MCR10174
Fix Padding Amount in mcs_echo_neg.incl.pl1
Revision 1.0

Eric Swenson

September 10, 2026

1 Problem Description

Ticket https://multics-trac.swenson.org/ticket/138 describes an issue with the `pad` field in the structure `echo_neg_data` in the include segment `mcs_echo_neg.incl.pl1`. Apparently a new version of the structure was added in 1986, and `break` field was changed in size, but the `pad` field was not correctly adjusted when that change was made.

Due to word alignment done by the PL/1 compiler, making the pad correct does not actually cause any changes to the compiled code that uses this include file. So this change is really mostly to "make things look right" and ease the pain of developers attempting to parse this data structure in their heads.

The structure is shown below:

```
dcl  1 echo_neg_data       based (echo_neg_datap) aligned,
                                                  /* Echo negotiation data */
       2 version           fixed bin,
       2 break             (0:255) bit (1) unaligned,
                                                  /* Break table, 1 = break */
       2 pad               bit (14) unaligned,
       2 rubout_trigger_chars (2) unaligned,      /* Characters that cause rubout
           action */
         3 char            char (1) unaligned,
       2 rubout_sequence_length
                           fixed bin (4) unsigned unaligned,
                                                  /* Length of rubout sequence, output
                                                      */
       2 rubout_pad_count  fixed bin (4) unsigned unaligned,
                                                  /* Count of pads needed */
       2 buffer_rubouts    bit (1) unaligned,     /* 1 = put rubouts and rubbed out in
           buffer */
       2 rubout_sequence   char (12) unaligned;   /* Actual rubout sequence */
```

Since the structure is aligned, the compiler places the `version` field at offset 0. And the `break` field

begins at bit 0 of word 1 and extends to bit 3 at offset 8. (256 bits is 7 words plus 4 bits). This means that the `pad` field starts at offset 8, bit 4, and because it is declared as `bit (7)`, extends to bit 10.

`rubout_trigger_chars` is of data type `char`, so the compiler forces alignment to the next character boundary, at bit 18.

This means that 7 bits are *unaccounted for*. The correct declaration of `pad` should have been `bit (14) unaligned`. Declaring it such mirrors the actual layout generated by the compiler.

2 Proposed Fix

The proposed fix is to change the declaration of `pad` from `bit (7) unaligned` to `bit (14) unaligned`.

As mentioned previously, the change will not impact any code that uses the structure. In fact, as part of testing this fixed, PL/1 listings of each of the programs that reference this include segment were made before and after the change, and then compared.

No differences (other than the listing's showing the new declaration) were seen. All offsets in the programs were identical to those prior to the change.

The old and proposed new include segment comparison can be seen below:

```
cpa >ldd>incl>mcs_echo_neg.incl.pl1 ==

A27                 2 pad               bit (7) unaligned,
Changed by B to:
B27                 2 pad               bit (14) unaligned,

Comparison finished: 1 difference, 2 lines.
```

As part of installing this change, all programs that reference this include segment will be recompiled, although this is not strictly necessary in this case, as their code will not change at all.

Because several programs that reference this include segment are in hardcore, a new hardcore tape will be generated and tested, for completeness.

3 Documentation

No changes to the documentation are required as none of this is documented now.

4 Test

All programs referencing this include segment:

```
pcref mcs_echo_neg.incl.pl1
References to mcs_echo_neg.incl.pl1: (07/10/86 2015.0 pdt Thu)
   e_pl1_.pl1, tc_.pl1, tc_input.pl1, tty_index.pl1, ws_tty_main_.pl1
```

will be recompiled as part of installation, and a new hardcore will be generated and tested.

Furthermore, already done was a comparison of a compiler listing of each of the above, before and after the update of the include segment. These listings were compared manually. An example comparison is shown below.

```
cpa before_fix>tty_index.list ==

A4                Compiled on: 09/10/26 1435.4 pdt Thu
Changed by B to:
B4                Compiled on: 09/06/26 2144.3 pdt Sun


A3759     17    27            2 pad                bit (7) unaligned,
Changed by B to:
B3759     17    27            2 pad                bit (14) unaligned,


A4120    2490       17    07/10/86 2015.0 mcs_echo_neg.incl.pl1
    >ldd>include>mcs_echo_neg.incl.pl1
Changed by B to:
B4120    2490       17    09/06/26 2144.3 mcs_echo_neg.incl.pl1
    >user_dir_dir>SysAdmin>Swenson>work>tickets>138>mcs
\c_echo_neg.incl.pl1


A5640    Length    15556  13154      156       1145    1041          2
Changed by B to:
B5640    Length    15566  13154      156       1156    1041          2

Comparison finished: 4 differences, 8 lines.
```

It can be seen that no offsets into fields in the affected structure were changed as a result of the structure definition change.

The change in the length, in the last line of the `compare_ascii` comes from this:

```
STORAGE REQUIREMENTS FOR THIS PROGRAM.

         Object   Text     Link     Symbol   Defs     Static
Start         0      0    14216      14374  13154      14226
Length    15566  13154      156       1156   1041          2
```

And reflects the fact that the symbol section is a little bigger because the name of the include segment includes the full path to the updated include segment, which is longer than the path to the installed version of the include segment in `>ldd>include>mcs_echo_neg.incl.pl1`.

5 Version History

2026-09-10	1.0	Initial version of the MCR.