To: Distribution

From: T. H. Van Vleck

Date: July 24, 1975

Subject: Handling Subroutine Errors

INTRODUCTION

Multics conventions currently forbid subroutines which may
be called by many different programs from performing output
unless that is their primary purpose. The reason for this rule
is the "principle of transparency," which requires that the
subroutine be usable in environments which do not have standard
I/O attachments, and in environments which wish to use the
subroutine without obtaining any output. In particular,
subroutines are currently forbidden to use com_err_ to report
status. The standard method for reporting status is to supply an
additional argument to the subroutine which will be set to zero
or to a standard status code by the subroutine.

The caller of such a subroutine must have some knowledge of
the cases in which status codes are returned. Often, the calling
program has the choice of including a series of tests for each of
the possible statess recognized by the subroutine, or of simply
assuming that any nonzero status code indicates that the routine
failed. When a status code is returned, the calling program
often wishes to produce a message describing the situation. But
in some cases, the subroutine can recognize so many different
situations that the calling program will be unable to produce a
helpful message without additional communication between the
calling program and the subroutine. This problem was encountered
in the design of delete_, when deleting a directory. If a
directory contains items which cannot be deleted, there is
currently no clean way to inform the caller of delete_ of the
pathname of the item.

Subroutines which can detect multiple errors (such as
compilers) have an even more difficult problem. The returning of
a status code is suited only to the detection of single errors.
Requiring the calling program to allocate storage for a usually
null array of status indicators or status messages seems
uneconomical; and saving the messages in storage allocated by
the subroutine encounters other problems if multiple invocations
of the routine may exist in the same process.

When we think of small subroutines, like square root
programs, these problems can be ignored or papered over. But
when large subsystems are used as subroutines of user application
programs, the need for a mechanism which allows subroutines and
subsystems to report status in detail, while still allowing the
calling environment control over the actual output and the
content of the message, becomes more and more important.

PROPOSAL

To accomplish these goals, a new subroutine is proposed,
called sub_err_, which will be used by subroutines in much the
same way that com_err_ is used by commands. Draft MPM SWG
documentation is attached. (The "sub" can be considered to be a
contraction of either "subroutine" or "subsystem.") A call to the
subroutine might look like this:

          call sub_err_ (code, "sort", infop, "c", retval,
                "Input record ^d ignored.", record_no);

When sub_err_ is called, the format string is assembled in the
same way that ioa_ does it, a structure is filled in, and the
condition

                          sub_error_

is signalled. Unlike the call to com_err_, however, sub_err_
does not print the message on return from the signal; it assumes
that the environment has disposed of the message.

The default_error_handler_ for standard Multics processes
will in fact currently print out such a message. However, the
format of the message currently produced should be improved
slightly, so that it looks something like this:

          name error by callername | location..
          Com_err string. Ioa_ formatted string.

(The message returned by com_err_ for small integer codes should
also be changed from "Code 1 not found in error_table_" to just
"Code 1".) For example, the call above might produce the message

          sort error by sort_|1234 (bound_sort_|5677)
          Record too short. Input record 334 ignored.

The sort routine could use sub_err_ as a way of printing a
message; or, by adding code which tested the value of retval, it
could allow the user the chance to intercept the error and
specify, for example, that the record be padded out with blanks.

The use of the "retval" argument is to allow environments
which wish to intercept the sub_error_ condition and specify
alternative action to the subroutine. The standard environment
will set retval to zero. It might be possible to propose a
future extension to the start command so that the command

    start -return 7

would locate the condition information structure, set retval to
7, and return to signal_.

The introduction of the sub_error_ condition is in fact a
concealed incompatible change for those users who have their own
default error handlers, since it now becomes a requirement that
the handler for sub_error_ understand the "no_pause" switch and
be able to dispose of the output message. The key step is the
introduction of a new principle, obverse to the principle of
transparency, which is that every process ought to have a handler
of last resort.

All subroutines which call sub_err_ should have the fact
noted in their documentation, showing the name and code values
used in each possible call and the action taken on return with
whatever values of retval are allowed.

Programs which have a handler for sub_error_ must check the
condition information structure and be prepared to pass signals
on if they cannot handle them.

Name: sub_err_

This program is called by subroutines which which wish to report an unexpected situation. The subroutine specifies an identifying message and may specify a status code. Switches which describe whether and how to continue execution and a pointer to further information may also be passed to sub_err_. The environment which invoked the subroutine caller of sub_err_ may intercept and modify the standard system action taken when sub_err_ is called.

Usage:

    dcl sub_err_ entry options (variable);

    call sub_err_ (code, name, flags, infop, retval,
        ctl_string, ioa_args);

where

1) code   is a status code describing the reason for
                        calling sub_err_. code should be declared
                        fixed bin (35). (Input)

2) name   is the name of the subsystem or module on
                        whose behalf sub_err_ is called. name should
                        be declared as a nonvarying character string.
                        (Input)

3) flags   describe how and whether restart may be
                        attempted. Flags should be declared as a
                        nonvarying character string. (Input)

                        The following values are permitted:

    "c"   continue after printing message.
    "f"   fatal error. No restart allowed.

4) infop   is an optional pointer to information
                        specific to the situation. The standard
                        system environment does not use this pointer,
                        but it is provided for the convenience of
                        other environments. infop should be an
                        aligned pointer. (Input)

5) retval   is a return value from the environment to
                        which the error was reported. The standard
                        system environment sets this value to zero.
                        Other environments may set retval to other
                        values, which may be used to select recovery
                        strategies. retval should be declared fixed
                        bin (35). (Input/Output)

6) ctl_string   Is an loa_ format control string which
                       defines the message associated with the call
                       to sub_err_. Consult the description of loa_
                       in AG93. ctl_string should be declared as a
                       nonvarying character string. (Input)

7) loa_args   are any arguments required for conversion by
                       ctl_string. (Input)

Operation

     Sub_err_ proceeds as follows: the structure described below
is filled in from the arguments to sub_err_, and the system
subroutine signal_ is called to raise the "sub_error_" condition.

     When the standard system environment receives a sub_error_
signal, it prints a message of the format

     name error by subrname|location
     Status code message. Message from ctl_string.

The standard environment then sets retval to zero and returns, if
"c" was specified; otherwise it calls the listener. If "start"
is typed, the standard environment will return to sub_err_, which
will return to the subroutine caller of sub_err_ unless "f" was
specified.   If   "f"   was specified, sub_err_ will signal
"illegal_return."

Use by Subsystems

     If an application program wishes to call a subsystem which
may report errors by sub_err_, and wishes to replace the standard
system action for some classes of sub_err_ calls, the application
should establish a handler for the "sub_error_" condition by a
PL/I ON-statement. When the handler is activated as a result of
a call to sub_err_ by some dynamic descendant, the handler should
call find_condition_info_ to obtain the "software_info_ptr" which
will point to a structure with the following declaration.

     dcl 1 info aligned based (software_info_ptr),
         2 length fixed bin,
         2 version fixed bin,
         2 action_flags aligned,
           3 cant_restart bit (1) unal,
           3 default_restart bit (1) unal,
           3 pad bit (34) unal,
         2 info_string char (256) var,
         2 code fixed bin (35),

```
2 retval fixed bin (35),
2 name char (32),
2 infop ptr;
```

where

length   is the size of the structure in words.

version   is the version number of the structure.
                        This is version 2.

cant_restart   is "1"b if the condition cannot be
                        restarted.

default_restart   is "1"b if the standard environment will
                        print the message and continue execution
                        without calling the listener.

pad   is padding

info_string   is the converted message from ctl_string
                        and ioa_args.

code   is the status code.

retval   is the return value. The standard
                        environment sets this value to zero.

name   is the name of the module encountering
                        the condition.

infop   is a pointer to additional information
                        associated with the condition.

The handler should check info.name and info.code to make sure
that this particular call to sub_err_ is the one desired, and if
not call continue_to_signal_. If the handler determines that it
wishes to intercept this call to sub_err_, the info structure
will provide the message as converted, switches, etc. Any change
made to the value of info.retval will be returned to the caller
of sub_err_ if control returns to sub_err_.