Customer Support Engineering Tech Note #20

Simple MotionBASIC® Error Handler

Abstract

Error handling is vital part of any MotionBASIC® program. A well designed error handler enhances safety, minimizes collateral damage from malfunctions and reduces downtime by making what is wrong with the system clear to the operator. This Tech Note is an example of a simple error handler that can be used as a model for your applications.

Description

The main priorities for an error handler are:

- React to the problem and render the machine safe.
- Annunciate what the problem is.
- Clear the faults and restart the program.

This example assumes you are using video display to annunciate the faults. The techniques can easily be modified to work with an MMI-840. Tech Note 21, shows how you can annunciate error codes using the user LEDs and I/O points instead of a display screen.

Implementation

The POWERUP routine shows how to initialize the program and its error handler. It includes calls to subroutines to initialize things like ON EVENT routines and user defined functions even though the example does not use them. They are included to show you where they should be initialized in your program.

POWERUP establishes ERROR.HDLR as the error handling routine then checks the ESTOP.OK@ input and either starts the main program or forces an ESTOP fault.

ERROR.HDLR is the routine the program jumps to any time a MotionBASIC® error occurs. First it stores the value of the error code then checks to see if it was a CTRL-C from MotionPRO™. During program development you need to be able to stop your program without going through the full error handling process. This allows you to do that. If MotionPRO™ is going to be permanently connected to the machine after development is complete, you should delete this test since stopping without going through the error handler can be hazardous.

If the error is not a CTRL-C, ERROR.HDLR halts all axes then waits for the commanded decelerations to complete. If FAULT@ is true, the No Fault relay will be open and the servo bus power will be off (the NO FAULT relay should drop out the bus power contactor). Since the servos would be coasting, it is possible they will generate position error or servo undervoltage faults before they complete their decelerations. These secondary faults confuse the situation making the original problem harder to troubleshoot. To prevent the secondary faults, if FAULT@ is true, ERROR.HDLR disables the axes immediately without waiting for the axes to come to rest.

Not all MotionBASIC® errors open the No Fault relay. ERROR.HDLR checks FAULT@ and if there are no controller faults, opens the No Fault relay by setting FAULT@ true. Finally, ERROR.HDLR rearms error handling by executing a RESUME statement that sends the program to the ESTOP.STATE routine.
Program errors in your error handler will cause your program to stop running and enter direct mode. This forces the operator to cycle power, reset the controller or RUN the program to recover. To avoid this possibility, there should be the minimum possible amount of time and code between the start of the error handler and the RESUME statement. However if an error occurs after the RESUME statement, the program will re-enter the error handler. This could put the program in an infinite loop preventing it from executing some critical safety code. It is therefore important that all code necessary for a safe ESTOP be before the RESUME statement.

**ESTOP.STATE** is the routine where your program will wait for the operator to clear the faults. On entering ESTOP.STATE, the program calls **DISARM EVENTS** which explicitly disarms ON EVENT trapping. When your program RESUMEes from the error handler, ON EVENT trapping is automatically re-enabled. DISARM EVENTS is where you would explicitly turn off any EVENTS that you did not want to be automatically re-armed.

ESTOP.STATE then turns on a fault output and stores the state of the ESTOP.OK@ input. It then calls a subroutine called **DISPLAY FAULTS** to annunciate any problems. The example routine shows a logical way to display fault information. You can modify this to suit your own particular style.

On returning from DISPLAY.FAULTS, the program enters a loop that waits for the ESTOP.OK@ input to make a transition from false to true. Once it sees this transition it will clear the faults and restart your main program routine **RESTART**.

It is possible that faults will occur while your program is deeply nested in subroutines. If it does, the program will go to the error handler leaving the subroutine return addresses on the control stack. If this happens enough times, the stack will overflow and you will get an Out of Memory error. To prevent this, the program executes a STACK CLEAR statement to empty the control stack before leaving ESTOP.STATE.

The **RESTART** routine is the start of your main program. Near the beginning of it, the program calls ARM EVENTS which re-arms any ON EVENT trapping your program uses. While your program is in an EVENT routine, the handling of all events is suspended. If a fault occurs before you return from the EVENT routine EVENTS will not get re-enabled. The EVENT ON statements makes sure they do get re-enabled.

Since this is just an example, there is no real main program. Instead the example includes a small routine that allows you to force MotionBASIC errors to see how the program responds.

Your system may not include any actual servos. If it does not, you may need to comment out the MODE@=5 statement near the beginning of **RESTART** to prevent nuisance faults.
Example Program

```basic
'!AUTO 0

Routine name: POWERUP

Abstract: Program entry point. Performs all the needed initialization then puts the machine in the E-STOP state.

Routines called: *MP.CONFIG, CLEAR.FAULTS, ERROR.HDLR, INIT.DIO
                   INIT.EVENTS, INIT.FUNCTIONS, INIT.VARS, RESTART

Variables used: ESTOP.OK@

POWERUP:
  MP.CONFIG 'run the unit/axis configuration routine. You will need to add this routine to your program using MotionPRO’s configuration menus.
  INIT.DIO 'initialize the discrete I/O
  INIT.VARS 'initialize your program variables
  INIT.EVENTS 'initialize any ON EVENT routines
  INIT.FUNCTIONS 'initialize any user defined functions
  CLEAR.FAULTS
  ON ERROR GOTO ERROR.HDLR
  IF ESTOP.OK@ THEN RESTART ELSE ERROR 1910
END

Routine name: INIT.EVENTS

Abstract: Initialize any EVENT subroutines.

Routines called: None

Variables used: None

INIT.EVENTS:
  'Any ON EVENT ... GOSUB statements, go here.
RETURN

Routine name: INIT.FUNCTIONS

Abstract: Initialize any user defined functions

Routines called: None

Variables used: None

INIT.FUNCTIONS:
  'Any DEF FN ... statements go here.
RETURN
```
Routine name: INIT.VARS
Abstract: Initialize any variables your program uses
Routines called: None
Variables used: INITIALIZED

INIT.VARS:
IF NOT INITIALIZED THEN
  'Things that only need to be initialized once
  'when the program is first run, go here.
ENDIF
  'String variables and other variables you need to
  'initialize each time the program is run, go here.
INITIALIZED=TRUE
RETURN

Routine name: INIT.DIO
Abstract: Initialize any DIO points your program uses
Routines called: None
Variables used: FAULT.LIGHT, IO.MODE@()

INIT.DIO:
  FAULT.LIGHT=1 :IO.MODE@(FAULT.LIGHT)="O"
  'Additional DIO initialization statements go here.
RETURN

Routine name: ARM.EVENTS
Abstract: 
Routines called: None
Variables used: None

ARM.EVENTS:
  'any EVENT DIO(..) ON statements go here. Depending on your application
  'some EVENTS may need to be armed at specific places in your program
  EVENT ON    're-arm EVENT TRAPPING
RETURN
**DISARM.EVENTS:**

EVENT DIO@() OFF statements go here

Any error will suspend EVENT trapping. You should use this routine to explicitly turn EVENT trapping off for DIO@() EVENTS you do not want automatically re-enabled when you resume from the error handler and execute the EVENT ON statement in ARM.EVENTS.

RETURN

**ESTOP.STATE:**

DISARM.EVENTS 'turn off any EVENT trapping that you do not want automatically re-enabled.

DIO@(FAULT.LIGHT)=ON
ESTOP.FLAG=NOT ESTOP.OK@
DISPLAY.FAULTS
PRINT
COLOR 15,0
PRINT "Cycle the ESTOP OK input to clear faults and restart...";
COLOR 7,0
EXIT=FALSE
WHILE NOT EXIT
   IF ESTOP.OK@ AND ESTOP.FLAG THEN
      EXIT=TRUE
   ELSEIF NOT ESTOP.OK@ THEN
      ESTOP.FLAG=TRUE
   ENDIF
WEND
CLEAR.FAULTS
STACK CLEAR
GOTO RESTART
'RESTART your main program
Program: TN020.BAS

Routine name: DISPLAY.FAULTS

Abstract: Waits for a time set in the DELAY variable (ms) while it waits.

Routines called: None

Variables used: AFAULT@(), ALARM@(), AXIS.FAULT@, AXIS.FLT1@

DISPLAY.FAULTS:
COLOR 7,0
CLS
PRINT "Controller fault(s): ";FAULT@
PRINT IF FAULT@={} THEN PRINT " No Faults"
IF {1}*FAULT@ THEN PRINT " RAM Checksum Error"
IF {2}*FAULT@ THEN PRINT " Battery Failure"
IF {3}*FAULT@ THEN PRINT " Invalid Unit ID"
IF {4}*FAULT@ THEN PRINT " Internal Error"
IF {5}*FAULT@ THEN PRINT " Axis Module Failed"
IF {6}*FAULT@ THEN PRINT " E-STOP Input Open 
IF {7}*FAULT@ THEN PRINT " Axis Fault"
IF {8}*FAULT@ THEN PRINT " Program Generarted Fault"
IF {9}*FAULT@ THEN PRINT " String Space Fault"
IF {10}*FAULT@ THEN PRINT " L.E.M. Card Fault 
IF (NOT {1,2,3,4,5,6,7,8,9,10})*FAULT@ THEN PRINT " Unknown Fault(s)"
PRINT PRINT "Program error(s): "
PRINT IF STORED.ERR THEN
PRINT USING " Code ####:& at line #####";STORED.ERR,ERR$(STORED.ERR),ERL
ELSE PRINT " No program errors"
ENDIF PRINT PRINT "Axis fault(s):"
PRINT IF AXIS.FLT1@ THEN
PRINT " Axis ";AXIS.FLT1@;" - was the first to fault"
ELSE PRINT " Fault was not caused by an axis"
ENDIF FOR TMP~ WITHIN AXIS.FAULT@
PRINT " Axis ";TMP~;" - fault code";AFAULT@(TMP~);"(";
IF AFAULT@(TMP~)=0 THEN
PRINT "No faults")"
ELSEIF AFAULT@(TMP~)=1 THEN
PRINT "Max position error")"
ELSEIF AFAULT@(TMP~)=2 THEN
PRINT "Servo drive alarm code";ALARM@(TMP~);") "
ELSEIF AFAULT@(TMP~)=3 THEN
PRINT "Open wire, channel A")"
ELSEIF AFAULT@(TMP~)=4 THEN
PRINT "Open wire, channel B")"
ELSEIF AFAULT@(TMP~)=5 THEN
PRINT "Pos. cmd. overspeed")"
ELSEIF AFAULT@(TMP~)=6 THEN
PRINT "Pacer overspeed")"
ELSEIF AFAULT@(TMP~)=7 THEN
PRINT "Axis encoder overspeed")"
ELSEIF AFAULT@(TMP~)=8 THEN
PRINT "MotionDATA error")"
ELSEIF AFAULT@(TMP~)=9 THEN
PRINT "LOOP.RATE@ too high")"
ELSEIF AFAULT@(TMP~)=10 THEN
PRINT "Pacer rate too high")
ELSEIF AFAULT@(TMP~)=11 THEN
PRINT "Missing MotionDATA")
ELSE
PRINT "Unknown fault")
ENDIF
NEXT TMP~
RETURN

Program: TN020.BAS
Routine name: RESTART
Abstract: Application restart location. This is where your application program takes over

Routines called: ARM.EVENTS

Variables used: AXIS.LIST@, MODE@(), X

RESTART:
COLOR 7,0
CLS
MODE@(AXIS.LIST@)=5 'This is optional. If you are testing the example and don’t want to enable any servos, comment this line out.
ARM.EVENTS 're-arm any EVENT trapping
WHILE TRUE
 'This is where the application program really starts. For this example we have a simple routine that lets you enter a MotionBASIC(R) error code to see how the system responds to various errors.
INPUT @ 1,1;" Error code ";USING " #### ";X
PRINT
PRINT " Forcing error code";X
WAIT 100
ERROR X
WEND
END

Program: TN020.BAS
Routine name: ERROR.HDLR
Abstract: React to a fault or error

Routines called: ESTOP.STATE

Variables used: AXIS.LIST@, DSP.DONE@(), FAULT@, MODE@()
STORED.ERR

ERROR.HDLR:
STORED.ERR=ERR 'remember the error code number
'A CTRL-C stops the program for debugging purposes. Normally you should comment this line out after completing your development.
IF STORED.ERR=1805 THEN CLOSE: ON ERROR GOTO 0
HALT AXIS.LIST@ 'stop all axes
'We need to wait until the axes have stopped before disabling. However if FAULT@ is true, the NO FAULT relay is open. This means the servos don’t have power anyway so we should disable 'now to prevent under-voltage and other nuisance secondary faults 'on the servos.
WAIT UNTIL DSP.DONE@(AXIS.LIST@) OR FAULT@<>{}}
MODE@(AXIS.LIST@)=0

'Not all MotionBASIC errors open the NO FAULT relay. This line
'will make sure the NO FAULT relay is opened.

IF FAULT@={} THEN FAULT@=TRUE
RESUME ESTOP.STATE
END

Program: TN020.BAS

Routine name: CLEAR.FAULTS

Abstract: Attempt to clear faults

Routines called: None

Variables used: AFAULT@, AXIS.LIST@, AXIS.SET@, DIO@()
FAULT.LIGHT, FAULT@, OTL.FWD@, OTL.REV@
STORED.ERR

CLEAR.FAULTS:
AXIS.SET@=AXIS.LIST@
OTL.FWD@=0 :OTL.REV@=0 :FAULT@=0 :FAULT@=0 :WAIT 300
DIO@(FAULT.LIGHT)=OFF
STORED.ERR=0
RETURN