UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 65

(November 1985)

Title:
Language-independent Event Trapping Mechanism on EMAS-3

Author:	Contact:	Software Support Category:
Stephen Hayes	Advisory service	Not yet set

Synopsis

This Note describes the language-independent method on EMAS-3 of trapping program errors and operator interrupts.

Keywords

Contingency, EMAS 3 DISCARD TRAP, EMAS 3 EVENT DATA, EMAS3 ALLOW INTERRUPTS, EMAS3 SIGNAL, EMAS3 GIVE EVENT, EMAS3 SET TRAP, EMAS3 UNSET TRAP, EMAS3 TRAP, EMAS3 TRIGGER TRAP, Event trapping, interrupt character, PTRAP

Edinburgh Regional Computing Centre
James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ. Telephone 031-667 1081

Introduction

This is the language-independent method of trapping program errors and operator interrupts i.e. Int:A etc. The 2900 mechanism of reroute contingency is not available on EMAS-3.

Though the examples given here are in IMP, all the routines are available for use from all languages, and the same method is applicable.

Notes

1. IMP %events still exist and work as defined in the IMP documentation. They are independent of this mechanism and there may be significant differences between the operation of the two schemes.
   The mechanism described here takes priority over any %on %event traps.

2. Other languages may have their own fault trapping mechanisms, and just as for IMP these may be different from the language-independent scheme described here.

3. The language-independent scheme cannot be guaranteed to cope with all the requirements of every programming language. The language's own scheme may be expected to be better integrated with the language The present mechanism is intended to supply the need where a language has no fault trapping, or where the language's trapping cannot intercept the events which the programmer wants to handle.

4. It is probably inadvisable, although it may be possible, to mix the use of this scheme with a language-specific mechanism in one unit of software.

5. The language-independent scheme is used within the EMAS-3 Subsystem, and it is the most direct way for a program to control the Subsystem's handling of events.

The external routine specifications which may be required are as follows:

```
%external %routine %spec EMAS3 (%string %name COMMAND, PARAMS,
                                %integer %name FLAG)

%external %routine %spec EMAS3 ALLOW INTERRUPTS

%external %routine %spec EMAS3 DISCARD TRAP (%integer %name ID, FLAG)

%external %routine %spec EMAS3 EVENT DATA (%integer %name AD50, FLAG)

%external %routine %spec EMAS3 GIVE EVENT (%integer %name CLASS,
                                SUBCLASS)

%external %routine %spec EMAS3 SET TRAP (%integer %name ID, CLASS,
                                SUBCLASS, FLAG)

%external %routine %spec EMAS3 SIGNAL (%integer %name CLASS, SUBCLASS)

%external %routine %spec EMAS3 TRAP (%integer %name ID, PROT, FLAG)

%external %routine %spec EMAS3 TRIGGER TRAP (%integer %name CLASS,
                                SUBCLASS)
```

```
%external %routine %spec EMAS3 UNSET TRAP (%integer %name ID, CLASS,
                                            SUBCLASS, FLAG)
```

The command PTRAP may be used to print out the state of the traps currently defined. It can be called from a program using EMAS3. For example

```
EMAS3("PTRAP","",FLAG)
```

Summary

The mechanism provides for the definition of up to sixteen different 'traps'. Each trap can have one or several events associated with it. The occurrence of such an event causes the corresponding trap to be 'sprung', i.e. code provided by the programmer is executed.

Using the mechanism

Up to 16 traps may be defined. They are identified by the ID parameter in the %specs above: ID will be returned by a call of EMAS3 TRAP, and will have a value in the range 1 to 16.

A trap is defined by calling EMAS3 TRAP, as in the following example:

```
 :
EMAS3 TRAP(ID,0,FLAG)
%if FLAG#0 %then %start
! Action to be taken when the event occurs ....
%finish
 :
```

When EMAS3 TRAP(ID,0,FLAG) is first called, it sets FLAG to zero so that the 'action' is skipped. It also puts an identification number for the trap into the %integer %name parameter ID. This value must be kept for future reference. The events to be associated with the trap are specified by use of EMAS3 SET TRAP, described below. When an event occurs that has been associated with trap ID, then the Subsystem passes control to the program just as if control were returning from the EMAS3 TRAP(ID,0,FLAG) call, but with FLAG set non-zero. That means that the action will now be performed.

The second parameter to EMAS3 TRAP would normally be zero. A non-zero value has the effect of ensuring that asynchronous interrupts are disabled when the trap is sprung - i.e., when the action is initiated, interrupts are inhibited and the action must include a call of EMAS3 ALLOW INTERRUPTS. Only asynchronous interrupts, such as Int:A, can be disabled; program error interrupts, which are synchronous, can not be disabled in this manner.

After this call of EMAS3 TRAP, the system knows that the trap exists but it does not know which events are to spring the trap. Clearly the trap can never be sprung until the system is told which events are associated with the trap. This is achieved by calls of EMAS3 SET TRAP.

Each trap can handle one CLASS or several CLASSes of event. It can be defined to handle one SUBCLASS of a CLASS, or several SUBCLASSes, or all SUBCLASSes. If it handles more than one CLASS, it can handle a different set of SUBCLASSes for each CLASS. SUBCLASSes are in the range 0 to 127.

EMAS3 SET TRAP(X, 10, -1, FLAG) says that trap number X is to handle all events of class 10, regardless of subclass. X must be the identification number returned by a previous call of EMAS3 TRAP.

EMAS3 SET TRAP(X, 12, 66, FLAG) says that trap number X is also to handle events of class 12 with subclass 66.

EMAS3 SET TRAP(X, 12, 98, FLAG) could then be called, so that in total trap number X handles subclasses 66 and 98 of class 12 events as well as all class 10 events.

EMAS3 SET TRAP sets parameter FLAG to zero if it is successful.

EMAS3 UNSET TRAP can be used to remove particular classes and subclasses associated with the trap by previous calls of EMAS3 SET TRAP. EMAS3 UNSET TRAP cancels the effect of a call of EMAS3 SET TRAP, but it does not cancel the effect of TRAP, so the trap still exists (even if there are no events which could spring the trap) and further calls of EMAS3 SET TRAP can associate a new set of events with the trap.

EMAS3 DISCARD TRAP is used to discard a trap altogether. It cancels all calls of EMAS3 SET TRAP for that trap, and also cancels the original EMAS3 TRAP call. The trap would have to be redefined by a new call of EMAS3 TRAP before any EMAS3 SET TRAP calls would be acceptable.

EMAS3 TRIGGER TRAP is used to provoke an event and thus to spring a trap. It is ignored if no user-level trap has been set for the CLASS and SUBCLASS specified in the call of EMAS3 TRIGGER TRAP.

EMAS3 SIGNAL (not the same as IMP %signal %event) is like EMAS3 TRIGGER TRAP except that if no user-level trap is defined then the system's trap will be sprung, which may lead to the program being terminated with diagnostics.

When a trap is sprung, the action code may call EMAS3 GIVE EVENT to determine the CLASS and SUBCLASS of the event which has occurred.

EMAS3 EVENT DATA(AD50, FLAG) will return more comprehensive information. AD50 should be the first of at least 50 consecutive integers. The data is copied into these 50 integers. Its format is closely tied to the architecture of the EMAS-A hardware, and so may be different if EMAS-3 is run on any different machine. Some of the important fields are as follows (the integers are numbered 1 to 50):

Word 1	Class
Word 2	Subclass
Word 3	Program Status Word 0.
Word 4	Program Status Word 1. Contains program counter in least significant 31 bits.
Words 5-20	General registers GR(0:15)
Words 21-28	Floating point registers FR(0:3) Each floating point register occupies two words.

Not all the fields are always set, but the general registers should always reflect the state of the machine at the time of the event.

Event classes

'Ordinary' class numbers have zero in the top byte. If it is wished to define private class numbers (probably in order to use EMAS3 TRIGGER TRAP), it is advised to use numbers whose top byte is X'01'.

Some of the events which one might most wish to trap are the asynchronous terminal interrupts. These interrupts are also used in the process closedown procedure. All these events have class 65 and subclass set to the upper case value of the interrupt character. For example an event of class 65, subclass 'A' corresponds to Int:A, and also to Int:a.

The meanings of the various interrupt characters are as follows:

A	User requests abort of current program, and of any current terminal output.
C	User requests abort of current program, of any current terminal output, and that any typed-ahead terminal input should be ignored.
Q	User requests program diagnostics.
K	Kill output...not trappable.
T	Give current time and page turns...not trappable.
V	End of booked slot (for Kent). Process should stop immediately.
W	Process has been inactive for a long time, and hence is being requested to stop.
X	Central operator requests user process to stop immediately.
Y	Interactive terminal has cleared down - possibly because of a communications break. There can be no further output to the terminal after this event has been signalled.

The following list contains the classes of some of the other possible events. Some of these should never occur. Not all of them are trappable.

Contingency	Class	
Floating overflow	0	
Floating underflow	1	
Fixed overflow	2	
Decimal overflow	3	
Zero divide	4	
Bound check	5	
Size error	6	
B overflow	7	
Stack error	8	
Privilege	9	
Descriptor	10	
String	11	
Instruction	12	
Accumulator	13	
System Call	16	
Instruction counter	17	i.e. time exceeded.
Disc transfer fail	18	
Block wrong length ("should not occur")	19	
Hard store fault	20	
Illegal OUT	21	
Local Controller error	22	
Virtual store error	32	
Interval timer	64	
Single-character "Int:"	65	
Text message	66	
Off stack top	67	
Message from a process	68	

Of these, classes 64, 65, 66 and 68 are ASYNCHRONOUS and all the others are SYNCHRONOUS.

The following example traps Int:X, Int:Y and Int:W.

```
%routine userprg
   %integer trapid, flag, class, subclass, resflg
   %external %routine %spec EMAS3 TRAP (%integer %name ID, PROT, FLAG)
   %external %routine %spec EMAS3 DISCARD TRAP (%integer %name ID, FLAG)
   %external %routine %spec EMAS3 GIVE EVENT (%integer %name CLASS,
                                             SUBCLASS)
   %external %routine %spec EMAS3 ALLOW INTERRUPTS
   %external %routine %spec EMAS3 SIGNAL (%integer %name CLASS, SUBCLASS)
   %external %routine %spec EMAS3 SET TRAP (%integer %name ID, CLASS,
                                             SUBCLASS, FLAG)

   emas3 trap(trapid,1,resflg); !define trap
   ! returns with resflg=0.
   ! program also resumes here after an event. In this case, resflg=1
   %if resflg#0 %thenstart
         ! this code executed after a contingency.
         emas3 give event(class,subclass);        !get class and subclass of event
         !
         { recovery code goes here}
         !
         emas3 discard trap(trapid,flag);         !remove trap
         emas3 allow interrupts;      !re-enable asynchronous interrupts
            !not required in this case. see note
         emas3 signal(class,subclass); !now send to higher level code
            !signal does not return
   %finish
   ! this code is executed first
   %if trapid<0 %thenstart
      ! may occur if program is called from another repeatedly,
      ! and all the traps have thus been used up.
      printstring("cannot define trap")
      newline
   %finishelsestart
      ! define conditions on which trap is to be sprung
         emas3 set trap(trapid,65,'X',flag)
         emas3 set trap(trapid,65,'Y',flag)
         emas3 set trap(trapid,65,'W',flag)
   %finish
   !
   { main body of program goes here}
   !
   %if trapid>=0 %thenstart
      emas3 discard trap(trapid,flag); !discard trap before returning
   %finish
%end
```

In the above example, the trap is defined using EMAS3 TRAP. In this case, PROT (second parameter) is specified as 1, which will be referred to later. The program returns from the call of EMAS3 TRAP with the variable RESFLG set to zero. Hence the %start - %finish block containing the recovery code is skipped.

It is possible that the call of EMAS3 TRAP will not succeed, because a large number of traps have already been defined. In this unlikely case, EMAS3 TRAP returns with the variable TRAPID negative.

After defining the trap, it is necessary to define the conditions under which it will be sprung. In this case, the single-character interrupts 'X', 'Y', 'W' are the conditions which are to be trapped.

The program now proceeds to perform its main function. In the ordinary case, there will be no interrupts. The program should discard the trap before returning to the calling routine. Failure to do this could have disastrous consequences if an event occurs after the return is made. If the program calls %stop, then the trap will be discarded automatically.

If a trapped event occurs, then normal sequential execution of the program is interrupted. The program resumes following the call of EMAS3 TRAP, but now RESFLG is set to 1. Thus, the program executes the recovery block of code delimited by %start - %finish.

In this example, PROT has been specified as 1 when calling EMAS3 TRAP. This has the effect of preventing ANY other asynchronous interruption while the program is executing the recovery-block of code. It does not prevent interruptions because of program error. The trap remains in effect until discarded.

In a program which trapped program errors, it would be necessary either to discard the trap immediately after the event occurred, or write special code to deal with the case where the recovery code itself provoked a further trap.

The example above starts by determining the cause of the interrupt, using EMAS3 GIVE EVENT. A program which trapped program errors would use EMAS3 EVENT DATA to obtain the registers at the time of the failure.

Some recovery action is taken next. This recovery action must depend on variables which exist and are valid at this level of the program. If the program has been interrupted in some inner routine, then it is not readily possible to determine the values of the local variables of that routine.

The trap is discarded, as it is no longer required. It can be discarded at any stage, and this MUST be done prior to EMAS3 SIGNAL, or the code will recurse.

When recovery action is complete, the EMAS3 ALLOW INTERRUPTS is called to allow further interruption. This would not have been necessary if PROT had been set to 0. Also in this case it is not necessary to allow interrupts, as EMAS3 SIGNAL takes account of the fact that interrupts may be disabled.

The example exits by calling EMAS3 SIGNAL. This has the effect of sending the interrupt to the program which called this example program. The calling program may itself take recovery action before signalling the event back to the subsystem, or the program which called that.

The example could also have terminated by calling %stop. In this case the actions of EMAS3 DISCARD TRAP and EMAS3 ALLOW INTERRUPTS would have been done automatically. Also, any Int:V, Int:W, Int:X or Int:Y would have been signalled to the next program level down. Other interrupts would be ignored.

In the case of a main program or command, it is appropriate to exit by calling %stop if necessary. In the case of a routine, as in our example, it is more appropriate to call EMAS3 SIGNAL or %return following error processing.