

Debugging
Syntaxfehler
Breakpoints
Errorlogging
Utilities

Im folgenden wird von folgender genereller Programmstruktur ausgegangen:

```
SUB main()  
    ON ERROR GOTO Hell  
    REM Vorlauf/ Initialisierung  
    GLOBAL GBEFEHL$  
    \....  
    \....  
    \....  
    REM Haupt und Endlosschleife  
    Superbase.ProcessQueue = - 1  
    WHILE - 1  
        Superbase.Process()  
        IF GBEFEHL$ <> "" THEN  
            SGross$ = GBEFEHL$  
            GBEFEHL$ = ""  
            EXECUTE SGross  
        END IF  
    WEND  
  
END SUB  
  
Hell:  
  
REM allgemeine Fehlerbehandlung
```

[\[Debugging\]](#) [\[Syntaxfehler\]](#) [\[Breakpoints\]](#) [\[Errorlogging\]](#) [\[Utilities\]](#)

Syntaxfehler

Debugging
Syntaxfehler
Breakpoints
Errorlogging
Utilities

Ausserhalb von Process() wird bei einem Syntaxfehler der Programmeditor geöffnet und der Cursor in die fehlerauslösende Zeile gestellt. Bei Anwendungen, die sich im Process() befinden, ist das leider nicht der Fall. Im einfachsten Fall lässt sich folgende Methode benutzen:

```
Hell:
REM - Fehlerbehandlung für allerlei bekannte Fehler:
SELECT CASE Superbase.LastError
CASE 3
    \.....
CASE 9
    \.....
END CASE

REM bei nicht abgefangenen Fehlern:
EDIT CURRENT:DEBUG ASK:RESUME
```

Der Editor wird unabhängig von der Einstellung von Superbase.AutoEdit geöffnet, der Debugger wird aufgerufen. Beim Klick auf *Zeile* wird die Fehlerauslösende Zeile angezeigt. Der Wert sämtlicher lokalen und globalen Variablen zum Zeitpunkt des Fehlers ist im Debugger sichtbar.

Der Aufruf des Debuggers muss natürlich vor Auslieferung des Programms gelöscht werden.

Um sicherzugehen, dass die Debuggingroutine nicht beim Kunden/Anwender erscheint, empfiehlt sich eine etwas erweiterte Methode:

```
SUB main()

    ON ERROR GOTO Hell
    REM Vorlauf/ Initialisierung
    GLOBAL GBEFEHL$,GDEBUG%%
    DIM Wert$
    REM Debuggingflag aus SB30.INI einlesen:
    REGISTER "Kernel.exe","GetPrivateProfileString","HCCCCFHC"
    ret%% = CALL ("GetPrivateProfileString","Debugging","DebugFlag","0",Wert$,4000,"sb30.ini")
    REGISTER CLEAR "GetPrivateProfileString"
    GDEBUG%% = VAL (Wert$)
    \.....
    \.....
    \.....
    REM Haupt und Endlosschleife
    Superbase.ProcessQueue = - 1
    WHILE - 1
        Superbase.Process()
        IF GBEFEHL$ <> "" THEN
            SGross$ = GBEFEHL$
            GBEFEHL$ = ""
            EXECUTE SGross
        END IF
    END WHILE
END SUB
```

WEND

Hell:

```
REM - Fehlerbehandlung für allerlei bekannte Fehler:  
SELECT CASE Superbase.LastError  
CASE 3  
  \.....  
CASE 9  
  \.....  
END CASE
```

REM bei nicht abgefangenen Fehlern:

```
IF GDEBUG%% THEN  
  IF GDEBUG%%=-2 THEN Superbase.Model.Visible=-1  
  EDIT CURRENT:DEBUG ASK:RESUME  
END IF
```

Auf diese Weise wird der Debugger nur aufgerufen, wenn die lokale SB30.INI um die Section *Debugging* erweitert wurde und der Schlüssel *DebugFlag* mit -1 besetzt ist. Wenn GDEBUG%% -2 ist, wird zusätzlich der Property-Browser angezeigt. Das empfiehlt sich besonders bei Fehlern, die mit dem Objektmodell zusammenhängen, da der normale Debugger die Objekteigenschaften nicht anzeigen kann.

[\[Debugging\]](#) [\[Syntaxfehler\]](#) [\[Breakpoints\]](#) [\[Errorlogging\]](#) [\[Utilities\]](#)



Debugging
Syntaxfehler
Breakpoints
Errorlogging
Utilities

Um logische Fehler oder ungeplantes Programmverhalten zu debuggen, ist es praktisch, an einer geeigneten Stelle den Debugger aufzurufen um so den Programmfortschritt und die Variablen zu überprüfen

Das geht natürlich am einfachsten, indem man

```
EDIT CURRENT:DEBUG ASK
```

an der entsprechenden Stelle einfügt. In Schleifen, die oft durchlaufen werden, kann das allerdings lästig sein, wenn man nur ein paar Durchläufe checken und dann das Programm fortsetzen möchte. Auch besteht natürlich die Gefahr, dass ein versehentlich übriggebliebener Breakpoint später die Anwender verunsichert.

Auch hier lässt sich über die SB30.INI ein bisschen Luxus und Sicherheit einfügen:

```
SUB Bugger()
  DIM ret%%,Wert$,BugFlag%
  REGISTER "Kernel.exe","GetPrivateProfileString","HCCCFHC"
  ret%% = CALL ("GetPrivateProfileString","Debugging","BreakPointFlag","0",Wert$,4000,"sb30.ini")
  REGISTER CLEAR "GetPrivateProfileString"
  BugFlag% = VAL (Wert$)
  IF BugFlag% THEN
    IF BugFlag% = - 2 THEN Superbase.Model.Visible = - 1
    EDIT CURRENT :Superbase.DebugDialog = - 1
  END IF
END SUB
```

Mit einem eingefügten CALL Bugger() lässt sich ein Breakpoint konfigurieren, der während des laufenden Programms an und ausgeschaltet werden kann. Auch hier lässt sich bei Bedarf wieder der Property-Browser zusätzlich einschalten..

[\[Debugging\]](#) [\[Syntaxfehler\]](#) [\[Breakpoints\]](#) [\[Errorlogging\]](#) [\[Utilities\]](#)

Errorlogging

Debugging
Syntaxfehler
Breakpoints
Errorlogging
Utilities

Das Debugging von Anwendungen, die bereits installiert sind und zu denen kein direkter Zugang besteht, ist besonders mühselig, da viele Probleme nur unter spezifischen Bedingungen auftreten, die auf den Entwicklungsrechnern meistens nicht vorliegen. Eine kleine Hilfe ist ein Errorlog.

```

Superbase.Error(138) = 0'Zugriff verweigert. Fehler beim gemeinsamen Zugriff (sharing violation) (138)
Superbase.Error(205) = 0'Variable bereits deklariert (205)
IF procArr$ = "" THEN '      *** damit nicht bei 2 Fehlern in einer Prozedur das Array mehrmals dimensioniert
wird....
    DIM procArr$(32)
    procArr$ = "gesetzt"
END IF
FILLARRAY procArr$,7
OPEN "errlog.txt" FOR APPEND
IF Superbase.LastError <> 138 THEN
    ? " _____"
    ? "Datum:", DATE$( TODAY , "dd.mm.yyyy")
    ? "Zeit:", TIME$( NOW , "hh.mm.ssss")
    ? "Benutzer:"; USERNAME
    ? "Fehler: ", ERR$(Superbase.LastError)
    ? "FehlerNr:", STR$(Superbase.LastError, "999")
    ? "Zusatz1:", ERR$( 0)
    ? "Zusatz2:", ERR$( - 1)
    IF NOT IS (Superbase.Window, NOTHING ) THEN ? "Fenster:", Superbase.Window.Name
    ? "Formular: ", FORM
    ? "Datei:", FILE
    errZaehler% = 0'      *** falls mehrmals aus gleicher Prozedur angesprungen
    WHILE ProcArr$(errZaehler%) > ""
        ? "Prozedur" + STR$( errZaehler%, "00") + ":", ProcArr$(errZaehler%)
        errZaehler% = errZaehler% + 1
    WEND
    ? "Lokale Variablen:"
    ? MEMORY
    ?
    ? "Globale Variablen:"
    ? MEMORY GLOBAL
CLOSE OUTPUT
END IF

```

Das Errorlog erzeugt eine Datei, in der sämtliche nicht abgefangenen Fehler protokolliert werden. Festgehalten werden:

- Datum und Zeit
- Benutzer
- Fehlernummer, Text und Fehlerzusatzinformationen
- Die Prozedur, in der der Fehler aufgetreten ist
- Die Prozedurkette, die zum Aufruf der betroffenen Prozedur geführt hat
- Lokale Variablen
- Globale Variablen

Die Ausgabe der Variablen lässt das Errorlog u.U. sehr schnell wachsen. Von Fall zu Fall sollte man erwägen, die entsprechenden Zeilen auszukommentieren.



Debugging
Syntaxfehler
Breakpoints
Errorlogging
Utilities

Um das Debuggen noch etwas zu vereinfachen, lässt sich - wieder in Abhängigkeit von einem INI-Eintrag- ein nur für die Entwicklung gedachtes [Zusatzmenü einbinden](#), das Zugriff auf kleine Utility-Funktionen bietet.

- Eine Kommandozeile
Hier lassen sich Programmbefehle eingeben.
- Neu Laden eines Programm-Moduls
Ein Programm-Modul kann neu geladen werden, um eine Änderung zu testen, ohne dass das Programm neu gestartet werden muss
- Property-Browser anzeigen
- Umgebung zurücksetzen
Standardmenu und Iconbar werden gesetzt
- Fenstermenü
falls noch nicht vorhanden, werden die Standard -Fenstermenu Items in das Menu Fenster eingefügt.

Dazu wird in SUB main zunächst eine weitere Globale eingefügt:

```

REM Vorlauf/ Initialisierung
GLOBAL GBEFEHL$,GDEBUG%%
DIM Wert$
REM Debuggingflag aus SB30.INI einlesen:
REGISTER "Kernel.exe", "GetPrivateProfileString", "HCCCFHC"
ret%% = CALL ("GetPrivateProfileString", "Debugging", "DebugFlag", "0", Wert$, 4000, "sb30.ini")
GDEBUG%% = VAL (Wert$)
ret%% = CALL ("GetPrivateProfileString", "Debugging", "DebugMenu", "0", Wert$, 4000, "sb30.ini")
GDEBUGMENU%%=VAL (Wert$)
REGISTER CLEAR "GetPrivateProfileString"

```

\

Nach der Definition des Anwendungsmenüs wird dann das Debuggingmenu eingehängt:

```

SUB MNSERVER ( )
    DIM Bar AS MenuBar,Level0 AS MenuItem,Level1 AS MenuItem

    SET Bar = MenuBars.Add("MNSERVER", "MenuBar")

    SET Level0 = Bar.Add("&Datei", "Menu")
    Level0.HelpText = ""

    SET Level1 = Level0.Add("&Beenden", "MenuItem")
    Level1.Procedure = "Exit"
    Level1.HelpText = "Server herunterfahren"

```

```
CALL AddDebugMenu(Superbase.Menubars.Mnserver)
Superbase.Menubars.Mnserver.SetActive()
END SUB

SUB AddDebugMenu(Bar AS MenuBar)
DIM Level0 AS Menu,Level1 AS MenuItem,Level2 AS MenuItem
IF GDEBUGMENU%% THEN
  IF NOT Bar.Exists("&Debugging") THEN
    SET Level0 = Bar.Add("&Debugging", "Menu")

    SET Level1 = Level0.Add("&Kommando", "MenuItem")
    Level1.Procedure = "dbgCmd"
    level1.ShortCut = "ALT+F1"

    SET Level1 = Level0.Add("&Reload", "MenuItem")
    Level1.Procedure = "Reload"

    SET Level1 = Level0.Add("&Model", "MenuItem")
    Level1.Procedure = "ShowModel"

    SET Level1 = Level0.Add("&Umgebung zurücksetzen", "MenuItem")
    Level1.Procedure = "SetBack"

  END IF
  IF NOT Bar.Exists("&Fenster") THEN
    SET Level0 = Bar.Add("&Fenster", "Menu")
    Level0.HelpText = "Die Fenster neu anordnen oder das angegebene Fenster aktivieren"

    SET Level1 = Level0.Add("&Vertikal teilen", "MenuItem")
    Level1.Procedure = "SplitVert"
    Level1.HelpText = "Die aktuelle Fensterfläche vertikal teilen"

    SET Level1 = Level0.Add("&Horizontal teilen", "MenuItem")
    Level1.Procedure = "SplitHorz"
    Level1.HelpText = "Die aktuelle Fensterfläche horizontal teilen"

    SET Level1 = Level0.Add("&Neues Fenster", "MenuItem")
    Level1.Procedure = "NewWindow"

    SET Level1 = Level0.Add("&Schließen", "MenuItem")
    Level1.Caption = "&Schließen"
    Level1.Procedure = "CloseWindow"
    Level1.Shortcut = "Strg+F4"
```

```

SET Level1 = Level0.Add("Ne&beneinander", "MenuItem")
Level1.Procedure = "TileWindow"
Level1.Shortcut = "Umschalt+F4"

```

```

SET Level1 = Level0.Add("Über&lappend", "MenuItem")
Level1.Caption = "Über&lappend"
Level1.Procedure = "CascadeWindow"
Level1.HelpText = "Die Fenster überlappend anordnen"
Level1.Shortcut = "Umschalt+F5"

```

```

SET Level1 = Level0.Add("Symbole &anordnen", "MenuItem")
Level1.Procedure = "ArrangeIcons"

```

```

END IF

```

```

END IF

```

```

END SUB

```

SUB dbgCmd()

```

DIM msga$,msgb$,Antwort%%,Befehl$
msga$ = "Bitte geben Sie einen Programmbefehl ein."
msgb$ = ""
REQUEST msga$,msgb$,4,Antwort%%,Befehl$,72
IF Antwort%% THEN
    Superbase.ErrorHandlingPush()
    SET ERROR OFF ALL
    Superbase.ClearLastError()
    EXECUTE Befehl$

```

```

IF Superbase.LastError > 0 THEN
    REQUEST LEFT$ ( ERR$ (SuperBase.Lasterror),72)
END IF
Superbase.ErrorHandlingPop()

```

```

END IF

```

```

END SUB

```

SUB Reload()

```

Superbase.ErrorHandlingPush()
Superbase.Error(269) = 0
DIM msga$,msgb$,Antwort%%,ProgName$,BufName$
msga$ = "Bitte das zu ladende Programm aussuchen."
msgb$ = ""
ProgName$ = "*.sbp"

```

```
REQUEST msga$,msgb$,26,Antwort%%,ProgName$
IF Antwort%% THEN
  BufName$ = FN root(ProgName$)
  IF PROGRAMFILE (BufName$) THEN CLOSE PROGRAMFILE BufName$
  LOAD Progname$, NEW
  IF Superbase.LastError = 269 THEN
    msga$ = "Programm wird gerade verwendet."
    msgb$ = ""
    REQUEST msga$,msgb$,139,Antwort%%
  END IF
END IF
Superbase.ErrorhandlingPop()
END SUB
```

```
SUB showModel()
  Superbase.Model.Visible = - 1:DEBUG ASK
END SUB
```

```
SUB SetBack()
  Superbase.MenuBars.Restore()
  Superbase.IconBars.Restore()
END SUB
```

[\[Debugging\]](#) [\[Syntaxfehler\]](#) [\[Breakpoints\]](#) [\[Errorlogging\]](#) [\[Utilities\]](#)