TO: MSPM Distribution
FROM: J. H. Saltzer
SUBJ: BC.2.00
DATE: 04/14/67

The introduction to character I/O, BC.2.00, has been updated and revised to include suggestions which have come from many sources. The primary change is that the order of erase editing and escape processing is reversed.

Published: 04/14/67
(Supersedes: BC.2.00, 01/02/66)

Identification

Introduction: Character Input/Output for Multics
J. H. Saltzer, F. J. Corbató, J. F. Ossanna

Purpose

The following subsections of this section set forth standards and conventions for character input, output, and storage within Multics. The subsections are divided as follows:

BC.2.00 Introduction: Character Input/Output for Multics.

BC.2.01 ASCII character set.

BC.2.02 The canonical form.

BC.2.03 Erase and Kill conventions.

BC.2.04 Escape conventions.

BC.2.05 Requirements for DIM specifications

Character Set and Escapes

All character input and output on Multics is done in terms of a single standard character set, the revised ASCII set. All of the graphics of the ASCII set are acceptable in the interface between the Multics I/O system and the user. Therefore, one character (the left slant) has been reserved by the I/O system as an escape character to allow input and output of missing characters on devices which do not have the full ASCII graphic complement. (To minimize the visual impact of escape sequences for commonly used characters, certain overstrike combinations are also reserved to represent these characters.) Certain of the ASCII controls (e.g., "New line" or "Backspace") are given specific meanings by the I/O system and are functionally performed wherever possible on output, or else they are printed with a graphic escape. The remaining control characters may be typed in or out using graphic escape conventions.

Thus stored character strings are made up of arbitrary ASCII characters. The printed equivalent of a stored string may or may not include escape sequences or overstrikes for some graphics or controls, depending on whether or not the device on which the string was printed can represent or perform the ASCII characters involved.

Section BC.2.01 describes the ASCII set as used on Multics, while BC.2.04 describes the complete set of escape conventions.

In this section we first discuss input character processing, since it is more complicated; a short discussion of output character processing then follows.

Canonical Form

The I/O system rearranges all character strings read through it into a standard or canonical form. After this rearrangement, the characters in the stored string are in the order that they appear on the printed page, rather than the order that they were typed. (If no special controls such as backspace were typed, the stored string is already in that order, so no reordering is done.) Multiple characters appearing in the same column, or print position, as a result of using backspace or similar functions, are arranged in the stored string so that all of the components of a single print position can be easily identified. Canonicalization is done on a line-by-line basis.

A general objective of the transformation to canonical form is that a given printed line image should be represented by a unique string of characters. To this end, all carriage motion (tabs, backspaces, and spaces) which results in white space appearing between printed graphics is represented in the stored string simply as some number of space characters. Further conventions are made to permit half-line forward and reverse feed control characters to add subscripts and superscripts. Section BC.2.02 describes the conventions of the canonical form completely and precisely.

Erase and Kill editing

Since the typical human typist makes many mistakes, the I/O system provides two standard editing conventions to allow him to correct noticed blunders on the spot. Two graphics are reserved (the typist may choose which two if he does not like the system default choice of the number sign and at sign) to mean erase and kill, respectively.

When the erase character appears in a print position by itself, the preceding print position and the print position containing the erase character itself are discarded from the line. When the kill character appears in a print position, it and all preceding print positions in that line are discarded. Processing is arranged in such a way that erase and kill editing can be applied to incorrectly typed escape sequences.

By making the erase and kill conventions apply to print positions, two advantages are gained: the character string ultimately stored depends only on the appearance of the typed line and not on the order of typing; and the typist can easily perform overstriking of graphics in the print positions preceding typed erase characters.

Erase and kill editing are described in detail in BC.2.03.

Break detection

The final service performed by the I/O system on character streams from interactive typewriters is break character detection. Break character detection allows a typist to tell his program "all right, I have typed a complete message; look at it." The arrival of a break character is the signal that the processing described above should be done, and that the resulting message should be handed to the program. To understand the way in which break characters are handled, the block diagram of figure one provides a conceptual picture of character flow on input. The term conceptual is appropriate, since the functions represented by the various blocks are probably tightly intertangled in a few modules to minimize character buffering and handling.

Coming from the typewriter is a raw stream of characters, generated by the typewriter hardware. Also generated by the typewriter and channel hardware are system interrupts, generally coincident with certain characters, but also occurring on things such as hardware buffer overflow. The character stream and interrupt source are indicated at the left side of the diagram. Data flow, including direction, is indicated by dotted lines, and solid arrows indicate closed subroutine calls.

At the right side of the figure is the user program, which when it needs more input calls on its own syntax scanner. The syntax scanner calls upon the library I/O system for input from the stream being typed at the typewriter. Generally, the user does not want the entire (unending) stream but rather a certain part, such as a fixed number of characters, or all characters up to and including the next occurrence of a specific character or pattern of characters. This character or pattern is known as a break.

Typically, the typist has not yet typed a message, much less a break, so the I/O system will initiate I/O by calling the GIOC interface module, and block this process pending arrival of a break.

The typist produces a stream of characters, and occasionally generates an interrupt. He does not generate the interrupt explicitly, but does so rather as a result of typing certain characters (such as new line). The specific characters which cause interrupts are established by the I/O system when the user connects his program to the typewriter and defines his break pattern. For example, if the break pattern is "any amount of horizontal carriage motion," the I/O system must arrange for hardware interrupts whenever the typist types a space, a horizontal tab, or a new line.

When the interrupt arrives, the typewriter manager process is awakened. The raw stream of input from the typewriter is now passed through the following four operations in order:

1. Convert the raw string of characters from the device into a "line image" in which the graphics typed in each print position are collected together and arranged in storage in the order that they appear on the printed page, rather than the order in which they were typed. If the raw device character codes were not ASCII, they are converted to ASCII in this step.

2. Perform erase and kill editing on the stored print positions. The erase-changing and kill-changing escape sequences are recognized as part of this step.

3. Remove all remaining defined escape sequences from the stored string, combining print positions when an escaped backspace character is encountered, and inserting space for escaped tabs. If the typed string contained escaped backspaces or tabs, the stored line image may now differ considerably from the originally typed line image.

4. Produce a standard (canonical) string of ASCII characters which represents the resulting line image.

The final string of characters is returned to the calling program. Note that by this time it is possible that the character which actually triggered the break is not the last one in the string, because of line-image rearrangement, or may not even be in the string any more, because of erase editing or canonicalization.

The user may now, of course, further process the input for his program in whatever way he likes, as indicated by the "syntax scanner" box on the block diagram.

Our simple illustration here does not attempt to include the complications of typewriter read-ahead, an additional function also performed by the device interface module if requested. A description of typewriter read-ahead strategy will be found in BF.11.

Character Output

Output character processing is considerably simpler than input, since there is no equivalent of erase and kill processing. The basic objective of conventions made on output is that the printed page should provide an unambiguous representation of the stored string offered for output. This objective needs to be completely realized only for stored strings which are in canonical form.

Graphic characters which are not available on the output device are represented either as escape sequences or as particular (reserved) overstrike combinations. Defined Multics control characters are functionally imitated, if not directly available on the device, or if inimitable they are printed as escapes (as for example, the red shift character on an all-black device). Control characters which are not defined for Multics are printed as octal escapes.

A device may have specialized control functions not included in the list of defined Multics control characters (e.g., a single code causing a jump to the 3rd tab stop, followed by a single backspace.) In such cases, the programmer cannot exercise direct control of these functions through

the standard device interface modules. Instead, any time a sequence of defined characters (e.g., 29 successive blanks) is to be typed which could profitably use some function, a clever device interface module may invoke it. In general, the strategy of the device interface module is to make as full use as possible of the available features of the device to accomplish speedy and reliable printing of canonical ASCII strings. For another example, the DIM is <u>not</u> constrained to type characters in the order they appear in the stored string if there is another order which would type faster.

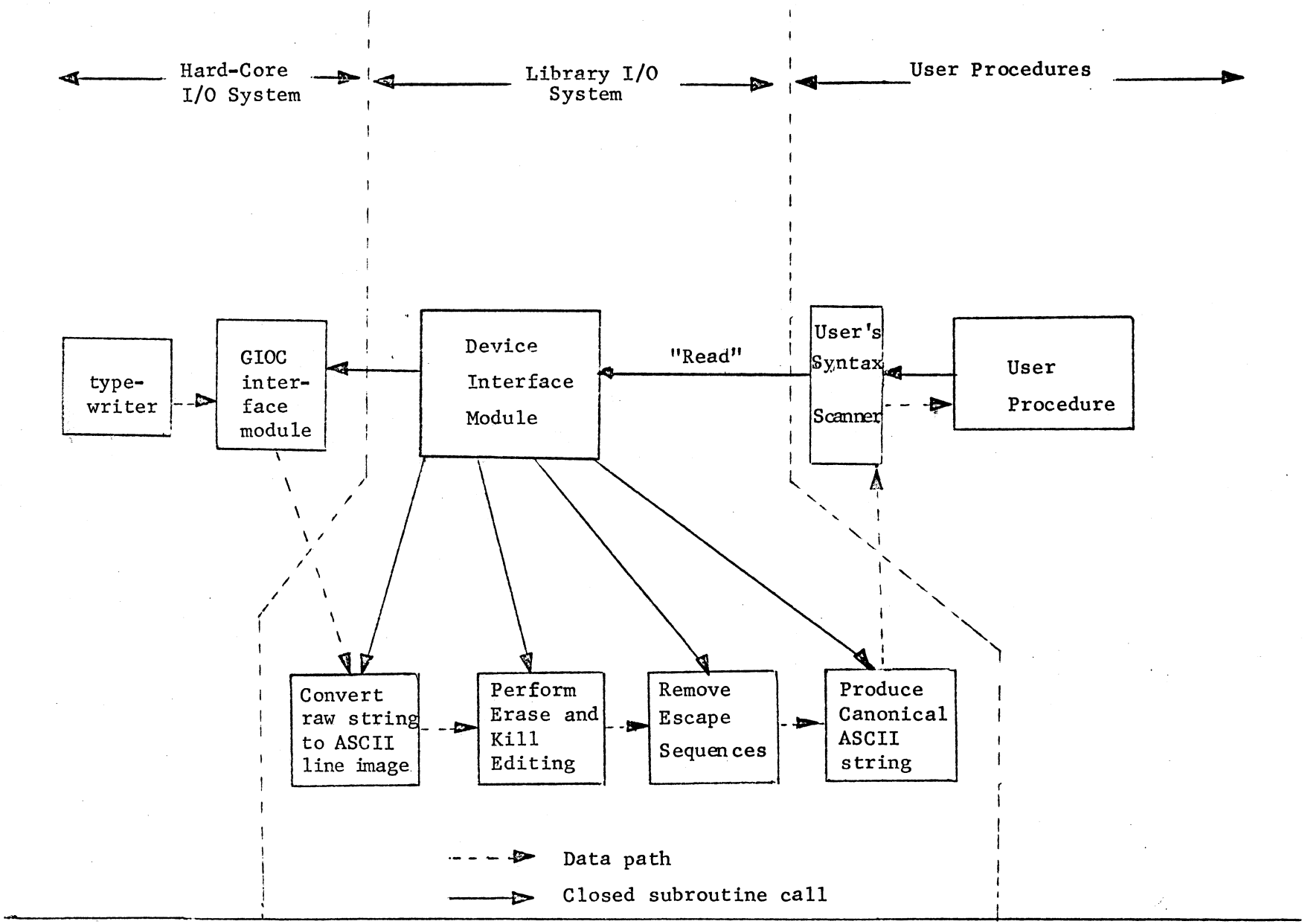

Figure 1 ---- Input character processing