useful for preventing the last word of a paragraph from being hyphenated.

The .hy (hyphenation) control line returns hyphenation processing to the mode initially specified by the presence or absence of the "-hph" control argument in the runoff command line. That is, if the "-hph" control argument was not given, .hy is interpreted as .hyf; if the -hph control argument was given, .hy is interpreted as .hyn. Note, however, that the minimum space that will trigger an attempt to hyphenate may not be specified with .hy; it is automatically reset to 3 spaces (the default).

Hyphenation Procedure Calling Sequence

The runoff command uses an external subroutine to perform hyphenation. This routine works in conjunction with the standard dictionary and is accessible through the standard search rules;

you do not have to provide the subroutine yourself. However, you may want to use your own hyphenation routine; its PL/I calling sequence is prescribed below:

```
declare hyphenate_word_ entry(char(*) unaligned, fixed bin,
          fixed bin);

call hyphenate_word_(string, space, break);
```

where:

1. string
   is the text word that is to be split. (Input)

2. space
   is the number of print positions remaining in the line. (Input)

3. break
   is the number of characters from the word that should be placed on the current line; it should be at least one less than the value of space (to allow for the hyphen), and can be 0 to specify that the word is not to be broken. Thus if the word "calling" is to be split, and 6 spaces remain in the line, the procedure should return the value 4 (adjustment is performed after hyphenation). (Output)

TABULATION

You may designate one or more characters to be converted to a tab on output. It is helpful to put tabs into a document using such a printable trigger character rather than a real tab (ASCII HT), since this makes it much easier to edit the (input-file) lines of a document which include tabs.

Each trigger character may be associated with a different pattern of tabstops, and each pattern of tabstops may have associated with it a string of characters (the "fill string") to replace the white space left by a tab on output (e.g., a line of periods). Default tabstop positions are 11 21 31 ... 10n+1; the default fill string is blanks; and the default trigger character is HT.

Trigger characters encountered in regular text lines are expanded (replaced with fill characters) according to their positions in the resulting output line (i.e., after output-line filling and after any substitution of variables). Otherwise they are treated in the same way as blanks--i.e., if filling causes a tab to

appear at the right end of an output line, the tab is discarded as if it were a blank. Furthermore, if adjust mode is in effect, tabs are subject to padding (addition of blanks) in the same way as blanks. Therefore, precede text consisting of tabulated columns with a .nf control line to preserve horizontal alignment and stop "filling". Remember to reenable filling (.fi) when you want to resume normal (filled) text.

Expansion of tabs on output is controlled by the four control lines .htd, .htc, .htn, and .htf. The .htd (horizontal tab define) control line allows you to specify a pattern of tabstops and give it a "name" by which it will be known later in the input file. Up to twenty such patterns may be defined. You may also specify a replacement string to fill the white space left by an expanded tab. The .htc (horizontal tab cancel) control line may be used to cancel a pattern. The .htn (horizontal tab on) control line associates a trigger character with a particular named tab pattern. A different character may be designated for each named pattern. The .htn control line enables the named pattern; in all text following the .htn control line, the trigger character is replaced by as much of a tab (or by as much of the tab's "fill string") as is required to place the next character at the next tabstop. This continues until runoff encounters a .htf (horizontal tab off) or .htc control line for the named pattern, or a .htn changing the trigger character. Note that more than one tab pattern and trigger character may be in force at one time.

Trigger characters encountered in control lines are treated as blanks except when found within a string expression or a "title" line (.he, .fo, .eq). Trigger characters in a string expression are not expanded until used (i.e., in a text line via .ur, or in a header, footer, or equation line).

Because tabulation is defined in a left-to-right fashion, there is no way to expand a tab according to its position in a centered line (.ce) without redefining the notion of a tab. This is true also for the centered and right-justified portions of "title" lines. Tab trigger characters in such lines are expanded according to their positions in the original specification string, after any substitution of variables.

NOTE: It is inadvisable to use the terminal's TAB key to enter real tabs (ASCII HT) into the input file in the absence of a preceding .htn control line; under these conditions, runoff converts tab characters to the number of spaces required to get to the next tab position (11, 21, ...) before processing anything else on the line(s) containing the tabs. This can cause problems in lines containing, for example, variables to be

evaluated (via .ur).

CHANGE MARKERS

When a document is revised, the author or publisher frequently wants to provide the reader with an easy way to find the differences between the previous editions of the document and the current revision. This usually is accomplished by placing change markers in the margin adjacent to those sections of text which have been amended. Runoff provides this facility via the .bbm (block-begin for markers) and .bem (block-end for markers) control lines.

A change block is a piece of contiguous text which is an amendment to a document or which contains amendments. A change marker is a character or set of characters placed in the margin of a document to call the reader's attention to differences between the current and previous editions of the document. A .bbm control line tells runoff to begin a change block; a .bem control line signals the end of a change block. The change block is the text between the .bbm and the .bem. The .bbm is said to open a change block and the .bem is said to close it. An open block may also be called active. Neither beginning nor ending a change block causes a break.

You may control the character or characters used for the change marker and the position of the marker in the margin with the margin specification of the .bbm control line. A margin specification contains two control characters and a string of characters enclosed in quotation marks ("). One control character (l, r, or b) indicates in which margin (left, right, or both) the change marker is to be placed. The other control character (o, e, or a) allows different change-marker patterns for odd and even pages (or the same pattern for all pages). Thus, a .bbm control line may contain up to four margin specifications. The string of characters enclosed in quotes in the margin specification is placed before (for left-margin markers) or after (for right-margin markers) each line containing text from the the change block, e.g.,

```
.bbm er"   |" ol"|   "
```

would produce:

```
(on an even page):          one line of a change block   |

(on an odd page):      |    one line of a change block
```

The terminals and printers on which runoff output is written print from left to right. Ordinarily the first character of output for each line is the first character of the text line, and ordinarily the paper is positioned so that the leftmost position of the printing device is to the right of the edge of the paper by an amount equal to the space reserved for the left margin. If change markers are used in the left margin different conventions must be used.

On a page with change markers in the left margin, runoff must insert spaces at the left of any output lines which do not have change markers, so that those lines will line up properly at the left margin with the lines having change markers. Runoff can do this only if given warning that change markers may appear in the left margin somewhere in the document. Such warning is given with the ".bbm set" control line at the beginning of the runoff file or after a page-break control line (.pa or .op).

TERMINAL ESCAPE SEQUENCES

Some terminals recognize certain sequences of two or more characters as controls for special terminal functions. Runoff can properly format those two-character sequences in which the first character is either the ASCII ESC character (octal \033) or the ASCII DC1 character (octal \021) and the second character is one which would normally print. This is useful for formatting documents containing superscripts and subscripts. The procedure is detailed below.

Enter the control sequences directly into the ".runoff" input file where needed. The ESC (or DC1) character must be input as "\033" ("\021") since the terminal may interpret it immediately (i.e., as a real terminal escape) if the ESC (or DC1) key is used. No special runoff control lines are associated with the use of terminal escapes, but the following .tr (translate) control line must appear in the input file before the first escape sequence:

```
.tr \033\033      or      .tr \021\021
```

The Multics I/O system currently converts the ESC (DC1) character to the string "\033" ("\021") in terminal output unless you tell it otherwise. To prevent the system from performing this "editing" while you are formatting a file containing escapes (and to reinstate I/O editing later), use the set_tty (stty) command. (See "Suggested Procedure for Use of Escapes", below.)

Telling Multics not to edit output ("raw-output" mode) has other implications, however. In raw-output mode, the linefeed character (ASCII LF; octal \012) which Multics treats as a newline is transmitted without the carriage return character (ASCII CR; octal \015) that normally accompanies it. To obtain properly-formatted final output, therefore, you must runoff the input file into a segment ("runoff filename -segment"), edit the ".runout" file thus produced (changing all linefeed characters to linefeed-and-carriage-return), then request raw-output mode (using the set_tty command), and then print the ".runout" file (using the print command). (See "Suggested Procedure for Use of Escapes", below.)

There is one further consideration. Since, by default, runoff formats "-segment" output as if it were destined for off-line printing, you must give the "-device 37" control argument in the runoff command, which requests formatting for a (full-ASCII) terminal.

Suggested Procedure for Use of Escapes

Step 1: Create or edit a ".runoff" file (filename.runoff) in the usual manner. Enter the ESC character as \033 (or the DC1 character as \021). Include the control line

```
.tr \033\033     or     .tr \021\021
```

before the first escape sequence.

Step 2: Invoke runoff to produce a ".runout" file formatted for a full-ASCII terminal:

```
runoff  filename  -segment  -device 37
```

(You may give any further runoff control arguments in the same command line.)

Step 3: Edit the ".runout" file to change LF to LF CR:

```
qedx
r filename.runout
1,$s/\012/\012\015/
w
q
```

Step 4: Type commands to (a) set I/O system editing to raw-output (rawo) mode, (b) print the ".runout" file, and (c) reinstate I/O system editing. Type these all on one line (separated by semicolons) to avoid the confusion that can result when attempting to type to the system with raw-output mode in effect:

```
stty -modes rawo; print filename.runout 1; stty -modes ^rawo
```

(The argument "1" to the print command avoids the name-and-date header that the print command normally displays.) Before executing this command line, align the paper so that the printer will be at the top of the output page.

To print additional copies, repeat Step 4. Repeat Steps 2 through 4 each time you change the input file (filename.runoff).

An exec_com is provided to perform Steps 2 through 4. To use it, type:

```
ec >IML>rfesc path -args
```

where path is the pathname of the ".runoff" file, and -args may be any runoff control arguments except "-device" or "-segment". Note that this exec_com creates a ".runout" file, in your working directory, which you may wish to delete afterward, to avoid extra storage charges.

DEFAULT CONDITIONS

When no control words are given, runoff prints the text single spaced, right adjusted, with no headers, no footers, and no page numbers.

If page numbers are substituted in headers or equations, they are arabic.

A page consists of 66 lines, numbered 1 through 66. The first line is printed on line 7, and the last on line 60, if no headers or footers are used. If headers are used, there are four lines of top margin (.m1 4), the headers, two blank lines (.m2 2), and then the text. If footers are used, there are two lines skipped after the text (.m3 2), footers printed, and four lines of bottom margin (.m4 4).

A line is 65 characters long; the left margin is that of the typewriter. The output is compatible with whatever is normal for

the device from which the runoff command is executed. The entire segment is printed, with no wait before beginning or between pages.

CONTROL WORD FORMATS

The following discussion gives a description of each of the control words that can be interspersed with the text for format control. Control lines do not cause an automatic break unless otherwise specified. Arguments of the control words are in the following form:

#	integer constant
N	integer expression
±N	integer expression preceded by optional + or - sign
<expression>	arbitrary expression (string or integer)
c	character
cd	character pair
f	segment name
'part1'part2'part3'	a title whose parts are to be left justified, centered, and right justified respectively.

.ad Adjust: text is printed right justified. Fill mode must be in effect for right justification to occur. Fill mode and adjust mode are the default conditions. This control line causes a break.

.ar Arabic numerals: when page numbers (% variable) are substituted into text or control lines as a result of a .ur control line or into a title or equation as it is printed, they are in arabic notation. This is the default condition.

.bbm {key} {margin_specifications}
Block-Begin for Markers: if "key" is omitted, open a change block. A change marker is placed in the margin beside each line of output containing text from this block of input. If a change block is already active the change markers continue, but an error message is printed.

key (optional) may have the value:

set indicates that the margin specifications on the control line are to be used in any .bbm control line which does not have margin specifications of its own. If one or more of the margin specifications on the control line are for the left margin, ".bbm set" also causes runoff to add spaces to the beginning of all output lines not in a change block so as to align them with those that have change markers. The number of spaces is equal to the number of characters, both printing and blank, in the quoted string part of the left margin specifications (see below). If the "set" key is used without a margin specification, the default margin specification (ar" |") is restored. A .bbm control line with this key specified does not open a change block.

margin_specifications are optional. If they are omitted the margin specifications from the most recent ".bbm set" line (or the default margin specification if no ".bbm set" has been encountered) are used for the change markers. A margin specification has the following form:

pm"string_expression"

where

p is the page type. It may be e (even), o (odd), or a (all).

m is the margin type. It may be l (left), r (right), or b (both).

" is the delimiter character.

string_expression
specifies the characters which are to appear in the margin adjacent to each line of the change block, including blanks (see "Change Marker Example", page); the string_expresssion may be up to 20 characters in length.

.bem Block-End for Markers: The current change block is closed. If no change block is active an error

message is produced, but no other action is taken.

.bp Begin page: the next line of text begins on a new page of output. This control line causes a break.

.br Break: the current output line is finished as is, and the next text line begins on a new output line.

.cc c Control character: this control line changes the character used to surround the names of symbolic variables and active strings when they are referenced to c. The default special character is "%". The character specified by c must thereafter be used to refer to symbolic variables and active strings, while percent signs are treated literally. Either ".cc %" or .cc restores the percent sign as the special character.

.ce N Center: the next N text lines are centered. Control lines and blank lines are not counted as part of the N lines being centered. If N is missing, 1 is assumed. This control line implies ".ne N" (or ".ne 2N" if double spacing) so that all lines centered are on the same page. A break occurs.

.ch cd.. Characters: each occurrence of the character c is replaced in the chars segment (the segment named entryname.chars) by the character d, set off by color-shift characters. An arbitrary number of cd pairs can follow the initial pair on the same line without intervening spaces. If the d character is blank, or an unpaired c character appears at the end of the line, the c character is not flagged; it either occurs as itself in the chars segment or not at all if no other character on the line was flagged.

.ds Double space: begin double spacing the text. This control line causes a break.

.ef # 'part1'part2'part3'
Even footer: this defines even page footer line number #. If # is omitted, 1 is assumed. If both #

and the title (parts 1 to 3) are omitted, all even footers defined by any .ef or .fo control lines are cancelled. For more information, see the previous discussion entitled "Headers and Footers", page 38.

.eh # 'part1'part2'part3'
Even header: this defines even page header line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all even headers defined by any .eh or .he control lines are cancelled. For more information, see the previous discussion entitled "Headers and Footers", page 38.

.eq N Equation: the next N text lines are taken to be equations. If N is missing, 1 is assumed. This control line implies ".ne N" (or ".ne 2N" if double spacing) so that all equations are on the same page. The format of the equations should be 'part1'part2'part3' just as in headers and footers.

.ex text Execute: the remainder of the control line (text) is passed to the Multics command processor. Substitution of variables can occur if the first or second character of text is "%".

.fh 'part1'part2'part3'
Footnote header: before footnotes are printed, a demarcation line is printed to separate them from the text. The format of this line can be specified through the title in the .fh control line. This title is printed in the same manner as headers/footers and equations. The default footnote header is a line of underscores from column one to the right margin.

.fi Fill: this control line sets the fill mode. In fill mode, text is moved from line to line to even the right margin, but blanks are not padded to justify exactly. Fill mode is the default condition. This control line causes a break.

.fo # 'part1'part2'part3'
Footer: even and odd footers are set at the same time; this is equivalent to:
.ef # 'part1'part2'part3'
.of # 'part1'part2'part3'
If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all footers are cancelled. For more information, see the discussion entitled "Headers and Footers", page 38.

.fr c
Footnote reset: this control line controls footnote numbering according to the argument c. Permitted values of c are:
t Footnote counter is reset at the top of each page. This is the default condition.
f Footnote counter runs continuously through the text.
u Suppresses numbering on the next footnote.

.ft
Footnote: when .ft is encountered, all subsequent text until the next .ft line is treated as a footnote. Any further text on the .ft line is ignored. If a footnote occurring near the bottom of a page does not fit on the page, as much as necessary is continued at the bottom of the next page. If a footnote reference occurs in the bottom or next to bottom line of a page, the current page is terminated and the line with the footnote reference is printed at the top of the next text page.

.gb STR
Go back: the current input segment is searched from the beginning until a line of the form ".la STR" is found; "STR" in this case means "the rest of the line." Processing is continued from that point.

.gf STR
Go forward: same as .gb, except search forward from the current position in the input segment.

.he # 'part1'part2'part3'
Header: even and odd headers are set at the same time. This is equivalent to:
.eh # 'part1'part2'part3'
.oh # 'part1'part2'part3'
If # is omitted, 1 is assumed. If both # and the

title (parts 1 to 3) are omitted, all headers are cancelled. For more information, see the discussion entitled "Headers and Footers", page 38.

.htc {name1 ... namen}
Horizontal Tab Cancel: cancel the named pattern(s) and turn it (them) off. If no names are given, all patterns are cancelled and turned off.

.htf {name1 ... namen}
Horizontal Tab Off: stop treating the trigger character for each specified pattern as a tab on output. If no names are given, all currently-enabled tab patterns are turned off.

.htd {name} {string_expression} {t1 t2 ... tn}
Horizontal Tab Define: define (or redefine) a tab pattern and (optionally) the fill string used to replace the trigger character for that tab pattern.

where:

name
(optional) is the name by which this pattern of tabulation is to be known in subsequent .htn and .htf control lines. The name may be up to 32 characters in length. One unnamed and up to 20 named tab patterns may be defined. A name composed of a single asterisk (*) also may be used to refer to the unnamed tab pattern.

string_expression
(optional) is the fill string to replace tab trigger characters (see "Tabulation Example", page .). It is a runoff string expression (i.e., it may include SUBSTRs and concatenations). If it is omitted, the fill string is not changed. The default fill string consists entirely of blanks. The default fill string can be restored by specifying the null string ("") as the fill string. If the fill string is shorter than the number of characters needed to replace a given trigger character, it is replicated as necessary. If longer, the last m characters are used to fill m spaces. If the number of characters is not an even multiple of

the length of the fill string, the partial string is truncated at the left and placed at the beginning of the tabspace. The string_expression should not contain overstrikes.

ti is an integer expression giving the position of the ith tabstop (i.e., the position of the first character after the tab) in the pattern being defined. The default tab pattern is 11 21 31 ... 10n+1. The ti's must be separated by blanks. For trigger characters occurring in the output line beyond the last tab position set by the .htd control line, the default tabstops remain 10i+1. The default tabstops may be restored by a .htd line specifying no tabstops (for example, .htd or .htd name "").

.htn name {c}
Horizontal Tab On: associate a trigger character, c, with the named tab pattern and start treating that trigger character as a tab on output. If the trigger character is omitted, the tab character (ASCII HT, octal \011) is used for the trigger character.

.hy Hyphenation (Return to Initial Mode): The initial hyphenation mode is determined by the presence or absence of the -hph control argument on the command line used to invoke runoff. The .hy control line returns runoff to that initial hyphenation mode and resets the smallest space that will trigger an attempt to hyphenate to the default value of 3.

.hyf Hyphenation Off: stop hyphenating. The minimum-space parameter n (above) is not affected by this control line--i.e., it will retain its most recent value if hyphenation is enabled again by a .hyn control line without an n specification.

.hyn {n} Hyphenation On: start attempting to hyphenate words whenever n or more spaces are needed to fill a line. If n is not given, use the value given on the most recent .hyn line. If n has not been specified on any .hyn line, or if a .hy line occurs after the most recent specification, use the default value of 3.

.if f <expression>

Insert file: the segment specified by f is inserted into the text at the point of the ".if f" control line. The inserted segment can contain both text and control lines. No break occurs. The effect is as if the control line were replaced by the segment. Inserts can be nested to a maximum depth of 30. The argument f is the entryname of a runoff input segment; f must not include the .runoff suffix (although the name of the segment itself must have the suffix). If a second argument is provided, it is evaluated in the same fashion as the expression in .sr, and its value and type are associated with the identifier "Parameter"; if no second argument is provided the value of "Parameter" remains unchanged (or undefined). (In either case, "Parameter" is not reassigned its prior value when processing of the insert file completes.)

The input file is located by use of the runoff search list (for more on search lists, see the "add_search_paths" command in the Multics Programmers' Manual: Commands and Active Functions, AG92). If the user does not have a runoff search list, the translator search list is used.

.in ±N

Indent: the left margin is indented N spaces by padding N leading spaces on each line. The right margin remains unchanged. By default N is 0. The margin can be reset with another ".in N" request. Either .in or ".in 0" resets the original margin. If N is preceded by a plus or a minus sign, the indentation is changed by N rather than reset. This control line causes a break.

.la STR

Label: defines the label STR for use as the target of the .gb or .gf control word.

.li N

Literal: this request causes the next N lines to be treated as text, even if they begin with a period (.). If N is not given, 1 is assumed.

.ll ±N

Line length: the line length is set to N. The left margin stays the same, and no break occurs. If N is not given, 65 is assumed. If N is preceded by a plus

or a minus sign, the line length is changed by N rather than reset.

.ma ±N Margins: top and bottom margins are set to N lines. If N is preceded by a plus or a minus sign, the margin is changed by N rather than reset. The margin is the number of lines printed above the first header and below the last footer. If N is not given, 4 is assumed. This control line is equivalent to:

```
.m1 ±N
.m4 ±N
```

Note: Care should be taken in using a top or bottom margin of less than three lines if output is to be directed to an off-line printer (-sm). Such printers typically are set up to skip automatically the first and last 3 lines on each page. Runoff takes this into account by putting out fewer newlines to compensate for the printer; the compensation cannot work for .m1 or .m4 less than 3. It is possible to get around this problem by using the -device 037 control argument when invoking runoff and special control arguments to the command that set up requests for the off-line printer (dprint).

.mp ±N Multiple pages: format the output text so that it prints on every Nth page (i.e., skips N-1 blank sheets of paper between printed pages). This control line is valid only for output intended for the bulk printer. If N is not given, 1 is assumed.

.ms ±N Multiple space: begin multiple spacing text, leaving (N-1) blank lines between text lines. If N is preceded by a plus or a minus sign, the spacing is changed by N rather than reset. If N is not given, 1 is assumed. This control line causes a break.

.m1 ±N Margin 1: the margin between the top of the page and the first header is set to N lines, or changed by N if N is signed. If N is not given, 4 is assumed. See note in description of .ma.

.m2 ±N — Margin 2: the number of blank lines between the last header and the first line of text is set to N, or changed by N if N is signed. If N is not given, 2 is assumed.

.m3 ±N — Margin 3: the number of blank lines printed between the last line of text and the first footer is set to N, or changed by N if N is signed. If N is not given, 2 is assumed.

.m4 ±N — Margin 4: the margin between the last footer and the bottom of the page is set to N lines, or changed by N if N is signed. If N is not given, 4 is assumed. See note in description of .ma.

.na — No adjust: the right margin is not adjusted. This does not affect fill mode; text is still moved from one line to another. This control line causes a break.

.ne N — Need: a block of N lines is needed. If N or more lines remain on the current page, text continues as before; otherwise, the current page is ejected and text continued on the next page. The number of lines remaining is calculated by subtracting from the current page length the sum of the number of lines already printed and the number of lines reserved for footers, footnotes, and bottom margins. No break is implied; if a line is partially formatted but not yet printed when the .ne is encountered, it is ignored in the calculation of lines remaining (i.e., it is neither printed nor in possession of reserved space). Similarly, a footnote or footer defined after the .ne does not have space reserved at the time the .ne is encountered. If N is not given, 1 is assumed. If several .ne control lines occur consecutively, the N's are not added together; only the largest N has effect.

.nf — No fill: fill mode is suppressed, so that a break is caused after each text line. Text is printed exactly as it is in the input segment. This control line causes a break.

.of # 'part1'part2'part3'
Odd footer: this defines odd page footer line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all odd footers defined by any .of or .fo control lines are cancelled. For more information, see the discussion entitled "Headers and Footers", page 38.

.oh # 'part1'part2'part3'
Odd header: this defines odd page header line number #. If # is omitted, 1 is assumed. If both # and the title (parts 1 to 3) are omitted, all odd headers defined by any .oh or .he control lines are cancelled. For more information, see the discussion entitled "Headers and Footers", page 38.

.op
Odd page: the next page number is forced to be odd by adding 1 to the page number counter if necessary. A break is caused and the current page is ejected. No blank even page is made; the even page number is merely skipped.

.pa ±N
the current line if finished as is (i.e., a break occurs) and the current page is ejected. The page number counter is set to N, or is changed by N if N was signed. If N is omitted, the page number counter is incremented by 1.

.pi N
Picture: if N lines remain on the present page, then N lines are spaced over; otherwise, the text continues as before until the bottom of the page is reached, then N lines are skipped on the next page before any text is printed. Headers are printed normally; the space resolved is below the headers. This option can be used to allow for pictures and diagrams. If several .pi control lines occur consecutively, each N is added to the number of lines pending and the total is checked against the space remaining on the page. All pending space is allotted together. If the total is greater than the usable space on a page, the next page contains only headers and footers and the rest of the space is left on the following page. If N is not given, 1 is assumed.

.pl ±N — Page length: the page length is set to N lines. If N is not given, 66 is assumed. If N is preceded by a plus or a minus sign, the page length is changed by N rather than reset.

.rd — Read: one line of input is read from the user_input I/O switch; this input line is then processed as if it had been encountered instead of the .rd control line. Thus it can be either a text line or a control line; a break occurs only if the replacement line is one that would cause a break.

.ro — Roman numerals: when page numbers (% variable) are substituted into text or control lines as a result of a .ur control line or into a title or equation as it is printed, they are in lowercase roman notation. This can be reset to arabic numerals (the default) by use of the .ar control line.

.rt — Return: cease processing characters from the current input segment. If the current input segment was entered by a .if control line in another segment, return to the line following the .if control line.

.sk N — Skip: N page numbers are skipped before the next new page by adding N to the current page number counter. No break in text occurs. This control line can be used to leave out a page number for a figure. If N is not given, 1 is assumed.

.sp N — Space N lines: If N is not given, 1 is assumed. If not enough lines remain on the current page, footers are printed and the page ejected, but the remaining space is not carried over to the next page. The N blank lines are produced in addition to any that may occur automatically due to a .ds or .ms control line. For example, if .sp 4 is used with .ss or .ms 1, in effect four blank lines will appear between two text lines, with .ds or .ms 2, five lines will appear, with .ms 3, six lines.

After skipping the space, the equivalent of a .ne 2 is performed in an attempt to avoid separating the first line of a paragraph at the bottom of a page

from the rest of the paragraph on the next page. The .ne feature may be avoided, if the user so desires, by using a blank line rather than .sp. Otherwise, a blank line is treated as if it were a .sp 1 control line.

This control line causes a break.

Note: A series of .sp control lines such as:

```
.sp a
.sp b
```

is not always equivalent to a single .sp control line whose argument is the sum of the individual arguments:

```
.sp a+b
```

If the .sp a finishes a page, causing a page ejection, b blank lines are produced at the top of the new page. If .sp a+b is used, the space does not appear at the top of the next page.

.sr name <expression>

Set reference: associates value of <expression> with the identifier name. The type of name is set to the type of <expression> (either numeric or string); if the expression is not provided or cannot be properly evaluated, a diagnostic message is printed. The name identifier can be either a user-defined identifier or one of the built-in symbols that the user can set (see "Built-In Symbols", page).

.ss

Single space: begin single spacing text. This is the default condition. This control line causes a break.

.tr cd.. Translate: the nonblank character c is translated to d in the output. An arbitrary number of cd pairs can follow the initial pair on the same line without intervening spaces. An unpaired c character at the end of a line translates to a blank character. (Translation of a graphic character to a blank only in the output is useful for preserving the identity of a particular string of characters, so that the string is neither split across a line, nor has padding inserted within it.) If several .tr control lines are used in a segment, the cd pairs are "added together." Also a particular c character can be translated to a different d character by using a new .tr control line to override the previous translation. To cancel a cd pair (i.e., have the c character print out as itself), use another .tr control line of the form ".tr cc". A .tr control line with no cd pair is ignored.

.ts N Test: process the next input line if the value of N does not equal zero (false). If N is not given, 1 is assumed.

.ty STR Type: write STR (i.e., the rest of the control line) onto the error_output I/O switch. Substitution of variables can occur if the first or second character of STR is "%". If STR is omitted, a blank line is written onto the I/O switch.

.un N Undent: the next output line is indented N spaces less than the current indentation. Adjustment, if in effect, occurs only on that part of the line between the normal left indentation and the right margin. If N is not specified, its value is the current indentation value (i.e., the next output line begins at the current left margin). This control line causes a break.

.ur text — Use reference: the remainder of the .ur control line (text) is scanned, with variables of the form "%name%" replaced by their corresponding values (converted back to character string form if they were numeric), and active strings of the form "%[af args%]" evaluated. The line thus constructed is then processed as if it had been encountered in the original input stream (e.g., it can be another control line, including possibly another .ur). Note that all white space between ".ur" and the next non-blank character is discarded. To get such white space in the output, define a variable equivalent to the white space you want and include a reference to this variable as the first item after the control word.

.wt — Wait: read one line from the user_input I/O switch and discard it (see the .rd control word description).

.* — This line is treated as a comment and ignored. No break occurs.

.~ — This line is treated as a comment and ignored with respect to the output segment. However, the line is printed in the appropriate place in the chars output segment.

SUMMARY OF CONTROL ARGUMENTS

-ball N, -bl N
Convert output to a form suitable for an N typeball.

-character, -ch
Create entryname.chars, listing page and line numbers with red reminder characters where certain characters, normally not printable, must be drawn in by hand.

-device N, -dv N
Prepare output compatible with device N.

-from N, -fm N
Start printing at the page numbered N.

-hyphenate, -hph
Call user-supplied procedure to perform hyphenation.

-indent N, -in N
Set initial indentation to N.

-no_pagination, -npgn
Suppress page breaks.

-number, -nb
Print source segment line numbers in output.

-page N, -pg N
Change the initial page number to N.

-parameter arg, -pm arg
Assign arg as a string to the internal variable "Parameter".

-pass N
Make N passes over the input.

-segment, -sm
Direct output to the segment or multisegment file named entryname.runout, where entryname is the name of the input segment.

-stop, -sp
Wait for a carriage return before each page.

-to N
Finish printing after the page numbered N.

-wait, -wt
Wait for a carriage return before the first page.

SUMMARY OF CONTROL WORDS

The following conventions are used to specify arguments of control words:

#	integer constant
c	character
cd	character pair
exp	expression (either numeric or string)
N	integer expression
+N	+ indicates update by N; if sign not present, set to N
f	segment name
t	title of the form 'part1'part2'part3'

Request	Break	Default	Meaning
.ad	yes	on	Right justify text
.ar	no	arabic	Arabic page numbers
.bbm k m	no	m=ar" |"	Open change block using marker "m", or define but don't open if k="set"
.bem	no		Close current change block
.bp	yes		Begin new page
.br	yes		Break, begin new line
.cc c	no	%	Change special character from % to c
.ce N	yes	N=1	Center next N lines
.ch cd....	no		Note "c" in chars segment as "d"
.ds	yes	off	Double space
.ef # t	no		Defines even footer line #
.eh # t	no		Defines even header line #
.eq N	yes	N=1	Next N lines are equations
.ex text	no		Call command processor with "text"
.fh t	no	line of underscores	Format of footnote demarcation line
.fi	yes	on	Fill output lines
.fo # t	no		Equivalent to: .ef # t .of # t
.fr c	no	t	Control footnote numbering: "t" reset each page "f" continuous "u" numbering suppressed for next footnote
.ft	no		Delimits footnotes
.gb STR	no		"go back" to label STR
.gf STR	no		"go forward" to label STR

Request	Break	Default	Meaning
.he # t	no		Equivalent to: .eh # t .oh # t
.htc n	no		Cancel tab pattern "n"
.htd n s p...	no		Define tab pattern "n" with positions "p", and fill string "s"
.htf n	no		Disable tab pattern "n"
.htn n c	no		Enable tab pattern "n" using trigger "c"
.hy	no		Reset hyphenation to initial mode
.hyf	no		Stop hyphenating
.hyn N	no	N=3	Try to hyphenate words if N space needed to fill line
.if f exp	no		Segment f.runoff inserted at point of request; value of "exp" assigned to "Parameter"
.in ±N	yes	N=0	Indent left margin N spaces
.la STR	no		Define label STR
.li N	no	N=1	Next N lines treated as text
.li ±N	no	N=65	Line length is N
.ma ±N	no	N=4	Equivalent to: .m1 ±N .m4 ±N
.mp ±N	no	N=1	Print only every N-th page
.ms ±N	yes	N=1	Multiple space N lines
.m1 ±N	no	N=4	Margin above headers set to N
.m2 ±N	no	N=2	Margin between headers and tect set to N
.m3 ±N	no	N=2	Margin between text and footers set to N
.m4 ±N	no	N=4	Margin below footers set to N
.na	yes	off	Do not right justify
.ne N	no	N=1	Need N lines; begin new page if not enough remain
.nf	yes	off	Do not fill output lines; print them exactly as entered
.of # t	no		Defines odd footer line #
.oh # t	no		Defines odd header line #
.op	yes		Next page number is odd
.pa ±N	yes		Begin page N
.pi N	no	N=1	Skip N lines if N remain; otherwise skip N lines on next page before any text
.pl ±N	no	N=66	Page length is N
.rd	no		Read one line of text from the user_input I/O switch and process it in place of .rd line

Request	Break	Default	Meaning
.ro	no	arabic	Roman numeral page numbers
.rt	no		"Return" from this input segment
.sk N	no	N=1	Skip N page numbers before next new page
.sp N	yes	N=1	Space N lines
.sr sym exp	no		Assign value of "exp" to variable named "sym"
.ss	yes	no	Single space
.tr cd....	no		Translate nonblank character c into d on output
.ts N	no	N=1	Process the next input line only if N is not zero
.ty STR	no		Write "STR" onto the error_output I/O switch
.un N	yes	left margin	Indent next text line N spaces less
.ur text	no		Substitute values of variables and active strings in "text", and scan the line again
.wt	no		Read one line of text from the user_input I/O switch and discard it (for synchronization with terminal)
.*	no		Comment line; ignored
.~	no		Comment line; ignored

BUILT-IN SYMBOLS

Only those symbols marked yes in the Set column can have values assigned by the user.

All symbols are of type Number unless they are specified to be of type String.

Control words and control arguments that affect the values of the variables are indicated in parentheses: (x/y) indicates that x sets the switch to true (-1), and y sets it false (0); (a) or (a, b, c) indicates that it is affected by a or by a, b and c.

Symbol	Set	Value
Ad		Adjust (.ad/.na)
Ce		Number of lines remaining to be centered (.ce)
CharsTable	yes	Translation table for chars segment output (String) (.ch)
Charsw	yes	A chars segment is being created (-character)
ConvTable	yes	Translation table for output. Product of DeviceTable and TrTable (String) (.tr, -device)
Date		Date of this invocation of runoff; format is mm/dd/yy (String)
Device	yes	Type of device output is to be formatted for (-device, -ball, -segment)
DeviceTable	yes	Translation table for physical device (String) (-device)
Eq		Equation line counter (.eq)
Eqcnt	yes	Equation reference counter (incremented each reference)
ExtraMargin	yes	Indent entire text this many spaces (-indent)
Fi		Fill switch (.fi/.nf)
FileName		Name of current primary input segment (String)
Filesw		True if output is going to a segment (-segment)
Foot	yes	Footnote counter (.ft, .fr)
FootRef	yes	Footnote reference string in footnote body (String)
Fp	yes	First page to print (set at the beginning of each pass to the value

Symbol	Set	Value
		of From)
Fr		Footnote counter reset switch
From	yes	First page to print (-from)
Ft		Footnote processing switch (.ft)
Hyphenating	yes	True if an attempt to break a word should be made (-hyphenate, .hy, .hyn/.hyf)
In		Indent to here (.in)
InputFileName		Name of current input segment (String) (.if)
InputLines		Current line number in current source file
LinesLeft		Number of usable text lines left on this page
Ll		Line length (.ll)
Lp	yes	Last page to print (initialized each pass from To)
Ma1		Space above header (.ma, .m1)
Ma2		Space below header (.m2)
Ma3		Space above foot (.m3)
Ma4		Space below foot (.ma, .m4)
Ms		Spacing between lines (ss = 1, ds = 2, etc.) (.ms, .ss, .ds)
MultiplePagecount		Form feeds between pages to printer (.mp)
NestingDepth		Index into stack of input files (.if)
Nl		Last used output line number on current page
NNp	yes	Next page number (-page, .pa, .op)
NoFtNo		True to suppress number on next footnote reference (.fr)
NoPaging	yes	True if no pagination is desired (-no_pagination)
Np	yes	Current page number (-page, .pa, .op, initialized each pass from Start)
PadLeft		Alternate left/right padding switch (.un, .ad)
Parameter	yes	Argument passed during insert processing (-parameter, .if)
Passes	yes	Number of passes left to make (= 1 when printing is being performed) (-pass)
Pi		Space needed for pictures (.pi)
Pl		Page length (.pl)
Print	yes	Whether or not to print

Symbol	Set	Value
		((Fp ≤ Np ≤ Lp) & (Passes ≤ 1))
Printersw		Output is intended for bulk printer (-device, -segment)
PrintLineNumbers	yes	True if source line numbers are to be printed in output (-number)
Roman		Roman numeral pagination (.ro/.ar)
Selsw		True if typeball other than 963 is being used (-ball)
Start	yes	Initial page number (-page)
Stopsw	yes	Stop between pages of output (-stop)
TextRef	yes	Footnote reference string in main text (String)
Time		Local time, in seconds, since January 1, 1901.
To	yes	Last page to be printed (-to)
TrTable	yes	Translation table for user-supplied substitutions (String) (.tr)
Un		Undent to here (.un)
Waitsw	yes	Wait for input before printing first page (-wait)

EXAMPLES

Sample Session

The following pages show the creation of a runoff segment and the result of invoking the runoff command on that segment. For an explanation of any of the control lines, refer to the respective control word definition earlier in this command description. Particularly notice the following:

1. The line length control is given before any headers and footers. If the user wants a line length other than the default one, he must specify it before he specifies his headers and footers; if he does not, the headers and footers on the first page are formatted for the default line length.

2. The .sr control line associates the page number count, at the time the paragraph on headers and footers is printed, with the identifier headfoot. Refer to the last line of the segment (the .ur control line) to see how this reference is used.

3. The translate character (!) is used both to "count" spaces (see the a, b, and c items of 2, below) and to prevent an unattractive line split (see the last line of the segment).

```
qedx
a
.tr !
.fo 3 $runoff sample$page %$runoff sample$
.he "-------------""-------------"
.he 2 XrunoffXXrunoffX
.he 3 XdemonstrationXXdemonstrationX
.he 4 8-------------88-------------8
.m1 6
.m2 3
.m3 2
.m4 6
.sp 7
.ce
RUNOFF SAMPLE PAGES
.sp 2
     The runoff command lets
the user format his text segments through
a variety of control words.
The control words specify such things as:
```

```
.sp 2
.in 10
.un 5
1.   Page length and line length (.pl and .ll respectively).
If not specified by the user, these control words are given
default values of:
.sp
.in +5
.li
.pl 66
.br
.li
.ll 65
.in -5
.sp
.un 5
2.   Headers and footers, for all pages or for just
.sr headfoot %
odd numbered or just even numbered pages.  The
control words for
headers and footers are as follows:
.sp
.in +5
.un 5
a.!!!Headers and
footers on both odd and
even numbered pages (.he and .fo)
.sp 1
.un 5
b.!!!Headers and footers on just odd
numbered pages (.oh and .of)
.sp 1
.un 5
c.!!!Headers and footers on just
even numbered pages (.eh and .ef)
.sp 1
.in -5
.un 5
3.   Margins that control
vertical spacing in relation to the top
of the page, headers, text, footers, and the
bottom of the page.
These margins are defined as follows:
.sp
.in +5
.un 5
a.   Between top of page and first header (.m1)
.sp
.un 5
```

```
.bp
b.   Between last header and first line of text (.m2)
.sp
.un 5
c.   Between last line of text and first footer (.m3)
.sp
.un 5
d.   Between the last footer and bottom of page (.m4)
.in -5
.sp
If not specified by the user, these margins are given default
values of:
.sp
.in +5
.nf
.li 4
.m1 4
.m2 2
.m3 2
.m4 4
.fi
.in 0
.ce 3
.li 3
.
.
.
.sp 2
     If you're wondering about the footer below, refer to
.ur the discussing of headers and footers on page!%headfoot%.
\f
w example.runoff
q
<ready message>

rf example -wt
```

runoff
demonstration

RUNOFF SAMPLE PAGES

The runoff command lets the user format his text segments through a variety of control words. The control words specify such things as:

1. Page length and line length (.pl and .ll respectively). If not specified by the user, these control words are given default values of:

   ```
   .pl 66
   .ll 65
   ```

2. Headers and footers, for all pages or for just odd numbered or just even numbered pages. The control words for headers and footers are as follows:

   a. Headers and footers on both odd and even numbered pages (.he and .fo)

   b. Headers and footers on just odd numbered pages (.oh and .of)

   c. Headers and footers on just even numbered pages (.eh and .ef)

3. Margins that control vertical spacing in relation to the top of the page, headers, text, footers, and the bottom of the page. These margins are defined as follows:

   a. Between top of page and first header (.m1)

runoff
demonstration

b. Between last header and first line of text (.m2)

c. Between last line of text and first footer (.m3)

d. Between the last footer and bottom of page (.m4)

If not specified by the user, these margins are given default values of:

```
.m1 4
.m2 2
.m3 2
.m4 4
```

.
.
.

If you're wondering about the footer below, refer to the discussing of headers and footers on page 1.

Tabulation Example

Tabs are useful tools for setting up columns. In this example, two sets of columns, using two different tab patterns, are produced.

The following control lines:

```
.htd NUMBERS 10 25 40
.htn NUMBERS $
```

define a tab pattern named "NUMBERS" and a trigger character, "$", to be used in the input file as (and converted on output to) a tab. The following input-file lines:

```
.nf
$Cardinal$Ordinal$Numeral
.sp
$one$first$   1
$two$second$   2
$three$third$   3
```

will then produce a table in this form:

```
    Cardinal        Ordinal         Numeral

    one             first             1
    two             second            2
    three           third             3
```

A different tab pattern may be defined, with a different trigger character, and may be in effect at the same time. Furthermore, a repeatable string of characters (the "fill string") may be specified in the .htd control line, to replace the tab trigger character on output. Thus, the following input-file lines:

```
.htd NUMBERS 10 25 40
.htn NUMBERS $
.htd DAYS ". " 15 30 45
.htn DAYS >
.nf
$one$first$1
$two$second$2
$three$third$3
.sp
Monday>Tuesday>Wednesday>Thursday
Friday>Saturday>Sunday>today
```

would produce output like this:

```
        one             first           1
        two             second          2
        three           third           3

Monday. . . . Tuesday. . . . Wednesday. . . Thursday
Friday. . . . Saturday . . . Sunday . . . . today
```

Note that, since the fill string has a length of more than one character, it is truncated at the left as necessary when the length of the space to be filled is not an even multiple of the length of the fill string. Tab patterns may be turned off and on repeatedly and independently. If a control line reading ".htf NUMBERS" were inserted, say, following the .nf control line in the input sequence shown above, the output would look like this:

```
$one$first$1
$two$second$2
$three$third$3

Monday. . . . Tuesday. . . . Wednesday. . . Thursday
Friday. . . . Saturday . . . Sunday . . . . today
```

The DAYS tab pattern is still in effect, but NUMBERS has been disabled. It may be reactivated at any time by respecifying a ".htn NUMBERS $" control line. A .htc control line cancels the specified tab pattern(s), whether they are enabled (on) or disabled (off). A .htc control line which does not specify any tab pattern(s) cancels all defined patterns.

Change Marker Example

The following sequence shows an example of the use of change markers:

> Smalltown University maintains a Xerox facility in the Student Center. The copier is located in Room 927, and is maintained by four work-study students. The Xerox Room is open Monday through Friday from 8 AM to 5 PM. The per-copy charge is $.05. During exam periods, the office tends to be quite busy; to obtain the best service at exam time, avoid the peak hours (noon to 3 PM).

The paragraph above might be revised, for example, to reflect changes in costs or schedules. The input file might then look like this:

```
.bbm set al">>  "
.ll 55
Smalltown University maintains a Xerox facility in the
Student Center.
The copier is located in Room 927, and is maintained
by four work-study students.
.bbm
The copying service is available Monday through
Tuesday from 11 AM to 3 PM.
Rates for use of the facility are $.20 per copy
with a $3.00 minimum.
.bem
During exam periods, the office tends to be quite busy;
to obtain the best service at exam time,
avoid the peak hours (noon to 3 PM).
```

The new document would then look like this:

> Smalltown University maintains a Xerox facility in the Student Center. The copier is located in Room 927, and
> >> is maintained by four work-study students. The copying
> >> service is available Monday through Tuesday from 11 AM
> >> to 3 PM. Rates for use of the facility are $.20 per
> >> copy with a $3.00 minimum. During exam periods, the office tends to be quite busy; to obtain the best service at exam time, avoid the peak hours (noon to 3 PM).

Escape Sequence Example

The following short runoff input file (named einstein.runoff) contains the escape sequences required by a Diablo 1620 terminal for half-index-up (ESC-U) and half-index-down (ESC-D). (Note that escape functions often differ from terminal to terminal; consult your terminal's operating manual for specific codes.)

```
.tr \033\033
The theory of relativity is written
as E=mc\033D2\033U, which is no more difficult
to format than, say, the common representation for
a member of a two-dimensional matrix (A-sub-i-j),
written A\033Uij\033D.
```

The command:

```
ec >IML>rfesc einstein
```

would produce appropriate output on the Diable 1620.

Remember that this feature works only for on-line output, on a terminal whose available escape functions include those requested (and in the form specified for the terminal used). Other devices (including the off-line printer) will not format the escape as desired.