To: Distribution

From: Robert S. Coren

Date: 01/22/76

Subject: Canonicalization of Terminal Input

INTRODUCTION

In theory, terminal input to Multics is converted by the ring-zero typewriter DIM to "canonical form", i. e., the physical appearance of a line uniquely defines the form in which it will be stored. In addition, well-defined meanings are attached to input streams containing erase, kill, and escape characters.

In actual fact, the current typewriter DIM does not meet the goals described in the preceding paragraph. The three basic types of canonicalization (column assignment, erase/kill, and escape) are each handled more or less correctly, but the current design does not lend itself to correct and consistent processing of combinations of canonicalization types. The trouble is that the three types are handled more or less simultaneously. Thus the final input resulting from strings such as "\027", "\016#7", "\000", "#025", etc., is not predictable under the current implementation.

A redesigned, more efficient version of tty_read is planned for Multics release 4.0; in the course of the new design, canonicalization will be cleaned up and made consistent. The details of this new design will be discussed in a future MTB; the purpose of the present document is to set forth a complete description of the rules of canonicalization that the new tty_read will implement. It is proposed that the rules described here be adopted as a standard for all situations in Multics where canonicalization is required.

CANONICALIZATION_RULES

The three types of canonicalization named above must be performed separately in a defined order, to ensure consistency and predictability. In particular, the canonicalization process

is conceptually divided into the following steps:

1. If the terminal is in "can" mode, perform column-assignment canonicalization on the typed input.

2. If the terminal is in "erkl" mode, perform erase/kill canonicalization on the result of step 1.

3. If the terminal is in "esc" mode, perform escape canonicalization on the result of step 2.

Of course, the actual implementation does not necessarily have to perform the three steps in sequence, provided that the result is the same as would have been achieved by doing so.

The three types of canonicalization are discussed in more detail below. If two or more of the rules listed below are applicable to a given input string, they are applied in the order in which they are presented here.

COLUMN ASSIGNMENT

This phase is concerned with determining which printing graphics, if any, appear in each physical column position. This is determined according to the following rules.

Rules for the Interpretation of Input Characters

1. The leftmost position of the carriage is considered to be column 1.

2. Each printing graphic or space typed increases the column position by 1.

3. Each backspace typed decreases the column position by 1 unless the column position is 1.

4. A carriage return sets the column position to 1.

5. A horizontal tab increases the column position to the next tab stop; tab stops are defined to be at columns 11,

21, 31, etc.

6. A newline, form feed, or vertical tab sets the column position to 1 and advances the carriage vertically; thus no character typed after such a character can share a column position with a character typed before it.

Rules for the Formation of the Canonical String

7. Characters on each line are sorted so that their associated column positions are monotone increasing.

8. No carriage return characters may appear in the canonical string.

9. A horizontal tab is preserved as typed unless a printing graphic appears in one of the columns skipped by the tab, in which case the tab is replaced by an appropriate number of spaces.

10. Backspaces appear in the canonical string only when two or more printing graphics share a column position.

11. When two or more different printing graphics share a column position, the characters are sorted as follows: graphic with lowest numeric ASCII code, backspace, graphic with next lowest numeric ASCII code, etc.

12. If the contents of a column position consist of two or more instances of the same printing graphic, that column is reduced to a single instance of the graphic.

13. A line-ending character (newline, form feed, or vertical tab) immediately follows the last printing graphic in the rightmost column position on the line.

ERASE AND KILL CHARACTERS

The placement of erase/kill canonicalization after column-assignment canonicalization and before escape canonicalization is strategic in that it causes erase/kill processing to work by column position rather than by character. This eliminates ambiguity with respect to erase characters combined with escape sequences. (See the examples at the end of this document.)

The rules for erase and kill canonicalization are given below.

14. An erase character alone in a column position results in the deletion of itself and of the contents of the preceding column position.

15. An erase character alone in a column position and preceded by more than one blank column results in the deletion of all immediately preceding blank columns, as well as of the erase character.

16. An erase character sharing a column position with one or more printing graphics results in the deletion of the contents of that column position.

17. A kill character results in the deletion of its own column position and all column positions to its left, unless it shares a column position with an erase character, in which case rule 16 applies (the kill character is erased).

18. If the terminal is in "esc" mode, an erase or kill character alone in a column immediately preceded by an escape character alone in a column is not processed as an erase or kill character.

Note that for rule 18 to apply, the erase or kill character must actually have been typed in the column immediately following the escape character. The reason for this is that it facilitates the erasing of escape sequences, e.g., \001####.

ESCAPE SEQUENCES

The processing of escape sequences is performed according to the rules given below.

19. An escape sequence consists of an escape character alone in its column position followed by one or more printing graphics each of which is alone in its column position. An escape sequence is replaced by a single character in the canonical string.

20. An escape sequence consisting of two successive escape characters is replaced by an escape character.

21. An escape sequence consisting of an escape character followed by an erase (or kill) character is replaced by an erase (or kill) character.

22. An escape sequence consisting of an escape character followed by one, two, or three octal digits is replaced by the character whose ASCII value is represented by the sequence of octal digits.

23. An escape character followed by a newline character results in the deletion of both characters from the canonical string.

24. Other escape sequences may be defined on a per-terminal-type basis, where such a sequence consists of an escape character and one character following.

25. If the character following an escape character does not result in an escape sequence as defined by rules 20-24, the escape and following characters are stored as they appear on the line.

EXAMPLES

In the examples below, the following conventions are used:

<NL>	represents a newline
<CR>	represents a carriage return
<BS>	represents a backspace
<HT>	represents a horizontal tab
<SP>	represents a space
(nnn)	represents a character whose ASCII value is nnn (octal)
\	is the escape character
#	is the erase character
@	is the kill character

The examples in the first group illustrate how various typed sequences are canonicalized in terms of column position; these are followed by examples of erase, kill, and escape

canonicalization. In the second group, lines are shown as they appear physically, with no consideration given to the precise sequence of keystrokes that might have produced them.

COLUMN CANONICALIZATION EXAMPLES

Example_1

Typed: Nothing special about this line.<NL>

Appearance: Nothing special about this line.

Result: Nothing special about this line.<NL>

Example_2

Typed: Extraneous white s<SP><BS>pace is ignored.<CR><SP><NL>

Appearance: Extraneous white space is ignored.

Result: Extraneous white space is ignored.<NL>

Example_3

Typed: Iwo ways (2<BS>_) to overstrike.<CR>___<NL>

Appearance: Iwo ways (2) to overstrike.

Result: I<BS>__<BS>w_<BS>o ways (2<BS>_) to overstrike.<NL>

Example_4

Typed: Tab + backspace is<HT><BS>reduced to spaces.<NL>

Appearance: Tab + backspace is reduced to spaces.

Result: Tab + backspace is<SP><SP><SP><SP>reduced to spaces.<NL>

(See rule 9.)

ERASE-KILL AND ESCAPE EXAMPLES

Example 5

Appearance: abz#cde

Result: abcde

Example 6

Appearance: ab #cde

Result: abcde

Example 7

Appearance: Not@Never o#n Sunday.

Result: Never on Sunday.

Example 8

Appearance: Nox#w it's right.

Result: Now it's right.

Example 9

Appearance: Nox#w it's right.

Result: Noxw it's right.

(Erase character is overstruck; see Rule 16.)

Example_10

Appearance: dcl rrs char (1) static init("\017#6");

Result: dcl rrs char (1) static init("{016)");

Example_11

Appearance: \023

Result: {002}3

(Overstruck 3 is not part of escape sequence.)

Example_12

Appearance: \112

Result: \112

(Overstruck \ is not an escape character.)

Example_13

Appearance: a\##b

Result: a\b

(First # is not an erase character by rule 18; second # erases itself and preceding # by rule 14.)

Example_14 (similar to Example 13)

Appearance: a\@#b

Result: a\b

Example_15

Appearance: a\#@b

Result: b

(The \ is erased by the overstruck #.)

Example_16

Appearance: a\\#b

Result: a\#b

(Erase canonicalization does not recognize the # by rule 18; escape canonicalization recognizes \\ by rule 20, and attaches no special meaning to the #.)

Example_17

Appearance: a\\##b

Result: a\b

(By rule 18, the first # is not an erase character; by rule 14, the second # erases itself and the preceding #; then rule 20 reduces \\ to \.)

Example_18

Appearance: a\\###b

Result: a\b

(The first # is not an erase; the next two are, erasing the second \ and the first #.)

Example 19

Appearance: a\\####b

Result: ab

(The first # is not an erase, and must be erased before the two \ characters. Examples 16-19 illustrate the difficulty of erasing a double \; the clearest method is probably to overstrike (a\\#\\#b).)

Example 20 (on 2741-like terminal)

Appearance: at<#b

Result: a\b

(Only the < is erased; t is translated to \.)