Published: 07/03/68

Identification

merge_edit command package
E. W. Meyer, Jr.

(Note that the following is an Abstract, which should be replaced by a full description at a later time.)

The Multics merge_edit package consists of the following seventeen segments:

(1) merge_edit - command procedures; checks argument format, initializes, calls pass 1 and pass 2, and wakes up tape daemon.

```
call merge_edit (arg1,arg2,arg3,arg4,arg5);
dcl (arg1,arg2,arg3,arg4,arg5)char(*);
```

The Multics Merge-Editor in general follows the conventions of the 6.36 Merge-Editor; see BE.5.02.

The meanings of the arguments are as follows:

1. Normal case

   arg1: control file name (the control file is actually named "name.gecos"; the merge_edit command will append the ".gecos", however)
   arg2: run name (≤ 6 characters)
   arg3: user name (≤ 12 characters)
   arg4,arg5: options - MAC, MH (for specifying that the resultant tape must be run on either the MAC or the Murray Hill machine); NOTAPE (make up control files, do not call Tape daemon to produce a tape). The options are optional.

2. Recovery case

   (If the control files were successfully produced in a previous merge_edit but the tape was not successfully produced, the merge-editor can be used to try producing the tape again. This case is recognized by the presence of only two arguments.)

   arg1: run name (same name as used in the previous run.)
   arg2: "tape" (literal)

(2) mg_pass1 - scans the control line segment and calls the proper handler for each control line.

```
call mg_pass1 (f_dir, f_name);
dcl  (f_dir,                /*directory in which the control
                              segment resides*/
      f_name                /*name of control segment*/
                              )char(*);
```

(3) mg_pass2 - controls the production of the tape driver segments from the list structure produced by pass 1.

```
call mg_pass2 (w_dir);
dcl  w_dir char(*);         /*current working directory*/
```

(4) mg_epl - handler for "epl" control lines.

```
call mg_epl$epl_pass1;
call mg_epl$epl_pass2;
```

(5) mg_eplbsa - handler for "eplbsa" control lines

```
call mg_eplbsa$ebsa_pass1;
call mg_eplbsa$ebsa_pass2;
```

(6) mg_entry - handler for "entry" control lines.

```
call mg_entry$entr_pass1;
call mg_entry$entr_pass2;
```

(7) mg_text_link - handler for "text+link" control lines.

```
call mg_text_link$txlk_pass1;
call mg_text_link$txlk_pass2;

call mg_text_link$putout (s_name, p_name, name_1,
     gf_cd, er_set);        /*used by mg_text_link$txlk_pass2
                              and mg_maketl$mktl_pass2 to put
                              out a binary card image into the
                              binary driver segment*/

dcl  (
     s_name,                /*636 segment name*/
     p_name,                /*Multics pathname*/ )char(*),
     gf_code                /*gefrc code, "tx", "lk", or
                              "st"*/ bit (36),
     er_set fixed bin(17);
                            /*driver code
                              0 - print error message if
                                  p_name not found
                              1 - do nothing if p_name not
                                  found
                              2 - do not look for p_name*/
```

(8) mg_maketl - handler for "maketl" control lines.

```
call mg_maketl$mktl_pass1;
call mg_maketl$mktl_pass2;
```

(9) mg_load_libe - handler for "load", "libe", and "pgsize" control lines; also used to generate pass 1 list structure and put out "load" cards for the "text+link" and "maketl" control lines.

```
call mg_load_libe$load;
call mg_load_libe$libe;
call mg_load_libe$tl_mk;
call mg_load_libe$pgsize;
call mg_load_libe$ldlb
```

(10) mg_fetch - handler for return tape activity-associated control lines: "fetch", "undump", and "notape".

```
call mg_fetch$ftch_pass1;
call mg_fetch$ftch_pass2;
call mg_fetch$undump;
call mg_fetch$notape;
```

(11) mg_deck - handler for dumper activity-associated control lines: "deck", "pure", "core", and "error".

```
call mg_deck$deck_pass1;
call mg_deck$deck_pass2;
call mg_deck$pure;
call mg_deck$core;
call mg_deck$error;
```

(12) mg_control_cards - contains entries to process the "limits", "library", and "simulate" control lines, and to write out the $ LIMITS, $ PERM/$TAPE, and $ use control cards.

```
call mg_control_cards$limits;
call mg_control_cards$write_limits;
call mg_control_cards$library;
call mg_control_cards$write_library;
call mg_control_cards$simulate;
call mg_control_cards$write_use;
```

(13) mg_initld - contains entries to alter the foundation values for the execution/simulation activity and to write out the initld card.

```
call mg_initld$write_initld;
```

```
     call mg_initld$X;
          where X = dspgsz, ntpgsz, lspgsz, dsgbnd, nmtbnd,
          lodarg, fltbas, stpgsz, and time.  These entries
          are the handlers for the corresponding control
          lines;
```

(14) mg_util - utility segment for the merge_edit package.

```
     call mg_util$next_token(token,nl_code);
     dcl  token char(*), nl_code fixed bin(17);
                                   /*returns the next token from the
                                     current control line and sets
                                     nl_code = 1 if token = ctl_char$nl.
                                     Otherwise nl_code = 0.*/

     call mg_util$bad_line;
                                   /*prints on-line an error message
                                     and the current control line*/

     call mg_util$fstate(pathname, fnd_code);
     dcl  pathname char(4), fnd_code fixed bin(17);
                                   /*searches for a non-directory
                                     entry defined by pathname.  If
                                     it can not be found then
                                     fnd_code=1.  Else fnd_code=0;*/

     call mg_util$compil_names(name,compil_list_sw);
     dcl  name char(*),compil_list_sw fixed bin(17);
                                   /*used to retrieve segment names
                                     from the epl and eplbsa lists in
                                     processing the "load*", "fetch*",
                                     and "deck*" control lines.*/

     call mg_util$gebcd(char_6, bit_36);
     dcl  char_6 char(6), bit_36 bit(36);
                                   /*an interface to ascii_gebcd to
                                     convert char_6 ascii to bit_36
                                     GE hollerith code*/
```

(15) mg_list - list processor for the merge_edit package

```
     call mg_list$list_init;  /*sets up an empty list segment*/

     call mg_list$ X (rpr, retpr);
          where X = car, cdr, caar, cadr, cdar, and cddr;
     dcl  (rpr, retpr) fixed bin(17);
                                   /*examines list structure pointed
                                     to by rpr and returns retpr*/
     call mg_list$cons(capr, cdpr, cnpr);
     dcl  (capr,cdpr,cnpr)fixed bin(17);
                                 /*allocates a list cell, places
                                   capr into the car position,
                                   places cdpr into the cdr
                                   position, and returns cnpr, a
                                   relative pointer to the list
                                   cell*/

     call mg_list$add_bits(bits, retpr);
     call mg_list$add_chars(chars, retpr);
     dcl bits bit(*), chars char(*);
                                 /*allocates space for the
                                   supplied string, copies it,
                                   and returns retpr to the stored
                                   copy*/

     call mg_list$get_bits(rpr, rbits);
     call mg_list$get_chars(rpr,rchars);
     dcl  rbits bit(*), rchars char(*);
                                 /*returns the string assumed to
                                   have been stored by add_bits/
                                   add_chars at beginning location
                                   rpr*/
```

(16) mg_file - creates and handles the tape driver control segments.

```
     call mg_file$file_init;     /*calls working_segs$init to
                                   create the ascii and binary
                                   control segments in the process
                                   directory*/

     call mg_file$file_finish(target_dir, perm_name);
                                 /*calls working_segs$finish to
                                   move the control segments to
                                   the working directory under the
                                   names <perm_name>.control and
                                   <perm_name>.control.binary.*/

     call mg_file$binary_block(block,count,blk_ptr);
     dcl  (block,count)fixed bin(17),blk_ptr ptr;
                                 /*allocates the next available
                                   block of block works in the
                                   binary control segment, places
                                   a 7-9 punch and the wordcount
                                   count in word 0, zeros the rest
                                   of the block, and returns
                                   blp_ptr to word 0 of the block*/
```