

## OPL Workshop 10 - Gute Fehler, schlechte Fehler

Ulrich Krinzner, 03/2001

*In Zeiten, in denen immer mehr Freizeitprogrammierer es den Profis nachmachen und frühe Beta-Versionen noch eher "unter's Volk" streuen, sollte es zum guten Ton gehören, diese Programme im Falle eines Falles wenigstens sanft "crashen" zu lassen. Nichts ist schlimmer, als zukünftige Nutzer vorab schon 'mal mit Softresets zu beglücken oder mit Hardresets, gepaart mit Datenverlust, gehörig zu verärgern. Fehlerbehandlungsroutinen schaffen hier Linderung. Wir geben Ihnen eine Einführung.*

Wenn wir von Fehlern reden, die in irgendeiner Form einer Behandlung bedürfen, dann sind es nicht die Syntaxfehler. Diese sind zumeist einfache Schreibfehler, bei denen der Interpreter schon beim Übersetzen nicht verzeiht, dass man sich nicht an die Spielregeln hält. Das ist sowohl die fehlende Lücke zwischen Befehl und dem Argument (z.B. PRINTa\$) genauso, wie der Strukturfehler (z.B. eine IF-ELSEIF-Konstruktion ohne abschließendes ENDIF). In diesen Fällen ist aber auch ein Schaden nicht zu erwarten.

### Die schwereren Fälle

Die schwerwiegenden Dinge passieren also erst, wenn das Programm bereits irgendwie lauffähig ist. Hier werden die Fallen durch nicht deklarierte Variablen, falsch dimensionierte Stringlängen, Division durch Null und alle anderen logischen Fehler gestellt. Sitzen die Fehler in selten durchlaufenen Programmzweigen, werden sie dummerweise oft auch erst sehr spät bemerkt. Stolpert dann das Programm darüber, liefert es zwar eine verständliche Fehlermeldung, bricht aber gleichzeitig die Ausführung ab - ein meist nicht erwünschter Effekt!

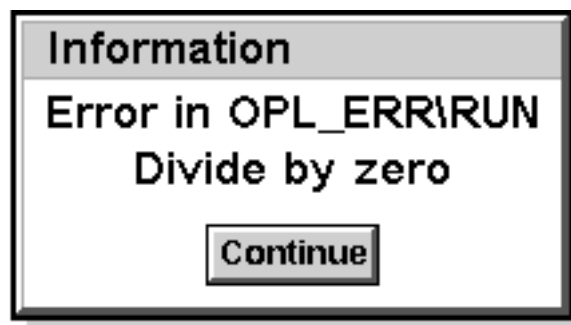


Abb.: Fehlermeldung und Programmabbruch

OPL bietet nun sowohl unter EPOC32 als auch bereits unter EPOC 16 eine Reihe von "Fehler-Befehlen". Damit lassen sich Fehlermeldungen neben der Fehlernummer auch im Klartext ausgeben, Fehlerbehandlungsroutinen ohne Programmabbruch einleiten, Fehlermeldung und Crash unterdrücken und sogar Fehler absichtlich erzeugen.

Aber aufgepaßt! Sie allein sind verantwortlich, dass Sie die mit den Befehlen ausgelösten Vorgänge auch richtig verarbeiten. Hier geschieht nichts mehr im Selbstlauf durch das System!

### Die Fehlerbehandlungsstrategie

Bei Programmen mit mehreren Prozeduren sollte jede ihre eigene, spezifisch zugeschnittene Fehlerbehandlungsroutine erhalten. Soweit die Theorie. Sie ist sicherlich formal und für kommerzielle Anwendungen auch korrekt. In der Praxis des Gelegenheitsprogrammierers steht jedoch im Vordergrund, einen Fehler ohne Programmabsturz zu überleben und korrekt signalisiert zu bekommen. Deshalb kommt man meist mit sehr wenigen oder sogar mit nur einer globalen Routine aus, die Fehlerart und Fehlerort beschreibt.

## Das Prinzip

Werfen Sie zuerst einen Blick auf den Quelltext unseres Demoprogrammes "OPL\_ERR" am Ende des Workshops. Dort befinden sich auch alle im weiteren beschriebenen Programmtexte. EPOC32 bietet zumeist erweiterte Möglichkeiten, achten Sie deshalb bitte immer auf unsere zusätzlichen Anmerkungen.

Im Programm "OPL\_ERR" wird die Fehlerbehandlung durch den Befehl "ONERR fehler::" in der "PROC run:" aktiviert. Tritt nun tatsächlich ein Fehler auf, verzweigt das Programm zum Anspringpunkt "fehler::". Dort wird die Fehlerbehandlung sofort mit "ONERR OFF" deaktiviert, um versehentliche Endlosschleifen zu vermeiden. Es folgt nun üblicherweise die gezielte Verarbeitung der Fehlerereignisse. Wir begnügen uns hier mit der Ausgabe einer Meldung durch die "PROC errprog:", die dem Programmierer bei der Fehlerbeseitigung behilflich sein soll. Es sind genau solche Meldungen, auf die die Beta-Tester stoßen und die sie dann dem Programmierer mitteilen sollen.

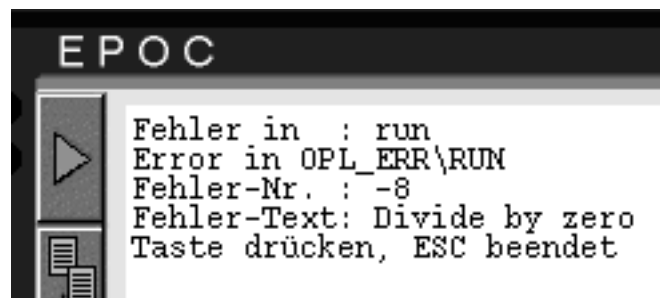


Abb.: Fehlermeldung per OPL ohne Programmabbruch

## Fehlerzentrale ERR

Im Falle eines Fehlerereignisses wird intern unverzüglich die Fehlernummer in der OPL-eigenen Variablen "ERR" abgelegt. Eine Fehlerbeschreibung im Klartext erhält man, wenn man den Befehl "PRINT ERR\$(fehlernummer)" bemüht. Da "ERR" die Fehlernummer enthält, kommt man mit der Ausgabe "PRINT ERR\$(ERR)" direkt zum Ziel.

EPOC32 bietet darüber hinaus einen weiteren Befehl, der den Fehlerort benennt (Programm- und Prozedurname). Er lautet "ERRX\$". Unter EPOC16 behilft man sich mit einer zusätzlichen Stringvariablen, der man den jeweils aktuellen Prozedurnamen übergibt. Bei uns ist das "pname\$".

## Im Kreis

Durch den auftretenden Fehler verlässt das Programm in der Prozedur "run:" die DO/UNTIL-Schleife, somit wird auch die Prozedur beendet und das Programm kehrt zur aufrufenden Prozedur "main:" zurück. Solange dort die Abbruchbedingung der DO/UNTIL-Schleife nicht erfüllt ist (Drücken der ESC-Taste), wird "run:" immer wieder aufgerufen, die Fehlerroutine neu aktiviert und das Spiel beginnt von vorn. Hoffentlich hat der Programmierer inzwischen den Fehler beseitigt ...

## An alle Fälle denken!

Neben der reinen Fehlerermittlung bietet OPL die Möglichkeit, Fehlerreaktionen zu unterdrücken. Es gibt ja schließlich nicht nur die Fehler, die man sich beim Programmieren selber erzeugt, sondern auch solche, die man eigenverantwortlich einfach mit berücksichtigen muss.

Beispiel: Ihr Programm soll ein zusätzliches Modul per "LOADM modul\$" hinzuladen. Das Modul befindet sich auf einem steckbaren Speicherbaustein, um den RAM-Hauptspeicher zu schonen. Nun wird aus irgendeinen Grunde der Speicherbaustein entfernt oder ausgetauscht. Die Folge: das Modul ist nicht verfügbar, das Programm stürzt ab und reißt vielleicht dabei wichtige Daten mit ins Grab. Dumm gelaufen!

## Fehler? Nicht jetzt!

Abhilfe schafft ein vor den eigentlichen Befehl gestellter TRAP-Befehl, kombiniert sieht das etwa so aus: "TRAP LOADM modul\$". "TRAP" verhindert für diese Programmzeile einen Crash im Fehlerfall. Der Wert für "ERR" wird trotzdem vom Betriebssystem gesetzt und sollte nun in weiser Voraussicht vom Programmierer benutzt werden, um auf den Fehler gezielt zu reagieren. Das kann eine Meldung sein (unser Beispielprogramm OPL\_TRAP unten) oder auch eine wie immer gestaltete "Auffang-Routine".

Mit "TRAP" arbeiten eine ganze Reihe von Befehlen des Datenbank- und Dateihandlings sowie Verzeichnis-, Eingabe- und Grafikbefehle zusammen. Die geeigneten Befehle findet man, genau wie die Fehlercodes, im Programmierhandbuch ([www.psion-gmbh.com/download/index.html](http://www.psion-gmbh.com/download/index.html), in English). Die Fehlercodes lassen sich aber auch am Bildschirm darstellen. Wie das geht, zeigt unser Programm ERRNUM unten.

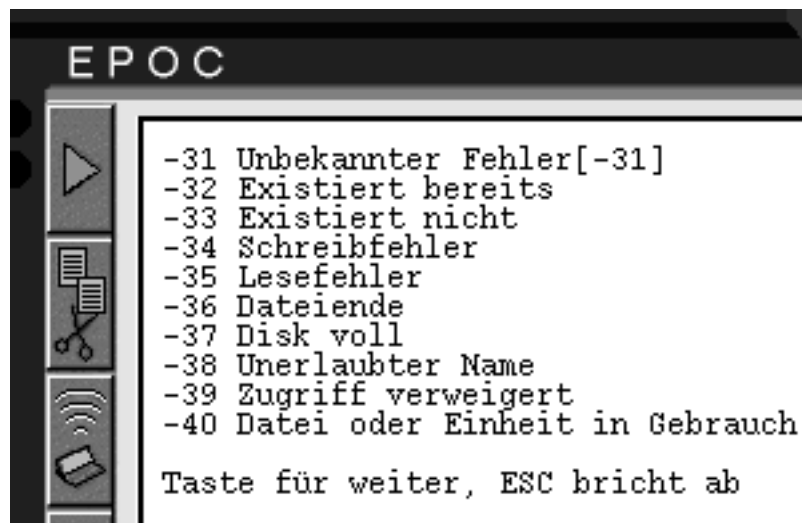


Abb.: Alle Fehlermeldungen mit eigenem Programm anzeigen lassen

## RAISE & ERR

Gelegentlich möchte man den in "ERR" befindlichen Code selber wieder auf den Standardwert Null setzen. Das ist mit Befehl "RAISE wert" möglich. Wie das geht, zeigen die Programme "RAISE1" (generell) und "RAISE2" (vereinfacht, nur für EPOC32).

"RAISE wert" ist aber nicht auf diesen Sonderfall festgelegt. Der Befehl ist in der Lage, dem System eine Fehlernummer vorzugaukeln, die anders als die "echte" ist. Die dann über "PRINT ERR\$(ERR)" ausgegebene Fehlermeldung kann für den Programmnutzer durchaus weit informativer sein, als es die Originalmeldung ist.

## Die bessere Fehlermeldung

Stellen Sie sich ein Programm vor, das Rechenoperationen anhand von Formeln ausführt, die dem Nutzer nicht bekannter sind. Die Meldung "Division durch Null" (ERR=-8) würde wohl nur Verwirrung und Hilflosigkeit stiften. Anders wäre eine Meldung "Unerlaubter Parameter" (ERR=-2) von gewisser Klarheit.

Unser unten stehendes Programm "RAISE3" zeigt ein einfaches Beispiel, in dem die sonst übliche Werte-Eingabe-Prozedur durch Übergabe eines "richtigen" und eines "falschen" Testwertes lediglich simuliert wird.

## Individuell

Haben Sie gestiegene Anforderungen, dürfen Sie sogar eigene Fehlernummern verwenden. Voraussetzung: diese dürfen nicht von OPL vorbelegt sein. Der zulässige Wertebereich liegt im positiven Bereich von 1 bis 127. Wenn Sie allerdings dazu auch einen Fehlertext haben wollen,

müssen Sie sich das selbst organisieren, denn "ERR\$(ERR)" kann nicht mehr greifen. Woher soll schließlich das Betriebssystem Ihre Fehlernummern kennen? Mit der Meldung "Unbekannter Fehler[xx]" werden Sie sich aber sicher auch nicht zufrieden geben wollen ...

Mit den beschriebenen Prinzipien und Befehlen sollten Sie in der Lage sein, Ihre Programme in "sauberer" Form an die Schar der Tester weiterzureichen, ohne sich deren Zorn zuzuziehen. Fügen Sie der Fehlermeldung neben einer netten kleinen Bitte um Rückmeldung den Programm-Namen, die Versionsnummer und auch noch Ihre E-Mail-Adresse hinzu, werden Sie vermutlich den erwarteten Informationsrücklauf bekommen.

Einmal mehr viel Spaß beim Experimentieren!

```
*****
REM Programm OPL_ERR, Prinzip Fehlerverarbeitung

PROC main:    REM Basis-Schleife
    GLOBAL esc%, a%
    GLOBAL prname$(8)    REM nur fuer EPOC16
    esc%=27
    do
        run:
    until a%=esc%
ENDP

PROC run:      REM Hauptprogrammschleife
    prname$="run" REM nur fuer EPOC16
    ONERR fehler::
    DO
        REM hier steht das Programm,
        REM und dann ein Fehler:
        print 2/0    REM Div. durch Null
    UNTIL a%=esc%
    fehler::
    ONERR OFF
    errprog:
ENDP

PROC errprog:
    BEEP 1,333
    PRINT "Fehler in  :",prname$    REM nur fuer EPOC16
    PRINT errx$    REM nur fuer EPOC32
    PRINT "Fehler-Nr. :",ERR
    PRINT "Fehler-Text: "+ERR$(ERR)
    PRINT "Taste drücken, ESC beendet"
    PRINT
    a%=GET
ENDP
*****
REM Programm OPL_TRAP, Fehler abfangen
PROC trap:
    TRAP LOADM "nix_da.opo"
    IF ERR
        PRINT "Fehler!",ERR,ERR$(ERR)
        GET
    ENDIF
    REM etc.
ENDP
*****
REM Programm RAISE1, Fehlercode ruecksetzen
PROC main:
    TRAP LOADM "nix_da.opo"
    PRINT ERR    REM ERR von LOADM
    ONERR weiter::
    RAISE 0
    weiter::
    ONERR OFF
    PRINT ERR    REM ERR = Null
    GET
ENDP
*****
```

```

REM Programm RAISE2, Fehlercode ruecksetzen/EPOC32
PROC main:
    TRAP LOADM "nix_da.opo"
    PRINT ERR      REM ERR von LOADM
    TRAP RAISE 0
    PRINT ERR      REM ERR = Null
    GET
ENDP

*****
REM Programm RAISE3, Fehlertext umdirigieren
PROC main:
    PRINT funkt:(27.0) REM "Eingabe"=27
    PRINT : GET
    PRINT funkt:(0.0)  REM "Eingabe"=0
    GET
ENDP

PROC funkt:(wert)  REM rechnet 1/x
    ONERR fehler::
    IF wert=0
        RAISE -2
    ENDIF
    RETURN 1/wert
    fehler::
    ONERR OFF
    PRINT ERR$(ERR)
    PRINT "Rechenergebnis ist falsch!"
ENDP

*****
REM Programm ERRNUM, Listet die Fehlernummern
PROC main:
    REM Fehlernummern bis -120 / EPOC16
    REM EPOC32 geht bis -134
    LOCAL n%, k%, z
    WHILE n%<120 AND k%<>27
        n%=n%+1 : z=n%/10.0
        IF z=INT(z)
            PRINT n%*(-1), ERR$(n%*(-1)) : PRINT
            PRINT "Taste für weiter, ESC bricht ab"
            k%=GET : CLS
        ELSE
            PRINT n%*(-1), ERR$(n%*(-1))
        ENDIF
    ENDWH
    PRINT : PRINT "Programmende" : PAUSE -40
ENDP
*****

```