From: D. Reed

To: V. Vaydock
R. Feiertag
M. Weaver

File - redesign prop.

Clever!

Could also go to character string.

Error Tables, Error Codes and The like

The problem of multiple error tables is as follows: Both the system, subsystems and users have the necessity to collect meaningful information back to the user about both error and status conditions.

The following operations must be defined on error codes:

1) A "no_error" value must be defined, and in the case of file system error codes this must be a single precision zero.
2) The code must be comparable with symbolic constant values, so that a program may interpret the error condition.
3) The code must give access to ascii printable information explaining the error, so that com_err and other such programs can work.
4) The code must take as little storage as necessary to reflect.

Currently there is no way for a subsystem to provide guaranteed unique code values for a routine such as com_err_ to interpret. This is because codes are predefined binary values, and not values assigned by the system as the codes are used. It is definitely not desirable to reserve, say, codes 1 million to 1 million + 1000 for subsystem y, necessitating registering each subsystem. Nor is it desirable to have individual com_err_'s.

The solution, of course, is to have an error code, or set of error codes, assigned a value or range of values when first used in a process. Since the process will only use probably a small subset of the possible error codes, this will result in an effective allocation, keeping the values unique, yet within a small enough range to be represented, say, in one word.

How can this be done? I reason by analogy to the Multics linkage conventions, and address space management, to get error linkage conventions and errorcode space management — one can

use (symbolically referenced) addresses or pointers.
An immediately obvious advantage presents itself — the pointer can directly specify (point to) the information to be printed.
In addition, the addresses of error_table_$foo, subsystem_errors_$bar, and so on are guaranteed to be distinct, yet reproducible for the comparison necessitated by property 2 above. Not only this, but only a small set of error tables need be known to the process, and none will need be registered with the system.
An example of use might be:

```
call hcs_$erroneous ("me", errorp);
if errorp = addr(error_table_$death_and_destruction)
        then call crash;
        else call alive;
```

And now for the (hopefully upwards compatible) implementation. First, some observations:

1) The error code can take the same space as presently defined error codes, if a short-format pointer is used.
2) If we can change all modules in the system so that existing user programs still work correctly, that is all to the good. This means that some semi-kludge must define the new error pointers so that those programs that expect codes of the binary variety to be returned will still work, by mismatching of declarations.
3) Current error codes may be distinguished from single word pointers by the fact that the upper 18 bits are all zero, whereas for pointers this will not be true.

Thus, we have the following proposal:

1) Change all documentation so that the following is said:
   a. all error codes are declared "ptr unaligned".
   b. The check for "no_error" should be done by the following PL/I statements:

```
dcl error_table_$bad_arg ext aligned fixed bin,
    error_table_$no_error ext aligned fixed bin,
    code ptr unaligned,
    hcs_$erroneous entry(char(*), ptr unaligned);

call hcs_$erroneous("me", code);
if code ^= addr(error_table_$no_error)
   then if code = addr(error_table_$bad_arg)
           then ...
           else call com_err_(code, ...)
   else;
```

what "null" for mean?

   c. This should be explained to users so they realize that their old programs will still work.

   This will be because:

2) <u>KLUDGE</u>: error_table_ must have segment number zero.

erk!

   This is necessary, as too much depends on a zero code value meaning no error.

3) Define an error table entry as it is now, but perhaps with provision for longer messages.

4) Recode com_err_.

5) Recode error_table_ compiler, so that the codes are assigned as 18 bits of zero, followed by 18 bits of the offset of the error entry.

6) In future recodings of system modules, clean up codes to be pointers.

Observations on amount to change:

1) The reservation of segment zero for error_table_ should require some recoding of hard core modules. This ought to be small, except maybe in the KST management routines, which should initiate error_table_ in ring 4 with seg # zero.

2) None of the system or user modules need to be recoded, other than those mentioned above.

3) If segment zero cannot be reserved, the following will also work:

if unspec(ptr) = "111"b then no_error_

and

a reserved segment number, not zero, for error table, which must be put in the high order half of standard error codes by error_table_ compiler.