TO: MSPM Distribution
FROM: P. G. Neumann
SUBJECT: BF.2.30, 2.32, 11.03
DATE: 01/10/68

The code conversion complex has undergone a redesign. The new design for typewriters is contained in the accompanying BF.11.03. The attached copy of BF.2.30 only partially represents this redesign. The accompanying BF.2.32 is in spirit still worth publishing, but has not been modified to indicate the change.

The code conversion module has vanished as an interal module, and has been redistributed. Canonicalization is now available as a subroutine for use by the typewriter DSM and other interested modules. Ascii conversion is now embedded in individual DCM's, i.e., for initial MULTICS in the typewriter DCM. Both BF.2.30 and 2.32 will eventually vanish, along with 2.31, and will reappear inside of BF.1.05.

Published: 01/10/68
(Supersedes: BF.10.00, 08/07/67)

Identification

An Overview of Input-Output Code Conversion

D. L. Stone
E. L. Ivie

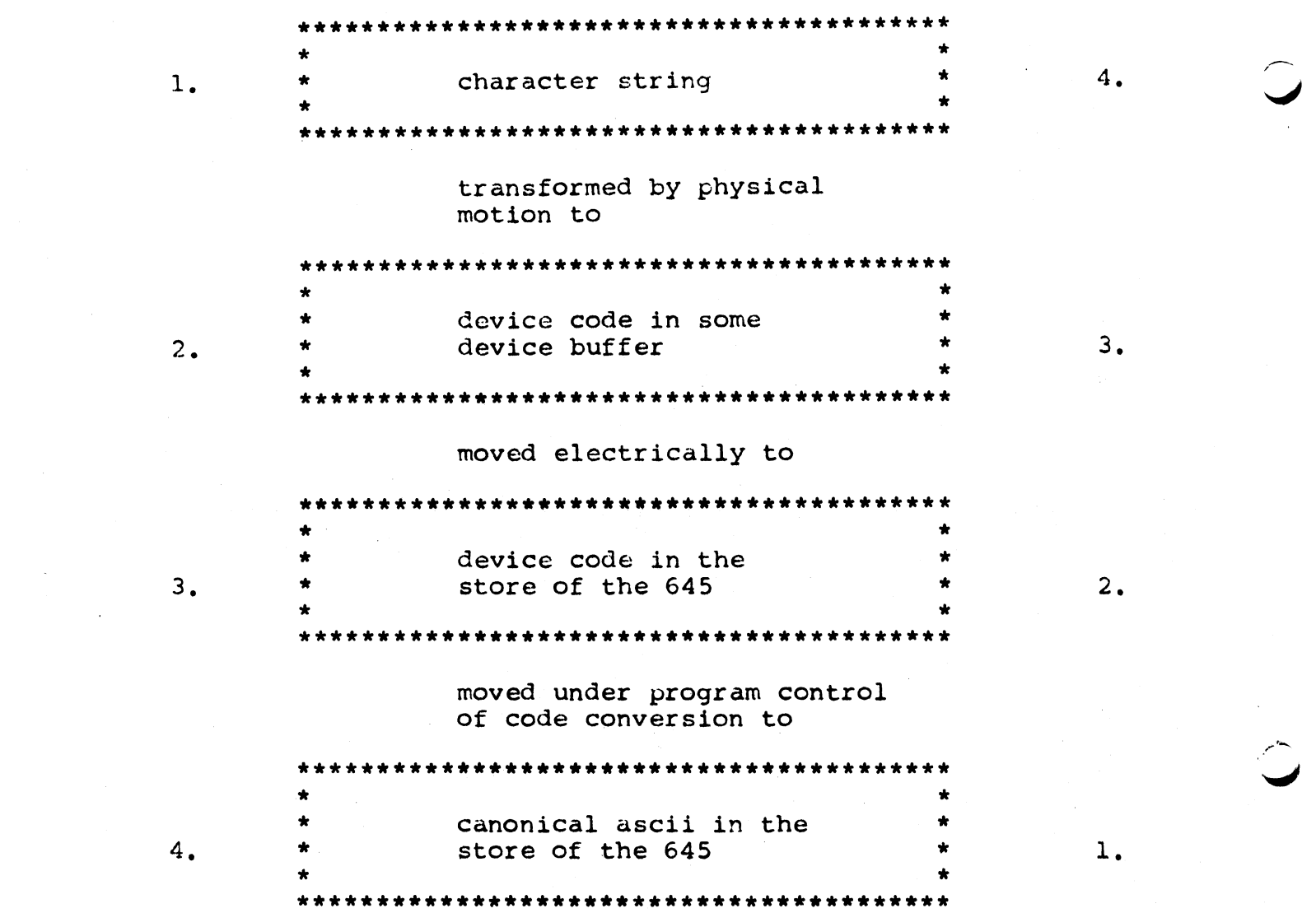
Purpose

BF.2.30 - 32 provide information about the use and implementation of Multics IOS code conversion. Output code conversion is performed by the Device Control Module (DCM). Input code conversion is performed by the DCM and by an inner module called the Canonicalizer (CANON) which is called by the Device Strategy Module (DSM). BF.2.31 describes input code conversion and the Canonicalizer while BF.2.32 describes output code conversion.

General

Code conversion is a necessary concomitant to the use of character oriented input-output under Multics. In concept, a character string is simply a sequence of graphic symbols. Code conversion concerns the ways in which character strings and their Multics representations can be converted from one to the other. For each character string which is input by some device, there correspond many possible internal representations; in Multics only one will appear -- the "canonical" form for that character string. For each canonical representation, there can be assigned a single method of causing the associated character string to be produced by some output device. It is the job of code conversion to produce the appropriate Multics representation in the former case and to produce the proper codes for the output device in the latter. In essence, code conversion is an interpreter between the basic language of Multics and the dialects and foreign languages spoken by the peripheral devices with which Multics converses.

The following outline shows the role played by code conversion in input-output. The flow from top to bottom indicates the transformations which occur during input; the flow from bottom to top, during output:



Although there are actually many more complications to character string I/O, this diagram includes sufficient details for understanding the operation of code conversion. The GIM treats the data between steps (2) and (3). However, its ministrations do not affect the character string in transit; rather, it facilitates its movement by properly directing the actions of the hardware. Any insertions or deletions of data performed by it does not affect the conceptual character string which is being transmitted.

Invocation

Code conversion is automatically performed in any iopath which the IOS can determine to be aimed at a character oriented device. Those devices are:

Typewriters
Card Readers and Punches
Printers
Character CRT displays.

Code conversion once in use is affected by the code mode specified in the attach or changemode call and by the device intended. Using this information, a set of driving tables are selected for use by the Canonicalizer and the DSM. For information on tables see BF.2.31 and BF.2.32.

Functions Performed by the CCM on Output

There are three basic actions which may be performed by a DCM on a character string specified by a write call:

1. Ordering the characters as per device strategy. Reordering the characters is required only for devices which must simulate certain control characters as, for instance, the printer simulating a backspace.

2. Editing the string according to the disposition of the characters specified in the code conversion tables used by the DCM. ~~s/table/table/p~~

3. Converting ascii codes to device codes.

All three of these actions are controlled by the type of device to which the output is directed. The editing function is the only one influenced by the user's code modes.

Per-Call Data Reordering

For the purpose of determining the optimal way to order an output string of characters, we may classify the output devices into two types -- those which must simulate a backspace and those which need not do so. The former type comprises line printers, certain typewriters and other devices which have a carriage return capability but can not backspace. In order to overstrike on the backspace-less devices, additional lines must be printed over the first; hence, these devices must sort their data by depth of overstrike and then by horizontal position as opposed to backspaceable devices which normally print data sorted first on horizontal position and then on overstrike depth. Accordingly, the DCM orders the data passing through it in whichever way the code table indicates is the desired method for the class of device which will receive the data.

Editing

The editing functions are controlled by the code modes specified by the user. In editing the data written by the user, a DCM divides the ascii character set into six conceptual categories. The characters in each category are placed there by the character disposition table specified in the driving tables. The categories are:

1. The character is precisely the bit pattern which should be sent to the device. Clearly, this category is only open to devices which can cope with the ascii set, or at least some part thereof. No conversion is necessary.

2. The character is a graphic in the device character set which is to be printed; or the character is a control character whose function is available on the device (e.g. -- backspace on a 1050 but not half-line feed), and which function is to be performed.

3. The character is to be deleted from the data.

4. The character is to be printed as an escape sequence which defines it (e.g. -- left parenthesis overstruck by minus sign for a left brace on a 1050 with a 938 ball).

5. The character is to be replaced by a blank in the data.

6. The character is a control character which must be simulated. Such a character is backspace on non-backspaceable devices.

As many characters as possible are placed in categories (1), (2) and (6) and all others in category (4). This corresponds to the "normal" code sub-mode. Two other sub-modes are available, if specified by the user's code mode, the "straight" mode, in which no code conversion processing is done, and the "edited" mode, in which all characters normally in category (4) are distributed between categories (3) and (5) depending upon whether they are controls or graphics, respectively. Non-ascii characters remain in category (4). The default tables for the "edited" mode implement this distribution for each device.

Conversion to Device Codes

The final task of output code conversion is to convert the result of ordering and editing the characters into device codes suitable for the specified device. This conversion is accomplished by code tables accessed by the DCM for each device class.

Code Conversion Functions Performed on Input

On input the code modes specified by the user are "raw" and "canonical". The "raw" code mode corresponds to the "straight" code mode on output. No processing of the input character string is done when the mode is "raw".

The functions performed by the IOS when the input mode is "canonical" are described below in the order in which they are performed. The first step is performed by the DCM. Steps 2 through 5 are performed by the Canonicalizer which is called by the DSM.

1. Conversion from Device Code to ASCII

There is a device-code-to-ASCII conversion table for each type of input device available to Multics. These tables map each device character into the ASCII character which most closely resembles it. See Section BC.2.01 for a description of the ASCII character set, the Multics standard control characters, and methods of dealing with non-ASCII characters.

Hardware escape sequences are also converted to the appropriate ASCII characters on this step.

2. Concealed Characters

Occasionally one may wish to type a character on his console which is to have only a local effect and is not to appear in the final character string or to initiate any action in the IOS. An example of where this might be useful is when one has reached the physical end of a line but has not completed the "logical line". In this case he could type a "concealed" newline character by the sequence "(escape-character)C(new-line)". All three of these characters would be deleted from the the input string on this step. See Section BC.2.04 for a description of concealed characters.

One should make special note of the fact that erase and kill characters can be concealed since erase and kill processing is not done until step 4.

3. Canonicalization

The objective of the third processing step is to create an internal "line image" which represents the actual appearance of the line as printed at the console and which is independent of the order in which individual characters are typed. For example, the sequences, "ab(bs)_c", "abc(bs)(bs)_", and "a_(bs)bc", are all converted to the same internal sequence, "ab(bs)_c", since they all appear the same on the printed page at the console. (Here "(bs)" means backspace.)

The internal string is called the "canonical" representation of the typed input string. Canonicalization is accomplished by grouping together characters according to the horizontal print position and vertical character position which they occupy. A description of what constitutes a canonical string is found in Section BC.2.02.

Note that the internal line image is not exactly equivalent to the line printed at the console if concealed characters are present.

4. Erase and Kill Processing

An elementary editing facility has been imbedded into the I/O System in the Canonicalizer. There will, of course, be additional editing capabilities which the user may provide or invoke which operate on the input string after it has been delivered to his calling procedure.

The editing performed by the Canonicalizer makes use of two reserved characters: the erase character and the kill character. The erase character provides the ability to remove from the input string all characters which have been (or are) typed in the horizontal print position in which the erase is found and also all characters in the previous print position. The kill character erases all characters in horizontal print positions to the left of (and including) the print position occupied by the kill character.

There can be only one erase character and one kill character but the selection of which particular characters these are is under the control of the person generating the input string. For example, the erase function is reassigned to the character "x" when the sequence, "(escape-character)Ex", is encountered by the Canonicalizer.

Since erase and kill processing is done after canonicalization all characters typed in the print positions affected will be erased, and not just those typed prior to the typing of the erase or kill character. Vertical motion and ribbon shifts are, however, not erased. See Section BC.2.03 and BF.2.31 for further documentation.

5. Interpretation of Escape Conventions

Most of the input devices which are currently planned for use with Multics do not generate the full ASCII character set. For these "deficient" devices escape sequences have been defined (see Section BC.2.04) which allow one to represent a character which is not available on the device by a sequence of two or more characters which are available of the device.

Indeed the escape capability is not limited to those characters which are not available on a given device. An octal escape

sequence has been defined which provides one with an alternative representation of every ASCII character.

Information on what escape sequences are valid for each device is stored in a special tree structure in the input code conversion driving table for that device. Section BF.2.31 describes this structure.

6. Recanonicalization

The processing of escape sequences in Step 5 may destroy the canonical order of the string that was established in Step 3. This occurs, for example, if a horizontal tab is generated by an escape sequence. This final step is necessary to insure that the string delivered by the Canonicalizer to the calling procedure is really canonical.