Catch and Throw Error Handling#

General Information

Author: Action Computer Services
Language: ACTION!
Compiler/Interpreter: ACTION!
Published: 1984

This module provides two PROCs (Catch and Throw) which can be used for error trapping (and flow control, yeck!) in ACTION!. To use them, you must call the Catch PROC to indicate where you want the program to continue when you call Throw. When throw is called, execution will continue following the last call to Catch with the same index as the call to Throw. Calling Catch is similar (but not identical) to TRAP in BASIC. It differs in that the actual trapping is generated by the user (by calling Throw) and that you can have multiple Catch'ers active at one time. Also, you cannot Throw to a Catcher that is no longer active (the PROC/FUNC containing it has RETURN to it's caller). The Throw procedure tries to check for this error, but it is possible to fool it into thinking it's OK.

If you want to solve this problem, you can set 'c_t_sp(index)' to zero before you return from the PROC that contained the Catch(index). If index is greater than 24 or if there is no matching Catch index for the Throw, then Error will be called with a value of CTERR (defined below to be 71). If you setup your own Error procedure and use Catch and Throw, your error procedure should handle this error as well or your program will most likely "go off the deep end".

```
MODULE ; CATCH.ACT

; copyright (c) 1984
; by Action Computer Services
; All Rights Reserved

; This module provides two PROCs
; (Catch and Throw) which can be used
; for error trapping (and flow
; control, yeck!) in ACTION!.  To
; use them, you must call the Catch
; PROC to indicate where you want
; the program to continue when you
; call Throw.  When throw is called,
; execution will continue following
; the last call to Catch with the
; same index as the call to Throw.
; Calling Catch is similar (but not
; identical) to TRAP in BASIC.  It
; differs in that the actual trapping
; is generated by the user (by
; calling Throw) and that you can
; have multiple Catch'ers active at
; one time.  Also, you cannot Throw
; to a Catcher that is no longer
; active (the PROC/FUNC containing
; it has RETURN to it's caller).  The
; Throw procedure tries to check for
; this error, but it is possible to
; fool it into thinking it's OK.  If
; you want to solve this problem, you
; can set 'c_t_sp(index)' to zero
; before you return from the PROC
```

```
; that contained the Catch(index).
; If index is greater than 24 or
; if there is no matching Catch index
; for the Throw, then Error will be
; called with a value of CTERR
; (defined below to be 71).  If you
; setup your own Error procedure and
; use Catch and Throw, your error
; procedure should handle this error
; as well or your program will most
; likely "go off the deep end".


DEFINE CTERR = "71"

BYTE ARRAY c_t_sp(25)=[0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
BYTE ARRAY c_t_hi(25), c_t_lo(25)


PROC Catch(BYTE index)
  DEFINE TSX="$BA", TXA="$8A",
            LDYA="$AC", STAY="$99",
            PLA="$68", LDAY="$B9",
            PHA="$48"

  IF index>=25 THEN
     Error(CTERR,0,CTERR) FI

  [
     LDYA index
     PLA
     STAY c_t_hi
     PLA
     STAY c_t_lo
     TSX
     TXA
     STAY c_t_sp
     LDAY c_t_lo
     PHA
     LDAY c_t_hi
     PHA
  ]
RETURN


PROC Throw(BYTE index)
  DEFINE TXS="$9A", PHA="$48",
            LDYA="$AC", STX="$86",
            TSX="$BA", TAX="$AA",
            LDAY="$B9"

  BYTE sp=$A2

; get current stack pointer
  [ TSX : STX sp ]

  IF index>=25 OR sp+2>c_t_sp(index)
     THEN Error(CTERR,0,CTERR) FI
```

```
  [
     LDYA index
     LDAY c_t_sp
     TAX
     TXS
     LDAY c_t_lo
     PHA
     LDAY c_t_hi
     PHA
  ]
RETURN

MODULE ; just in case
```