TO: MSPM Distribution
FROM: Charles Garman
SUBJ: BY.3.01
DATE: 06/23/67

The attached MSPM section describes a simple scheme to accomplish character string I/O for segments; it was originally a part of the edit command, but has been re-worked and documented separately on the basis of general utility.

Published: 06/23/67

Identification

Character String I/O for Segments
read_cs, write_cs
Charles Garman

Purpose

Read_cs and write_cs provide a set of calls which allow simplified movement of PL/I character strings to and from segments, using based character strings to access successive lines of the segments.

Both routines are "single-level", that is, they can be used only for reading from (writing into) one segment at a time (but each is independent of the other); multiple-level reading or writing by one routine, while equally simple, requires a different calling sequence, and is beyond the scope of this document.

Usage and Implementation

In order to simplify the calling sequences of the basic "transmission" references, the procedures keep certain variables in PL/I "internal static" storage:

1) a pointer to a region from (to) which the characters are moved;
2) the number of the last (terminal) character to be read from (written into) the region; and
3) the number of the current character, that is, of the last character involved in a transmission call.

Entries are provided to initialize any or all of these variables in either procedure, and to obtain their current values.

Since these procedures are used by the edit command, the variables are grown (with the current implementation of EPL internal static storage) in the segment edit_stat_ (through the use of the options clause in the procedure statement), but this should not hinder the usage of the procedures in other contexts.

To initialize for reading (writing),

```
dcl ch_ptr ptr,
   (n_chars, cur_char_no) fixed bin(17);
```

```
call {read_cs  }$setup (ch_ptr, n_chars, cur_char_no);
     {write_cs }
```

Ch_ptr is a pointer to a region of storage from (into) which read_cs (write_cs) is to obtain (place) its characters. On "transmission" calls (see below), characters will be moved from (into) the region, starting at character cur_char_no + 1, but never beyond the character n_chars from the start of the region (the first character of the region is assumed to be in the first character position [left-most 9 bits] of the word addressed by ch_ptr).

The region pointed to by ch_ptr may be the base of an entire segment (e.g., ch_ptr may have been obtained by a call to the Segment Management Module [BD.3]), or it may be any smaller piece within a segment (e.g., a portion of the stack).

In the call to read_cs$setup, n_chars may have been obtained by a call to get_count$chars (BY.2.07) for the information stored in the branch for the segment, or it may have been derived from other data; similarly, in the call to write_cs$setup, if ch_ptr points to the base of the segment, and the maximum length of the segment has been set to 2^{18} words (256 1024-word blocks), a maximum value of 1048576 may be used for n_chars.

The calls may be used to prevent initialization of certain variables by the proper selection of values for the parameters:

parameter:	not initialized if value is:
ch_ptr	null pointer (BB.5.03)
n_chars	< 0
cur_char_no	< 0

To initialize for reading from (writing into) the beginning of the region, the value of cur_char_no should be zero.

Once the preliminaries have been attended to, "lines" may be moved from (into) the region as follows:

```
dcl text char(K),      /* K is determined by the caller's
                          procedure. */
    n fixed bin(17),
    status_bits bit(4);
```

```
call {read_cs  } (text, n, status_bits);
     {write_cs }
```

Text is the character string into which read_cs (from which write_cs) is to move its data; n is the number of characters that were read from the region (the number of characters to be written); status_bits provides information to the caller about the success or failure of the intended transmission. For write_cs, if n is < 0, the length of text (K) will be used in its stead. The procedures described in BY.10.02 and BY.10.03 are used, allowing text to be a varying character string if the user so desires.

In the normal case, read_cs moves characters from the region addressed by ch_ptr into text, from the current character to and including the next <NL> character, and then adjusts its notion of the location of the current character. Likewise, write_cs appends the contents of text behind the current character of the region, and adjusts the value of the current character. <NL> characters will not be inserted by write_cs; the caller must make his own provisions for them in text.

For exceptional conditions, however, certain alternate actions occur, and these are reflected in the various bits returned in status_bits: (the table below summarizes the text which follows)

Bit Number	Meaning if "0"b	Meaning if "1"b
1	call accepted	call rejected
2	number of characters remaining in region is > 0	number of characters remaining in region is ≤ 0
3	read_cs: string to be moved fitted into text	entire string would not fit in text
	write_cs: value of n was ≤ length of text	n was > length of text, only length(text) characters were moved
4	read_cs: <NL> character found within region	No <NL> character left in region.
	write_cs: always "0"b	---

If the call was rejected, bit 1 is "1"b. For read_cs, this will occur only if all the characters in the region had been removed before the call which was rejected. For write_cs, the desired number of characters from text could not be moved into the region without exceeding the value of n_chars set in the setup call.

Bit 2 will be set if the number of the current character equals or exceeds n_chars as defined by a $setup call; it may be set by any call if the transmission involved the n_chars-th character. (It also occurs concurrently with the "reject" bit in a call to read_cs.)

The interpretation of bit 3 differs slightly between read_cs and write_cs: for reading, since n is a return argument, it is set if the number of characters which could have been moved exceeded the length of text; the characters which would have been moved will be moved on succeeding calls until the end of the line is reached. For writing, it is set if n exceeded the length of text; only length(text) characters were moved into the region. (Note, if n was < 0, this bit will not have been set.)

For read_cs only, bit 4 is normally "0"b, but if a <NL> character could not be found within the remainder of the region (from the current character to n_chars), it will be set to "1"b; its setting is basically independent of bit 3, and means only that the last character in the region (character number n_chars) is not a <NL> character.

To inquire about the current values known to the procedures,

```
call {read_cs  } $fetch (ch_ptr, n_chars, cur_char_no);
     {write_cs }

        /* Declarations as for setup entries. */
```

The fetch entry of each procedure places the current values of the static variables into the variables passed in the call: ch_ptr is the pointer last passed in a call to the setup entry; its value is the null pointer if the corresponding setup entry has not been called by any procedure in the process; n_chars, similarly, is the value last passed in a setup call, or 0; and cur_char_no is the number of the character (within the region) last referred to by a "transmission" call (as above), or the same as the previous cur_char_no, or 0.

After a call to write_cs$fetch, the value of cur_char_no may be used as an argument to set_count$chars, BY.2.07, to save this value in the segment hierarchy for future reference.