Published: 05/26/67
(Supersedes: BY.11.01, 10/06/66)

Identification

Seterr - a procedure to write a complete error description at the end of a user's error segment.
D. Widrig, K. J. Martin

Purpose

Seterr gathers all relevant information concerning an error into a standard format and leaves the information in a prearranged place. It is expected that the programmer will use this procedure whenever his procedures wish to convey error information to other procedures.

Usage

The call to seterr requires five arguments. These arguments are sufficient to convey certain minimal error items (see BY.11.00) as well as almost unlimited possibility for expansion into more elaborate error comments. The call and seterr's declarations for the arguments are:

```
call seterr (error_loc, error_code, error_info,
             extra_bit_info, extra_char_info);

dcl error_loc label,

     /* pointer to the location in the offended procedure
     where the error was detected */

error_code char (*) varying,

     /* a short character code to identify the error among
     the possible errors signalled by the offended procedure */.

error_info char (*) varying,

     /* a helpful description of the error, possibly
     including references. Example: "Improper arguments.
     See BX.12.02, MSPM" */

extra_bit_info bit (*) varying,

     /* a bit string containing supplemental information.
     For example a procedure might want to store relevant
     machine conditions on an error. */
```

```
extra_char_info char (*) varying;

     /* a character string containing supplemental information.
     For example, a "file not found" error might include
     the name(s) as supplemental information */
```

Implementation

Seterr builds a structure of error information and places it at the end of the segment error_out in the process directory. The segment error_out is a structure as declared below. The element err_ptr->error_out.space is a threaded list of error-description structures. The element err_ptr->error_out.recent is a relative pointer to the last error-description structure. Each structure contains a relative pointer, error.last_ptr, to the previous structure. Thus, the first error-description structure accessed is the one most recently placed in error_out. The declarations for the segment, error_out, are:

```
dcl err_ptr ptr ext static init (null);
dcl 1 error_out ctl (err_ptr),
    2 recent bit (18),
    2 space area ((13107));
```

The error-description structure is declared as:

```
dcl 1 error ctl (eptr),
    2 last_ptr bit (18),
    2 attempted_delete bit (1),  /* delete requested but
                                    not done, see
                                    BY.11.03 */
    2 time char (9),             /* provided by seterr */
    2 date char (6),             /* provided by seterr */
    2 call_loc,                  /* provided by seterr */
      3 size fixed bin (17),
      3 data char (eptr->error.call_loc.size),
    2 error_loc,                 /* first argument of
                                    call */
      3 size fixed bin (17),
      3 data char (eptr->error.error_loc.size),
    2 error_code,                /* second arg */
      3 size fixed bin (17),
      3 data char (eptr->error.error_code.size),
```

```
2 error_info,                          /* third arg */
   3 size fixed bin (17),
   3 data char (eptr->error.error_info.size),
2 extra_bit_info,                      /* fourth arg */
   3 size fixed bin (17),
   3 data bit (eptr->error.extra_bit_info.size),
2 extra_char_info,                     /* fifth arg */
   3 size fixed bin (17),
   3 data char (eptr->error.extra_char_info.size);
```

Seterr builds the error-description structure as follows:

1. Get the current calendar time using the PL/I built-in functions "time" and "date".

2. Call the procedure who_called (BY.12.01) to trace back in the stack to determine the caller of the offended procedure; that is, the procedure that called the procedure that called seterr. Who_called returns a pointer, call_loc, to the location of the call to the offended procedure.

3. Convert call_loc and error_loc (an argument to seterr) to symbolic form:

```
dcl (c_seg, e_seg) char (31) var, (c,e) char (38)
     var, (c_offset, e_offset) char (6);

c_seg = getname$segment (1, call_loc);
c_offset = substr (bin_oct (ptr$rel (call_loc)),
           7, 12);
e_seg = getname$segment (1, error_loc);
e_offset = substr (bin_oct (ptr$rel (call_loc)),
           7, 12);
c = c_seg||"$"||c_offset;
e = e_seg||"$"||e_offset;
```

The function ptr$rel is described in BY.14; getname$segment is described in BD.3.02; bin_oct is described in BY.7.01. They are declared as

```
dcl ptr$rel ext entry (ptr) bit (18);
dcl getname$segment ext entry (fixed bin (17), ptr)
    char (*) varying;
dcl bin_oct ext entry (bit(36)) char (12);
```

4. Calculate the length of the various character string elements (for use in the third level size elements). These lengths must be calculated before allocating the error-description structure. The length variables calculated are the extents of third level data elements in a dummy structure which is similar to the error-description structure but contains no self-relative extents. This dummy structure must be used for the actual allocation of the error-description structure. The calculations of lengths are:

```
c_size = length (c);
e_size = length (e);
code_size = length (error_code); /* second
                                    argument */
info_size = length (error_info); /* third argument */
bit_size = length (extra_bit_info); /* fourth
                                       argument */
char_size = length (extra_char_info); /* fifth
                                         argument */
```

5. Allocate the error-description structure in the error_out segment using the dummy structure allocation.

6. Fill in the elements of the allocated structure (using the dummy structure declaration for safety). The elements are those obtained in steps 1 - 4.

7. Set err_ptr→error_out.recent to point to this allocation, and set eptr→error.last_ptr to point to the previous structure.

8. Set eptr→error.attempted_delete equal to "0"b. This bit is used to indicate that some procedure tried to delete this error-description but was not allowed to - see BY.11.03.

9. On successful storage of the items of the structure, seterr returns. If seterr is notified of errors in any of the procedures it calls or if the structure cannot be allocated, it comments to the user and signals the condition shell_anchor. Clearly, seterr cannot call itself to announce this error as an infinite loop might result. Upon the Shell's regaining control through the anchor entry point, the Shell signals an error and returns, presumably to the Listener. At this point the user may examine the situation on a more leisurely basis.