Published: 08/07/67

Identification

An Overview of Input-Output Code Conversion
D. L. Stone, E. L. Ivie

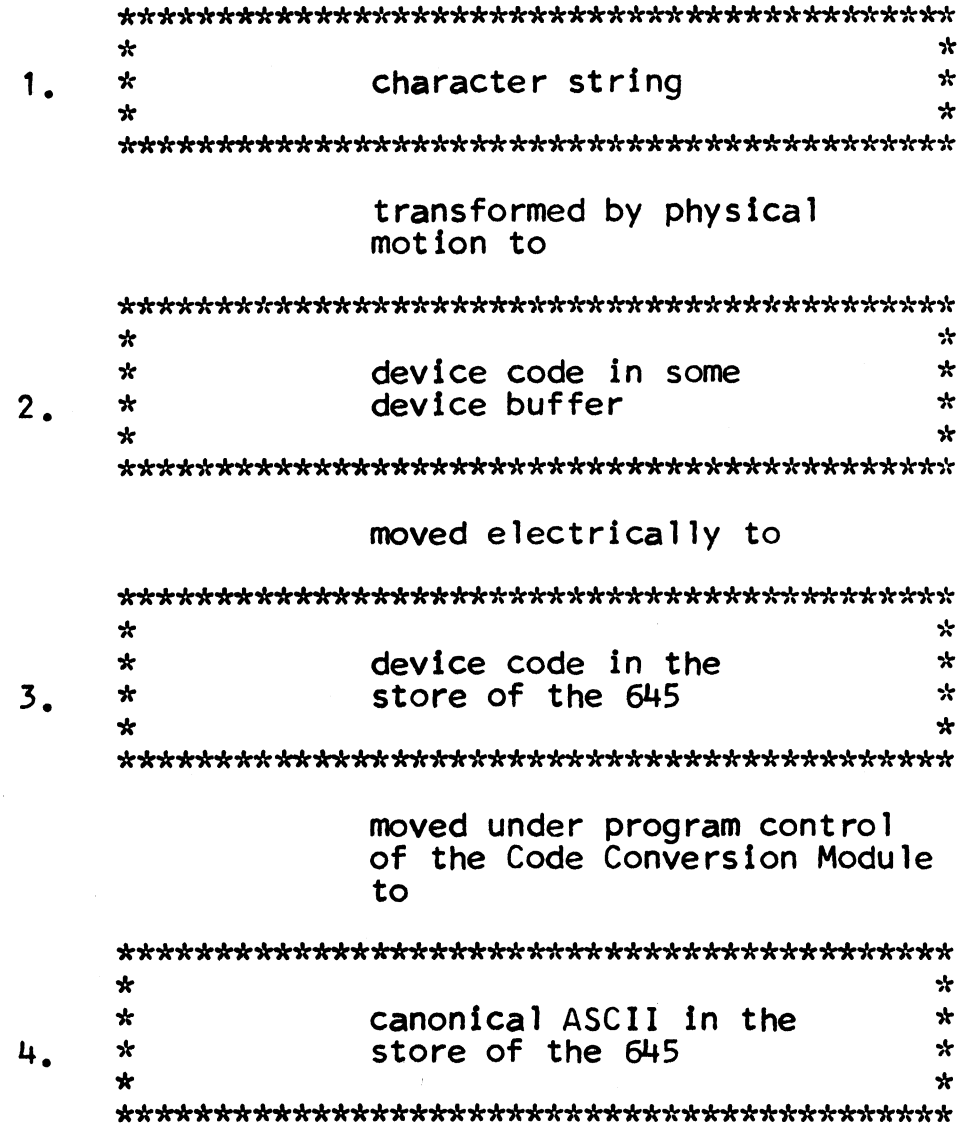
Purpose

BF.10.00 provides information about the use and implementation of Multics IOS code conversion. This conversion is accomplished by the Code Conversion Module (CCM), an IOS outer module located between the user and the Device Interface Module (DIM). The input and output functions of the CCM are handled separately and driven from separate tables, although the two parts constitute a single outer module. Section BF.10.01 describes the implementation of input code conversion; section BF.10.02 describes the implementation of output code conversion; and section BF.10.03 describes the means by which the tables used to drive the CCM on both input and output can be created, edited and replaced.

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 (see BC.2.00). 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 the code conversion module 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, the CCM 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 the CCM 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 the CCM. Both the GIM and the DIM treat the data between steps (2) and (3). However, their ministrations do not affect the character string in transit; rather, they facilitate its movement by properly directing the actions of the hardware. Any insertions or deletions of data performed by them do not affect the conceptual character string which is being transmitted.

Invocation

The CCM is automatically included 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.

In addition, the CCM may be included by the user in any iopath which satisfies the restraints of the CCM on data format. Typically, the user would splice the CCM in front of a Device Strategy Module (DSM).

The CCM's operation once attached 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 CCM. For information on tables see BF.10.01, BF.10.02, and BF.10.03.

Outer Call Interpretation

The CCM handles most of those IOS outer calls which it explicitly accepts in standard fashion. Below are listed the exceptions and areas of interest in the CCM's actions in response to outer calls.

Attachment and Initialization

The calls relating to attachment and initialization handled by the CCM are attach, detach, changemode, divert and revert. With the exception that the attach call is not forwarded, these calls are handled normally by the CCM. Divert and revert are passed on untouched. CCM initialization only involves obtaining a set of driving tables and using the information therein to prepare for future calls. Precise specifications of the information in the tables is given in BF.10.01 and BF.10.02, and the method in which the tables are specified is given in BF.10.03.

Element Size

The two outer calls which refer to the element size are getsize and setsize, (see BF.1.08). The restrictions on allowable element size are that only nine bit characters will be allowed -- these being interpreted in standard

Multics fashion. The exception to this rule occurs when the "straight" output code mode or "raw" input mode is specified in the mode argument of an attach or changemode call (see BF.1.02). In this case, six-bit or nine-bit will be allowed. Six-bit and nine-bit are the only element sizes transferred by the GIOC to character-oriented devices.

Synchronization Calls

The module responsible for synchronization management in the IOS is the device strategy module (DMS). The DSM controls both the amount of read-ahead and the amount of write-behind that takes place. The CCM normally occurs in the iopath between the user and the DSM. It must therefore forward to the DSM all calls dealing with synchronization. These include: readsync, writesync, resetread, resetwrite, worksync, iowait, and abort. (See Section BF.1.04 for a description of these calls.)

The only synchronization calls that initiate special action in the CCM as they pass through are resetread, abort, and iowait. There is a certain amount of unavoidable read-ahead that may take place in the CCM. This is due to the fact that no canonicalization or erase-kill processing can be performed until the appropriate delimiter has been found. Thus, the CCM may process a whole line at a time but only return to the user that part of the line requested by the user. The remaining portion of the line is unavoidably read-ahead data. It is kept in a buffer and used to satisfy future read requests or is deleted if a resetread or abort call is issued to the CCM.

The CCM may also have to transform the oldstatus argument in the abort and iowait calls it receives to the corresponding transaction pointers in the calls it has issued to the DSM.

Delimiters and Pointers

The CCM always stores the list(s) specified by a setdelim call and usually passes the call on. For a getdelim call the CCM sets the appropriate arguments and returns to the caller. There is no need for passing this call on. (See Section BF.10.01 for further discussion on these calls.)

Seek and tell calls issued by a user to the IOS are in terms of the number of items which he reads or writes. In general the CCM will be the only module which will

be able to keep such counts, because code conversion is not a one-to-one process. Therefore, the CCM sets the offset argument of a tell call based on the number of ASCII characters which have been translated and returns to the caller.

The CCM rejects the seek call because it is not in general possible to transform the ASCII-oriented seek into a meaningful call to lower modules in terms of the elements with which they deal. Further, when the CCM is sending blocks of calls to lower modules, the meaning of the seek changes in a way which prevents any useful transformation of the user's seek.

Functions Performed by the CCM on Output

There are three basic actions performed by the CCM 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 character disposition tables specified. The character disposition tables are explained in BF.10.02.

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 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 cannot backspace. In order to overstrike on the backspaceless 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 CCM 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 of the CCM are controlled by the code modes specified by the user. In editing the data written by the user, the CCM divides the ASCII character set into six 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 the shortest 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.

The default tables used by the CCM place as many characters as possible 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 all characters are treated as if they were in category (1), 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. For more detailed control of the editing capabilities of the CCM, the user must create his own output code conversion tables and specify that they should be used. See BF.10.03 for details.

Conversion to Device Codes

The CCM's final task on output is to convert the result of ordering and editing the characters into device codes suitable for the specified device. This conversion is accomplished by the code tables associated with each device class and obtained from the output code conversion tables in the driving tables. These output code conversion tables are not alterable by the CCM. The specifications contain in these tables enable the insertion of case shift characters and other controls specific to each device. No timing considerations or tabbing strategies are incorporated; these functions are performed by the DCM's.

Buffering

On output the CCM passes the data which it receives from a write call by means of writerec calls. The number of records (print lines) passed in each writerec call is a variable bound by the driving tables selected for a particular attachment of the CCM. The CCM issues as many writerec calls as are necessary to the DSM, but always processes all of the data before returning control to its caller. The CCM always retains its processed data until physical completion of the transactions involving it has been signalled from the modules below.

Functions Performed on Reading

The functions performed by the CCM on input are all associated with the canonical code conversion mode. They are briefly described below in the order in which they are performed by the CCM.

1. Conversion from Device Code to ASCII

There is a device code-to-ASCII table for each type of input device available to Multics. The general layout of the tables is given in BF.10.01. The precise contents of these tables are given in the Sections dealing with each device in question.

In general 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 charac ers, and methods of dealing with non-ASCII characters.

2. Hidden 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 "hidden" newline character by the sequence "(escape-character)C(new-line)". All three of these characters would be deleted from the input string on this step.

Note that erase and kill characters cannot be hidden. See Section BC.2.02 for a description of hidden characters.

3. Print Position Alignment

The third step performed by the CCM is the interpretation of the character string as a printed line. Characters are grouped together according to the horizontal print position and vertical character position which they occupy. The objective of this step is to create an internal "line image" which represents the actual appearance of the line as printed at the console and is independent of the order in which the individual characters were typed. Thus, the sequence "abc(bs)(bs)_", would be made equivalent to the sequence, "ab(bs)_c". (Here "(bs)" means backspace.) Note that the internal line image is not exactly equivalent to the line printed at the console if hidden characters are present. Section BC.2.00 provides further information on this step.

4. Erase and Kill Processing

This step makes use of two reserved characters: the erase character and the kill character. The erase character provides the ability to erase all characters which have been typed in the horizontal print position in which the erase is found and also all characters in the previous print position. A kill character erases all characters in horizontal print positions to the left of (and including) the print position occupied by the kill character. Vertical motion and ribbon shifts are not erased. See Section BC.2.03 for a more complete explanation. At the same time the CCM processes erase and kill characters it interprets those escape sequences which change the erase or kill character. These sequences are described in Section BC.2.04.

5. Interpretation of MULTICS Escape Conventions

The next function performed by the CCM is to replace all valid escape sequences with their equivalent character. What constitutes a valid sequence of each type of input device is described in BC.2.04. The basic purpose for having escape sequences is to allow the user to generate the full ASCII character set even on deficient input devices.

Universal octal escape sequences can be used on any input device. Any 9-bit code can be inserted by this mechanism. In addition to octal escape sequences there are specialized escape sequences for each device which are designed to be easy to remember and which insert certain unavailable characters.

6. Canonicalization

Canonicalization arranges the characters within a line in a fixed order so that subsequent processing and interpretation does not require checking for alternate equivalent orders. Characters within a line are first ordered from left to right. Characters within a given horizontal print position are ordered from top to bottom. Overstruck characters are ordered according to ascending code number and interleaved with backspaces. A full description of canonicalization is found in Section BC.2.02.