Published: 01/10/68
(Supersedes: BF.10.01, 08/14/67)

Identification

Output Code Conversion

D. L. Stone

Purpose

This section describes the conversion of ascii character strings to strings of device codes by the output half of the Code Conversion Module (CCM). It is intended to be useful to those people interested in the specification of output code conversion tables or in the workings of the CCM code which uses them. The call interface of the CCM is given in BF.2.30.

General

Output code conversion is necessary for the following devices supported by the Multics IOS:

typewriters	IBM 1050, 2741
	TTY M35, M37
line printers	GE PRT202
	IBM 1403-n1, 1403-2
card punches	GE CPZ200
	IBM 1442-5, 2520-a1

The peculiarities of each of these devices are reflected by a set of driving tables for each one. During the processing of output data for one of these devices, an output driving table is selected to accord with the user's choice of code conversion mode; that table will be used to initialize the CCM and, during each output call, to guide it in translating the data. In order to change the way in which the output code conversion is performed, a new table must be selected (for the same or a different device). The makeup of the tables is detailed below and the selection and means of construction are given in BF.2.33.

The driving tables are the heart of the CCM's function. Each output driving table consists of three types of information: 1) initialization information, including certain printing characteristics and hardware capabilities, 2) the character table, which specifies the desired treatment of each character which may be processed by the CCM, and 3) the escape arrays, which contain sequences of characters which are to replace certain characters as specified by the character table. All

these categories are described at great length below.

Output Code Conversion Tables

The declaration of an output driving table for the CCM is given below. For all switches, a "1"b indicates an affirmative answer to the question posed by the switch.

```
dcl 1 outccm_table based (p),
2 relps,                          /*relps to vble length data*/
 3 (ctable_relp,                  /*to "character_table"*/
    ascii_relp,                   /*to "escape_strings"*/
    device_relp,                  /*to "device_escape_strings"*/
    modes_relp,                   /*to "hardware_modes" array*/
    ctl_modes_relp) bit (18),     /*to "ctl_modes*/
2 sizes,
 3 (ctable,
    ascii,
    device,
    modes,
    ctl_modes) bit (18),
2 initialization,
 3 num_overstrikes bit (9),       /*max num simulated backspaces
                                    per print position*/
 3 software_esc bit (9),          /*in ascii*/
 3 prt202 bit (1),                /*"1"b = each line begins in upper case*/
 3 pad bit (17),
2 character_table (0:511),
 3 action bit (6),                /*what to do with each character*/
 3 ls_count bit(3),               /*how does this char affect the
                                    print position;  0 = backspace,
                                    1 = no change; 2 = graphic*/
 3 device_code bit (12),          /*left justified*/
 3 esc_length bit ( 6),           /*index into escape tables*/
 3 esc_offset bit (9),
2 escape_strings (N),
 3 escapes char (1),
2 device_escape_strings (M),
 3 escapes bit (element_size),
2 hardware_modes (num_hardware_modes),
 3 status bit (1),
 3 pad bit (2),
 3 mode_index bit (6),            /*correlated bit index in DSB
                                    hardware status string*/
 3 esc_length bit (9),
 3 esc_offset bit (18),           /*indices into "device_escape_strings"*/
2 ctl_modes (num_ctl_modes),
 3 fake bit (36);
```

Since all of the conversion functions of the CCM are controlled by the driving tables, a detailed description of the output tables will elucidate all functions of the output part of the CCM. Since the character table controls the flow, we begin with that.

The Character Table

The character table is an array of information about the way in which each of the 512 nine-bit characters is to be treated. As ascii data is processed by the CCM, the binary equivalent of the character code is used as an index into the character table. The six bits of information in the "action" entry are used as an index into a table which specifies the appropriate action by the CCM for the character. The possible actions are specified below with their octal equivalents.

octal	meaning
0	Append the actual bit configuration of this character to the output buffer; useful for devices which accept ascii or some subset thereof.
1	Append the bits given by the "device_code" entry for this character to the output buffer.
2	Mode change; this character may cause a change in the physical state of the device which will affect its printing (e.g. - red-ribbon-shift or case-shift). Use the "esc_offset" field to get the index which corresponds to this mode and change the status of the mode in the current device status block. The "esc_length" field is "XXX0"b for set and "XXX1"b is interpreted as reset. Having set the status, use the appropriate "hardware_modes" pointer and length pair to find the device codes to be placed in the output buffer. See discussion of hardware modes later in this section.
3	This character requires that a certain mode be in a certain state for proper printing. "esc_offset" , interpreted as a fixed binary number, gives the mode index and "esc_length" is "XXX0"b if mode should be set ("XXX1"b if mode should be reset). If necessary, add the device codes to change the hardware mode. Append the "device_code" to the output buffer after verifying this mode.
4	Escape this character by means of the standard octal software escape sequence specified in BC.2.04.

5 Escape this character using the ascii character string specified by the "escape_length" and "escape_offset" index into the "escape_strings".

6 Escape this character using the device code string in "device_escape_strings" specified by "escape_length" and "escape_offset".

7 Ignore this character. The output will appear just as it would have if this character had not been in the character string.

10 Simulate a backspace. This action is meant for those devices which can not backspace but possess a carriage-return capability (e.g. - the PRT202). Backspaces are simulated by using two print lines with a carriage return between them.

11 This character will end the print line (NL). Apply the appropriate sequence to the output buffer using the "escape_length" and "escape_offset" entries as an index into "device_escape_strings".

12 As in category 11 (NP). (new page).

13 Carriage-return; reset line_size and use "escape_length" and "escape_offset" as above, if bit 7 of "device_code" is 0. Otherwise, simulate a carriage-return as if it were the appropriate number of backspaces to get to the beginning of the line. This category takes into account those devices which can overprint but do not respond to a single carriage return character. (PRT202).

14 Vertical tab; increment counter appropriately, then use "device_escape_strings" as for category 11.

15 Horizontal tab; see category 14.

16 Half-line feed (HLR); increment counter, then use "device_escape_strings" as for category 11.

17 Half-line feed (HLF); see category 16.

20 Undefined character; this category is treated as category 4 except that the "undefined character" bit is set in cstatus upon return.

Each character processed by the CCM is treated according to the category in which it is placed by the character_table in the driving table specified. The creator of the driving table can tailor the processing of the CCM to suit any output device and/or personal fancy since this mechanism allows a completely general

character-by-character transliteration.

To allow faster processing by the CCM, each character string is assumed to be in a generalized canonical form. A precise definition of the criterion for canonicalization is given later in this section.

The entry "ls_count" in the character table can take on the values:
octal 0 for backspace
octal 1 for non-printing characters (as rrs)
octal 2 for graphics
It is used to reflect the change in horizontal print position caused by a character.

The "device_code" entry in the character_table contains the (six, nine or twelve-bit) device code which corresponds to the ascii character on the intended device (the GIOC only transmits six- and nine-bit elements).

The "escape_length" and "escape_offset" entries specify a string of characters in either the ascii "escape_strings" array or the "device_escape_strings" array. The intended string of characters is of length escape_length and begins at the escape_offsetth character in escape_strings. Since the ascii strings are fed back into the "character_table", no ascii characters requiring escapes are allowed in the escape strings.

Escape Arrays

The "escape_strings" array in the driving table is simply a packed array of characters representing ascii escape strings. The strings are in no particular order.

The "device_escape_strings" array, similarly, is a packed array of fields which represent device code sequences. Since the "ls_oount" associated with such a string can only reflect a change of one print position, it is not advisable to use these strings to introduce sequences which cause a change of more than one print position (other than those which cause predictable actions as NL, NP, CR).

The two escape arrays are intended for different purposes. The ascii array should only be used when the escape sequence is ambiguous in device code; that is, when case-shifts or other device status can affect the interpretation of the sequence. The PRT202 is an example of such a device. The device code array can be used for all other escape sequences and for special sequences which take the place of a single character -as with new-line and carriage-return on the PRT202.

Hardware Modes

In order to implement character table "action" entry 3. and the "set_status" call, the CCM needs information on all of the character-settable device hardware modes. The initial status of the modes is assumed from the default DSB in the driving table segment header. In the DSB, the thirty-five bit "hardware_status" string provides the setting of each of (a potential) thirty-five modes. A "0"b in the nth bit from the left is taken to mean that the nth mode is "set" and "1"b, "reset". Whenever it becomes necessary for the CCM to change the setting of a hardware mode, it uses the hardware_modes array as follows:

1. The "mode_index" entry (regarded as a fixed binary number) specifies which hardware mode this entry refers to.

2. The "status" entry specifies whether the indices "set" or "reset" the mode.

3. "esc_length and "esc_offset" specify a bitstring in "device_escape_strings" which will cause the specified hardware mode to become "set" or "reset" according to the "status" entry.

This use of the "hardware_modes" array implies that there are two entries for each useful mode -- one which specifies the "setting" sequence and one which specifies the "resetting" sequence.

An example of the use of hardware modes for a 1050 typewriter follows:

The hardware_modes array has four entries:

mode 1 -- interpreted as ribbon-shift
"set" equated to black
"reset" equated to red
hardware_modes(1).status = "0"b
hardware_modes(1).mode_index = "000001"b;
esc_ (length and offset) would point to an entry in the "device_escape_strings" array containing the six-bit character sequence which makes a 1050 shift to black. Entry two would point to the red-shift character sequence.

The action entry in the character table for BRS and BRS (ascii 016 and 017) would be 2; "esc_offset"would be 1; "esc_length" would be "0001"b for 016 and "0000"b for 017. No other action entries would specify mode 1.

mode 2 -- interpreted as case-shift
set equated tc lower

reset to upper
hardware_modes(3).status = "0"b
hardware_modes(3).mode_index = "000010"b
The set and reset sequences would be the 1050 six-bit codes to shift to lower and to upper case, respectively. No ascii character would have a character table entry specifying this mode index (2) from an "action" of 2 (set or reset mode); but any graphic which prints only in one case-shift mode (e.g. -- "A") would have an action entry of 3 and specify hardware mode 2 (and for "A" , "esc_length" would be "0001"b to indicate that reset or upper case was to be in effect before the device code could be appended to the output buffer).

mode 3 -- interpreted as line feed mode
set equated to single line
reset to double line
status = "0"b
No ascii characters currently defined deal with this mode. The DSM can cause the CCM to change the hardware setting of this or any other mode by means of the "set_status" call (for which see BF.2.30).

Ctl_modes

The "ctl_modes" array will be used for characters which have peculiar interpretations and hardware effects (such as the ascii "escape" character on the new model 37 teletype). It will be clarified when the Multics policy towards such characters is defined.

Miscellany

"num_overstrikes" specifies the maximum number of overstrike lines to be created during backspace simulation.

"software_esc" gives the ascii character which is to precede all octal escape sequences on output.

"prt202" indicates that the case-shift mode (number 2) is always to be put to upper ("reset") at the beginning of a line.

Implementation Details

One of the parts of the CCM implementation which requires further explanation is escaping. The replacement of a given ascii character by a string of ascii characters is accomplished by temporarily changing the input pointer and index so that they

point to a buffer containing the desired ascii characters. The current pointer and index are saved and restored when the escape buffer has been exhausted. The count of characters left in the escape buffer is put into the normal comparison for end of input buffer. Since this mechanism is not recursively implemented, no further characters specifying ascii escape should be put into an ascii escape string.

Replacement by a string of device code characters is intended for use by DIM writers whose devices require special sequences for certain characters (carriage-return, new-line). The specified string of output characters is placed in the current output buffer directly.

Another part of the CCM implementation which needs explication is backspace simulation. This feature of the CCM was included to deal with the PRT202 line printer which has no ability to handle overstrikes in its hardware. The feature is potentially useful for any device which has an overprinting capability but no backspace. A backspace is simulated by creating a new output line and padding it with blanks until the desired print position is reached. The character to be overstruck is then put into position. When the end of line is reached, the CCM returns all of the lines created in this way with a "carriage-return" equivalent except the last one, which is issued with a "new-line" equivalent. Clearly, any output code conversion table which specifies backspace simulation must also include entries for both new-line and carriage-return, since both those entries will be accessed by the CCM directly.

The backspace simulation is implemented by changing the output buffer pointer and length to a new output buffer which is first padded to reach the proper print position. A list of output buffers for each line is kept -- the maximum number of such buffers is controlled by "num_overstrikes" in the table. For each buffer a pointer, the current size in output elements and a device status block are maintained. The DSB is set to the status assumed at the beginning of the line so that appropriate changes of status can be made between lines , if necessary. When a backspace is encountered in the data processed by the CCM, a check for contiguous backspaces is made. A cluster of backspaces generates a count which, subtracted from the print position of the current output buffer, yields the desired print position for the next character. The print positions of all currently allocated output buffers are checked to see if any is less than the desired one. If one such is found, that buffer becomes the current one, spaces are inserted to adjust the print position if necessary, and processing continues. If no currently allocated output buffer has a print position less than the desired one, then a new buffer is allocated, provided that the limit has not been reached. If the limit is met, then the "non-canonical" bit is set in cstatus and a return is made. Previously translated data is returned as usual. The DSM can determine the beginning of the non-canonical data from the "output_tbe", which will have

line pointers only to the translated data.