

14 Fehlerbehandlung

Die Fehlerbehandlung hat zwei wichtige Funktionen: Erstens soll sie gewährleisten, dass eine Anwendung auch noch nach dem Auftreten eines Laufzeitfehlers stabil weiterläuft und nicht etwa abstürzt. Zweitens soll sie Informationen bereitstellen, um den Fehler zu analysieren und zu beheben. In diesem Buch lernen Sie zwei Möglichkeiten der Fehlerbehandlung kennen. Die klassische Variante implementieren Sie direkt in den Prozeduren der Anwendung. Der Aufwand ist nicht unerheblich: Sie müssen dazu jede einzelne Prozedur mit einigen zusätzlichen Zeilen ausstatten und außerdem alle Zeilen mit Zeilennummern versehen. Auf diese Weise können Sie die Fehlerinformationen um die Angabe der Zeile erweitern, in welcher der Fehler aufgetreten ist.

Die zweite Möglichkeit ist die Fehlerbehandlung mit *vbWatchdog*. Dabei handelt es sich um eine Art DLL, die sich in die Fehlerbehandlung von VBA einklinkt und so Fehler behandelt, ohne dass Sie der entsprechenden Prozedur überhaupt eine Fehlerbehandlung hinzufügen müssen.

14.1 Klassische Fehlerbehandlung

Die klassische Fehlerbehandlung sieht vor, beim Auftreten eines Laufzeitfehlers nicht die in VBA eingebaute Standardfehlermeldung anzuzeigen, sondern eine eigene Fehlerbehandlung zu implementieren. Nehmen wir die folgende, einfache Prozedur als Beispiel, die sich in der Beispieldatenbank im Modul *mdlFehlerbehandlung_Klassisch* befindet:

```
Public Sub Beispielfehler()  
    Debug.Print 1 / 0  
End Sub
```

Das Ausführen dieser Prozedur löst den Fehler aus Abbildung 14.1 aus.

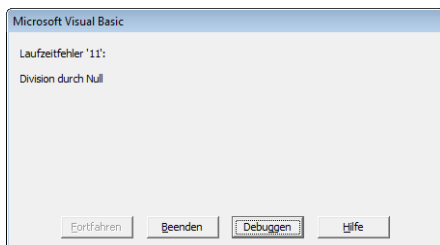


Abbildung 14.1: Fehler beim Dividieren durch die Zahl 0

Mit den eingebauten Fehlermeldungen kann der Benutzer erstens je nach Fehler wenig anfangen. Außerdem wird die Ausführung des Codes unterbrochen und VBA löscht den Inhalt von

Kapitel 14 Fehlerbehandlung

Variablen. Dies kann insbesondere kritisch sein, wenn Sie Objekte mit globalen oder modulweit deklarierten Variablen referenzieren – diese sind nach dem Auftreten eines unbehandelten Fehlers anschließend leer.

Das Voranstellen der folgenden Anweisung sorgt dafür, dass die Fehlermeldung ausbleibt und die Variablen ihre Werte behalten:

```
On Error Resume Next
```

Allerdings erfahren weder Nutzer noch Entwickler vom Auftreten des Fehlers. Um es kurz zu machen, finden Sie hier die standardmäßig verwendete Fehlerbehandlung inklusive Zeilennummerierung:

```
Public Sub BeispielfehlerMitFehlerbehandlung()  
10     On Error GoTo Fehler  
20     Debug.Print 1 / 0  
Ende:  
30     On Error Resume Next  
       'Hier finale Anweisungen  
40     Exit Sub  
Fehler:  
50     ErrNotify Err, "mdlFehlerbehandlung_Klassisch", _  
       "BeispielfehlerMitFehlerbehandlung"  
60     Resume Ende  
End Sub
```

Diese Prozedur enthält gegenüber der vorherigen Variante ohne Fehlerbehandlung die folgenden Erweiterungen:

- » Die Anweisung *On Error Goto Fehler* sorgt dafür, dass die Prozedur nach dem Auftreten eines Fehlers weiter unten an der mit *Fehler*: gekennzeichneten Stelle fortgeführt wird.
- » Unter *Fehler*: wird die Prozedur *ErrNotify* aufgerufen, wobei ein Verweis auf das *Err*-Objekt mit den Fehlerinformationen sowie der Name des Moduls und der Prozedur mit dem Fehler übergeben werden. Das *Err*-Objekt liefert mit seinen Eigenschaften *Number* und *Description* eine Fehlernummer und eine Beschreibung.
- » Nach dem Aufruf von *ErrNotify* wird die Prozedur mit dem Teil hinter der Marke *Ende*: fortgeführt. Hier bringen Sie zum ordnungsgemäßen Abschluss der Prozedur nötige Anweisungen unter, etwa zum Schließen von zuvor geöffneten Dateien oder zum Leeren von Objektvariablen.
- » Die hinter der Markierung *Ende*: angeführten Anweisungen werden auch bei fehlerfreiem Verlauf der Prozedur erreicht. Vor der Markierung *Fehler*: befindet sich jedoch eine *Exit Sub*-Anweisung, damit die Prozedur bei fehlerfreier Ausführung nicht mit dem Auslösen der Fehlerbehandlung beendet wird.

Klassische Fehlerbehandlung

- » Schließlich finden Sie noch die Zeilennummerierungen vor. Diese werden für alle Zeilen angelegt, die ausgeführt werden. Es gibt ein paar Ausnahmen, zum Beispiel die Prozedurköpfe, Deklarationszeilen und die *Case*-Zeilen in *Select Case*-Anweisungen.

Um die *ErrNotify*-Prozedur kümmern wir uns weiter unten. Zunächst interessiert uns, wie wir eine Fehlerbehandlung ohne allzugroßen Aufwand zu allen Routinen der Anwendung hinzufügen – je nach Umfang der Anwendung kann dies eine ganze Menge sein.

Die gute Nachricht ist: Das Hinzufügen der für die Fehlerbehandlung nötigen Codezeilen und das Nummerieren der Routinen kostet uns schlappe zwei Mausklicks. Dazu müssen Sie jedoch die bereits erwähnte Software *MZ-Tools* installiert haben.

Wenn dies der Fall ist und Sie die im Download befindliche Datei *MZTools3VBA.ini* im entsprechenden Verzeichnis gespeichert haben, brauchen Sie im VBA-Editor nur noch die Einfügemarke in der mit der Fehlerbehandlung auszustattenden Prozedur zu platzieren und auf die Schaltfläche *Fehlerbehandlung hinzufügen* zu klicken (siehe Abbildung 14.2).

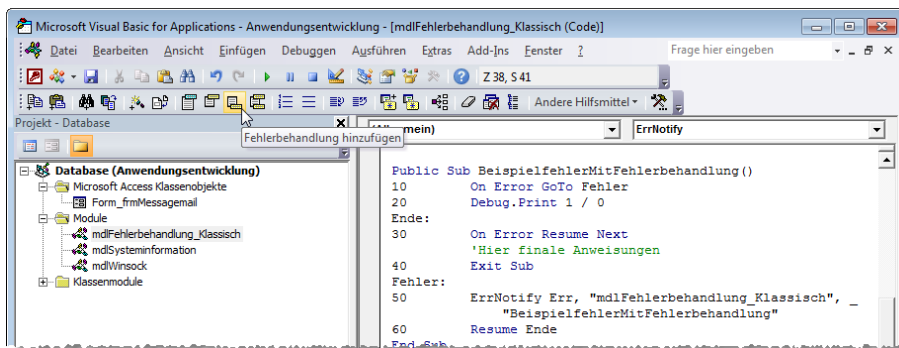


Abbildung 14.2: Hinzufügen der Fehlerbehandlung per Mausklick

Gleich zwei Schaltflächen daneben finden Sie die Schaltfläche *Zeilennummern hinzufügen* – Sie ahnen bereits, welche Aufgabe das Betätigen dieser Schaltfläche für Sie erledigt. Mit der Schaltfläche *Zeilennummern entfernen* werden Sie die Zeilennummern wieder los, wenn Sie beispielsweise den Code ändern oder erweitern möchten.

14.1.1 Fehlermeldung anzeigen

Die folgende einfache Variante der Routine *ErrNotify* gibt die Meldung aus Abbildung 14.3 aus:

```
Sub ErrNotify(AErr As VBA.ErrorObject, strModule As String, strProc As String)
    MsgBox "Fehler in Modul " & strModule & ", Routine " & strProc _
        & " in Zeile " & Er1 & "." & vbCrLf _
        & "Fehlermeldung: " & Err.Description & vbCrLf & "Fehlernummer: " & Err.Number
End Sub
```

Kapitel 14 Fehlerbehandlung

Die Prozedur nimmt einen Verweis auf das durch den Fehler gefüllte *Err*-Objekt entgegen sowie den Namen des Moduls und der Prozedur. Das *Err*-Objekt liefert mit den beiden Eigenschaften *Number* und *Description* die Fehlernummer und die Beschreibung. Die Zeilennummer ermittelt die nicht dokumentierte *Erl*-Funktion. Diese Informationen werden zu einer Zeichenkette zusammengefasst und per *MsgBox*-Anweisung ausgegeben.

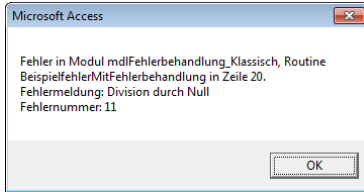


Abbildung 14.3: Beispielfehlermeldung mit Modul, Routine, Zeile, Meldung und Fehlernummer

14.1.2 Fehlermeldung per E-Mail versenden

Wenn die Fehlermeldung beim Entwickler landen soll, damit dieser den Fehler gleich reproduzieren und beheben kann, versenden Sie die notwendigen Informationen am besten gleich automatisch per E-Mail. Allerdings sollten Sie den Benutzer vorab darüber informieren, welche Daten nun an den Entwickler geschickt werden. Dies erledigen Sie, indem Sie diese Daten in einem Formular anzeigen (siehe Abbildung 14.4) und den Benutzer die E-Mail per Mausklick absenden lassen.

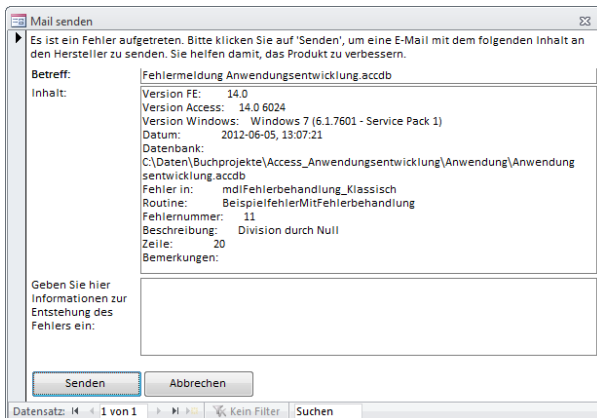


Abbildung 14.4: Formular zum Absenden einer Fehlermeldung an den Entwickler der Software

Das Textfeld des Formulars füllt die *ErrNotify*-Prozedur. Diese ist ähnlich aufgebaut wie die vorherige Variante, die lediglich ein Meldungsfenster mit den Fehlerinformationen anzeigt. Allerdings ermittelt diese Prozedur neben den übergebenen Fehlerinformationen noch einige weitere Daten. Dazu gehört die Version der Anwendung, die mit der *DLookup*-Funktion

Fehlerbehandlung mit vbWatchdog

aus der Tabelle *tblOptionen* ausgelesen wird. Dahinter finden Sie eine auskommentierte Zeile, welche die Version des Backends hinzufügen würde. Die Access-Version ermittelt die nicht dokumentierte Funktion *SysCmd* mit dem Wert *715* als Parameter. Für die Ermittlung der Windows-Version ist eine benutzerdefinierte Funktion namens *GetWindowsVersion* nötig, die Sie im Modul *mdlSysteminformation* finden. Eine ausführliche Beschreibung sparen wir uns an dieser Stelle. Danach folgen die mit den Parametern der *ErrNotify*-Prozedur übergebenen Informationen:

```
Sub ErrNotify(AErr As VBA.ErrObject, strModule As String, strProc As String, _
    Optional strRemarks As String)
    Dim strMessage As String
    Dim strErrNumber As String
    Dim strErrDescription As String
    Dim strEr1 As String
    strErrNumber = AErr.Number
    strErrDescription = AErr.Description
    strEr1 = Er1
    strMessage = strMessage & "Version FE:          " & _
        & Nz(DLookup("Version_FE", "tblOptionen"), "") & vbCrLf
    strMessage = strMessage & "Version BE:          " & _
        & Nz(DLookup("Version_BE", "tblOptionen_BE"), "") & vbCrLf
    strMessage = strMessage & "Version Access:      " & _
        & Access.Version & " " & SysCmd(715) & vbCrLf
    strMessage = strMessage & "Version Windows:    " & GetWindowsVersion & vbCrLf
    strMessage = strMessage & "Datum:              " & _
        & Format(Now, "yyyy-mm-dd, hh:nn:ss") & vbCrLf
    strMessage = strMessage & "Datenbank:          " & CodeDb.name & vbCrLf
    strMessage = strMessage & "Fehler in:          " & strModule & vbCrLf
    strMessage = strMessage & "Routine:            " & strProc & vbCrLf
    strMessage = strMessage & "Fehlernummer:      " & strErrNumber & vbCrLf
    strMessage = strMessage & "Beschreibung:      " & strErrDescription & vbCrLf
    strMessage = strMessage & "Zeile:              " & strEr1 & vbCrLf
    strMessage = strMessage & "Bemerkungen:       " & strRemarks & vbCrLf
    DoCmd.OpenForm "frmMessageMail", OpenArgs:=strMessage, WindowMode:=acDialog
End Sub
```

14.2 Fehlerbehandlung mit vbWatchdog

Wenn Sie diesen relativ aufwendigen Weg nicht gehen möchten, können Sie ein paar Euro investieren und sich *vbWatchdog* von Wayne Philips zulegen. Damit erhalten Sie ein COM-Add-In, das Sie jedoch nur zum Ausstatten der Zielanwendung mit der Fehlerbehandlungsfunktionalität

Kapitel 14 Fehlerbehandlung

benötigen. Die Beispielanwendung ist bereits damit ausgestattet, dafür fallen für den Benutzer keine Kosten an.

Um *vbWatchdog* zu Ihrer Anwendung hinzuzufügen, wählen Sie nach der Installation des COM-Add-Ins einfach den Menüeintrag *Add-Ins|vbWatchdog|Add vbWatchdog to this project* aus (siehe Abbildung 14.5).

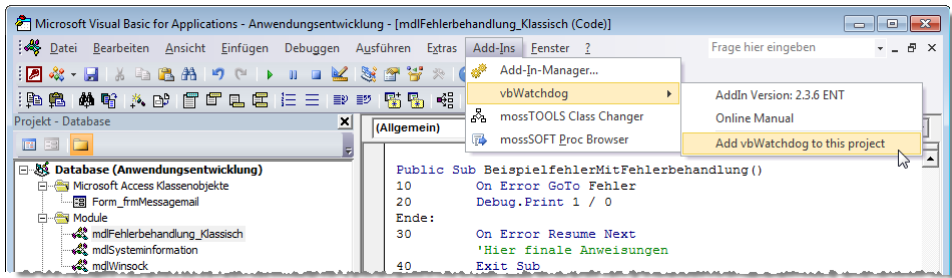


Abbildung 14.5: Hinzufügen von *vbWatchdog* zum aktuellen Projekt

Danach finden Sie im Projekt-Explorer vier neue Klassenmodule vor, welche die komplette Funktion von *vbWatchdog* enthalten (siehe Abbildung 14.6).

Wenn Sie Ihre Anwendung inklusive dieser vier Klassenmodule weitergeben, benötigen Sie keine weiteren Dateien wie beispielsweise eine DLL.

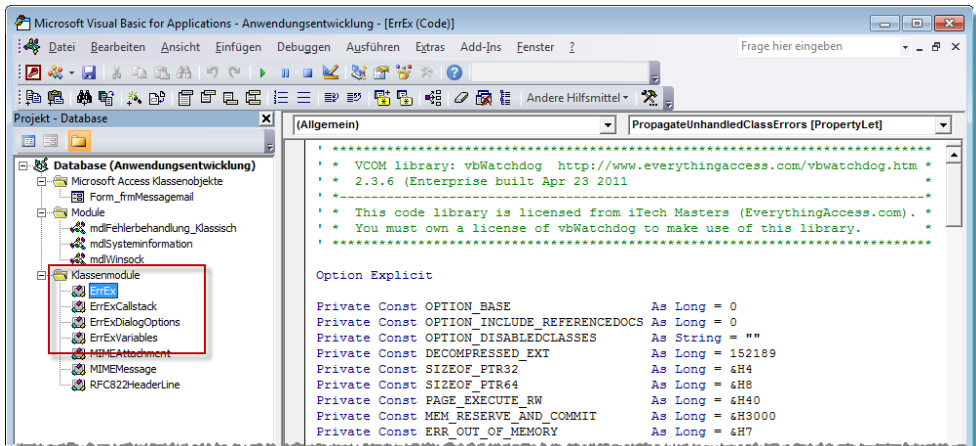


Abbildung 14.6: *vbWatchdog* fügt dem VBA-Projekt vier Klassenmodule hinzu.

Fehlerbehandlung aktivieren

Damit die Fehlerbehandlung funktioniert, müssen Sie diese lediglich zu aktivieren. Dies erledigen Sie im einfachsten Fall mit einem Einzeiler:

```
ErrEx.Enable ""
```

Diese Anweisung bringen Sie beispielsweise in der Ereignisprozedur unter, die durch das Laden des Startformulars *frmStart* ausgelöst wird:

```
Private Sub Form_Load()  
    Call ErrEx.Enable("")  
End Sub
```

Danach brauchen Sie sich nicht mehr um die Fehlerbehandlung zu kümmern: *vbWatchdog* zeigt beim Auftreten eines jeden Laufzeitfehlers eine entsprechende Meldung an. Setzen Sie beispielsweise einmal die folgende Anweisung im Direktfenster des VBA-Editors ab:

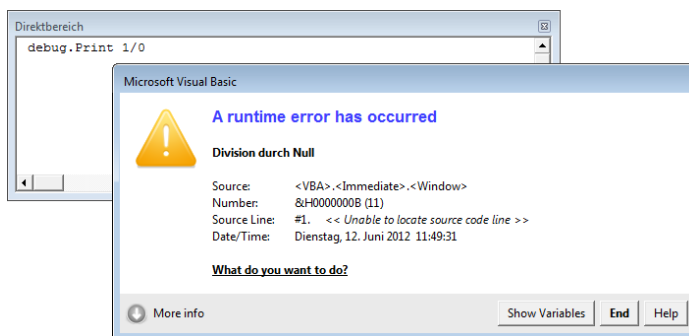


Abbildung 14.7: Eine einfache Fehlermeldung

Mit der folgenden Anweisung deaktivieren Sie *vbWatchdog* (obwohl dies in der Regel nicht wünschenswert ist):

```
ErrEx.Disable
```

14.2.1 Eigene Fehlerbehandlung

Wenn Sie möchten, dass *vbWatchdog* Ihre eigene Fehlerbehandlungsroutine aufruft, geben Sie den Namen dieser Prozedur als Parameter der *Enable*-Methode des *ErrEx*-Objekts an, also beispielsweise so:

```
Private Sub Form_Load()  
    Call ErrEx.Enable("")  
End Sub
```

Diese Prozedur sieht in einem einfachen Fall so aus:

```
Public Sub ErrNotify_1()  
    MsgBox "Error"  
End Sub
```

Kapitel 14 Fehlerbehandlung

Wenn Sie nun an beliebiger Stelle, also im Code oder auch im Direktfenster, eine fehlerhafte Zeile wie `Debug.Print 1/0` aufrufen, wird zunächst die in Ihrer eigenen Methode angegebene Meldung angezeigt und dann die Meldung von `vbWatchdog`.

`vbWatchdog` temporär deaktivieren

Wenn Sie eine eigene Fehlerbehandlung einsetzen möchten, um beispielsweise gezielt auf spezielle Fehler zu reagieren, können Sie `vbWatchdog` genau so umgehen, wie Sie es üblicherweise mit den eingebauten Fehlermeldungen des VBA-Editors tun würden. Dabei gibt es verschiedene Varianten. Bei der ersten aktivieren Sie `vbWatchdog` ohne individuelle Fehlerroutine, wodurch ein Laufzeitfehler die Standardmeldung von `vbWatchdog` anzeigt:

```
ErrEx.Enable ""  
Debug.Print 1 / 0
```

Wenn Sie `vbWatchdog` ohne eigene Fehlerroutine aktivieren und dann nach `On Error Resume Next` einen Fehler auslösen, wird der Fehler schlicht übergangen:

```
ErrEx.Enable ""  
On Error Resume Next  
Debug.Print 1 / 0
```

Wenn Sie `vbWatchdog` mit einer individuellen Fehlermeldung aktivieren, löst der Laufzeitfehler zunächst die benutzerdefinierte Fehlerbehandlung aus und zeigt dann die `vbWatchdog`-Standardmeldung an:

```
ErrEx.Enable "ErrNotify_1"  
Debug.Print 1 / 0
```

Schließlich gibt es noch die Möglichkeit, `vbWatchdog` mit einer benutzerdefinierten Fehlerbehandlung zu aktivieren und dann die Fehlerbehandlung mit `On Error Resume Next` außer Kraft zu setzen. In diesem Fall wird die benutzerdefinierte Fehlerbehandlung ausgelöst, aber nicht die Standardmeldung von `vbWatchdog` angezeigt:

```
ErrEx.Enable "ErrNotify_1"  
On Error Resume Next  
Debug.Print 1 / 0
```

14.2.2 Benutzerdefinierte Fehlermeldung

Das Ziel ist es, mit `vbWatchdog` erstens eine ansprechende und aussagekräftige Fehlermeldung zu generieren und zweitens dem Benutzer die Möglichkeit zu geben, eine E-Mail mit Informationen zum Fehler an den Entwickler der Anwendung zu senden.

Die Fehlermeldung wird im HTML-Format zusammengestellt und verwendet Platzhalter, die zur Laufzeit mit den entsprechenden Fehlerinformationen gefüllt werden. Die Fehlermeldung pas-

Fehlerbehandlung mit vbWatchdog

sen Sie mit einer Reihe von Eigenschaften an, die das Objekt *DialogOptions* des *ErrEx*-Objekts zur Verfügung stellt. Grundsätzlich beeinflussen Sie mit diesen Methoden das Aussehen der Elemente des Standarddialogs. Wenn Sie beispielsweise alle vorhandenen Schaltflächen entfernen und eine eigene Schaltfläche hinzufügen möchten, welche die fehlerhafte Prozedur beendet, verwenden Sie die folgenden Anweisungen beim Initialisieren des *ErrEx*-Objekts:

```
ErrEx.Enable ""  
With ErrEx.DialogOptions  
    .RemoveAllButtons  
    .AddButton "Prozedur beenden", BUTTONACTION_ONERROREXITPROCEDURE  
End With
```

Die vorhandenen HTML-Texte für die einzelnen Bereiche können Sie mit den vier Eigenschaften *HTML_MainBody*, *HTML_MoreInfoBody*, *HTML_CallStackItem* und *HTML_VariableItem* ausgeben. Den HTML-Text für den Haupttext erhalten Sie mit folgender Anweisung, beispielsweise im Direktfenster ausgeführt:

```
Debug.Print ErrEx.DialogOptions.HTML_MainBody
```

Der HTML-Code sieht so aus:

```
<font face=Arial size=13pt color="#4040FF"><b>A runtime error has occurred</b></font><br><br><b><ERRDESC></b><br><br>Source: |<SOURCEPROJ>. <SOURCEMOD>. <SOURCEPROC><br>Number: |&H<ERRNUMBERHEX> (<ERRNUMBER>)<br>Source Line: |#<SOURCELINENUMBER>.  
<i><SOURCELINECODE></i><br>Date/Time: |<ERRDATETIME><br><br><b><u>What do you want to do?</u></b>
```

Interessant hierbei sind die verschiedenen Platzhalter. Diese haben folgende Bedeutung:

- » **<ERRDESC>**: Fehlerbeschreibung
- » **<ERRNUMBER>**: Fehlernummer
- » **<ERRNUMBERHEX>**: Fehlernummer (hexadezimal)
- » **<ERRDATETIME>**: Datum und Zeit
- » **<SOURCEPROJ>**: Projektname
- » **<SOURCEMOD>**: Modulname
- » **<SOURCEPROC>**: Prozedurname
- » **<SOURCELINENUMBER>**: Zeilennummer
- » **<SOURCELINECODE>**: Inhalt der Zeile
- » **<CALLSTACK>**: Liste der Prozeduraufrufe

Das Pipe-Zeichen (|) entspricht einem Tabulatorzeichen.

Kapitel 14 Fehlerbehandlung

Wenn Sie im einfachsten Fall einfach nur die aktuelle Version des Fehlerdialogs ins Deutsche übersetzen möchten, geben Sie die Werte der Eigenschaften *HTML_MainBody*, *HTML_More-InfoBody*, *HTML_CallStackItem* und *HTML_VariableItem* aus, ersetzen die englischen Ausdrücke der Ausgabe durch die deutschen Ausdrücke und weisen diese im Code den entsprechenden Eigenschaften wieder zu.

Die englische Originalvariante sieht, von einer VBA-Prozedur aus ausgelöst, wie in Abbildung 14.8 aus.

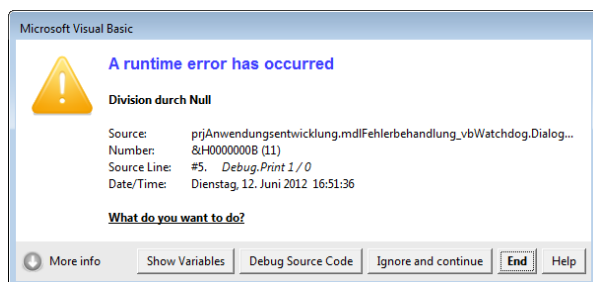


Abbildung 14.8: Englische Originalfehlermeldung

Die folgende Anweisung stellt die Texte der Meldung um. Beachten Sie, dass Sie die im HTML-Code enthaltenen Anführungszeichen beim Zuweisen an die Eigenschaft verdoppeln – andernfalls würde VBA ein einzelnes Anführungszeichen innerhalb des Ausdrucks als Ende der Zeichenkette interpretieren und der Rest der Zeile würde einen Fehler auslösen.

Die folgende Anweisung teilt die einzelnen Informationen auch noch auf einzelne Zeilen auf, um die Lesbarkeit zu erhöhen:

```
.HTML_MainBody = "<font face=Arial size=13pt color=""#4040FF"">  
<b>Es ist ein Laufzeitfehler aufgetreten.</b></font><br><br>  
<b><ERRDESC></b><br><br>  
Projekt: |<SOURCEPROJ><br>  
Modul: |<SOURCEMOD><br>  
Prozedur: |<SOURCEPROC><br>  
Fehlernummer: |<ERRNUMBER><br>  
Zeile: |<SOURCELINENUMBER><br>  
Zeileninhalt: |<SOURCELINECODE><br>  
Datum und Zeit: |<ERRDATETIME><br><br>  
<b><u>Was möchten Sie tun?</u></b>"
```

Dies übersetzt allerdings noch nicht die Texte der Schaltflächen. Deren Beschriftung lässt sich nicht direkt anpassen. Stattdessen entfernen Sie alle Schaltflächen mit der *RemoveAllButtons*-Methode und fügen die Schaltflächen mit benutzerdefinierten Beschriftungen und eingebauten Funktionen wieder hinzu. Das sieht so aus:

Fehlerbehandlung mit vbWatchdog

```
With ErrEx.DialogOptions
    .RemoveAllButtons
    .AddButton "Variablen anzeigen", BUTTONACTION_SHOWVARIABLES
    .AddButton "Quellcode debuggen", BUTTONACTION_ONERRORDEBUG
    .AddButton "Ignorieren und fortsetzen", BUTTONACTION_ONERRORRESUMENEXT
    .AddButton "Beenden", BUTTONACTION_ONERROREND
    .AddButton "Hilfe", BUTTONACTION_SHOWHELP
    .MoreInfoCaption = "Details einblenden"
    .LessInfoCaption = "Details ausblenden"
    .WindowCaption = "Fehler im Projekt '<SOURCEPROJ>'"
End With
```

Wie Sie sehen, können Sie die Platzhalter wie `<SOURCEPROJ>` auch in den übrigen Eigenschaften verwenden, die nicht im HTML-Format angegeben werden. Das Ergebnis sieht schließlich wie in Abbildung 14.9 aus und entspricht etwa der Fehlermeldung, die Sie während der Entwicklung anzeigen können.

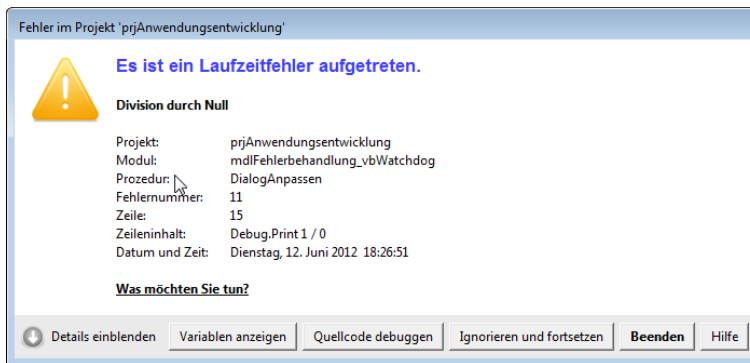


Abbildung 14.9: Angepasste Fehlermeldung

Wenn Sie die Anwendung weitergeben, möchten Sie dem Benutzer sicher keine Fehlermeldung wie die in dieser Abbildung liefern. Die Fehlerdetails sind wahrscheinlich eher uninteressant. Wichtiger ist, dass Sie eine Möglichkeit zum Versenden der Fehlerinformationen durch den Benutzer an den Entwickler der Anwendung integrieren.

Vorher schauen wir uns allerdings noch an, wie Sie die in der Fehlermeldung angezeigten Informationen nutzen können.

Fehlerhafte Zeile

Wichtig ist hier vor allem die Zeile, in welcher der Fehler auftritt. Während der Ort über Projekt, Modul und Prozedur bereits relativ eng eingegrenzt wurde und Sie auch wissen, welchen Text die Zeile enthält, die den Fehler ausgelöst hat, möchten Sie vielleicht über die Angabe der

Kapitel 14 Fehlerbehandlung

Zeilennummer auch in größeren Prozeduren schnell die betroffene Zeile auffinden – und zwar ohne Einsatz der Suchfunktion.

Die Fehlermeldung liefert zwar die Zeilennummer, aber wo im Quellcode finden Sie diese Zeile? Sie müssen dafür nicht etwa von oben durchzählen, sondern einfach nur eine Option von *vbWatchDog* aktivieren.

Diese findet sich in einer zusätzlichen Symbolleiste im VBA-Editor und heißt *Toggle LineNumbers*. Nach einem Mausklick auf diese Schaltfläche wird neben dem Codefenster eine Spalte mit den Zeilennummern eingeblendet (siehe Abbildung 14.10).

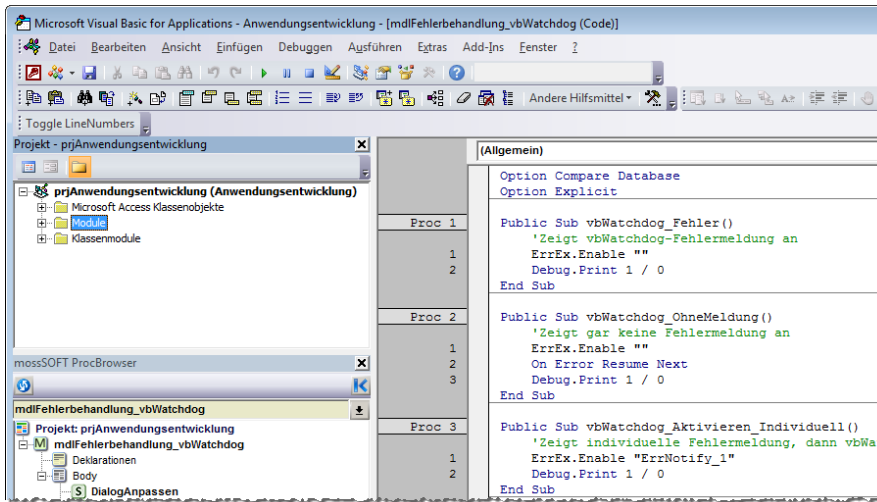


Abbildung 14.10: Einblenden der Zeilennummern im VBA-Code

Davon abgesehen gelangen Sie aber auch leicht durch einen Klick auf die Schaltfläche mit der eingebauten Funktion `<BUTTONACTION_ONERRORDEBUG>` in der fehlerhaften Zeile.

Aufrufeliste

Die Aufrufeliste (Stack) ist eine Liste der Prozeduren, in deren Folge ein Fehler ausgelöst wurde. Während Sie mit herkömmlichen benutzerdefinierten Fehlerbehandlungen ohne größeren Aufwand nur die Prozedur ermitteln können, die einen Fehler ausgelöst hat, liefert *vbWatchdog* nach Wunsch auch alle zuvor aufgerufenen Prozeduren.

Angenommen, Sie verwenden ein Formular mit einer Schaltfläche namens *cmdFehler*, die folgende Prozedur auslöst:

```
Private Sub cmdFehler_Click()  
    Fehlerbeispiel_1  
End Sub
```

Diese Prozedur ruft zwei weitere Prozeduren auf, von denen die letzte einen Fehler auslöst:

```
Public Sub Fehlerbeispiel_1()  
    Call Fehlerbeispiel_2  
End Sub  
  
Public Sub Fehlerbeispiel_2()  
    Dim intZahl As Integer  
    intZahl = 1 / 0  
End Sub
```

Dann liefert die erweiterte Ansicht die Liste aus Abbildung 14.11. Dort können Sie im unteren Bereich genau erkennen, wo der Fehler geschieht und über welche Prozeduren die fehlerhafte Prozedur aufgerufen wurde.

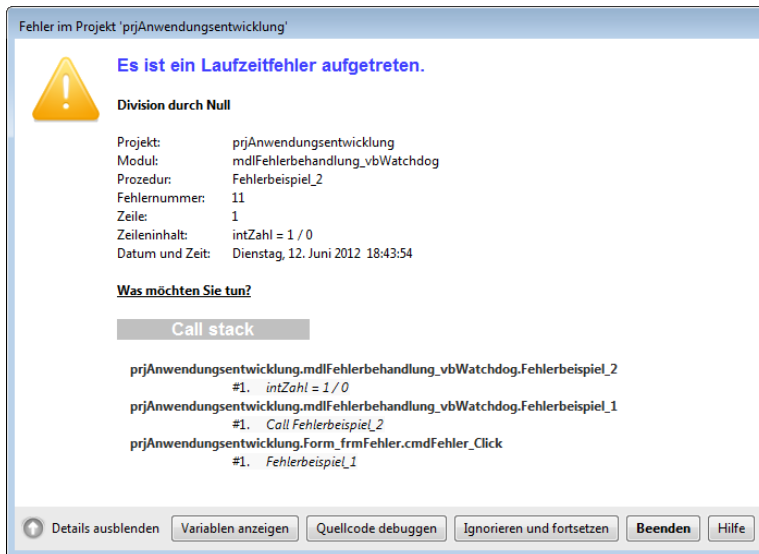


Abbildung 14.11: Herleitung des Fehlers über mehrere Prozeduren

Das Aussehen der Aufrufeliste beeinflussen Sie über die Variable `HTML_CallStackItem` des Objekts `ErrEx.DialogOptions`. Wenn Sie die Ausgabe variieren möchten, stellen Sie diese Variable auf einen neuen Ausdruck ein.

Die folgende Anweisung enthält den standardmäßig verwendeten Ausdruck mit deutschen Texten:

```
ErrEx.DialogOptions.HTML_CallStackItem = "<b><font color=#303030>  
<SOURCEPROJ>.<SOURCEMOD>.<SOURCEPROC></font></b><br>  
|<font bgcolor=#F8F8F8>Zeile <SOURCELINENUMBER>:  
<i><SOURCELINECODE></i></font><br>"
```

Kapitel 14 Fehlerbehandlung

Die Überschrift (in der Abbildung *Call stack*) ändern Sie über die Eigenschaft *ErrEx.DialogOptions.HTML_MoreInfoBody*:

```
ErrEx.DialogOptions.HTML_MoreInfoBody = "<br><b><font face=Arial size=13pt color=#FFFFFF  
bgcolor=#C0C0C0> Aufrufeliste </font></b><br><br><CALLSTACK>"
```

Variablen

Das ist ja fast zu schön, um wahr zu sein – fehlt nur noch, dass man sich auch noch die Werte der Variablen zum Fehlerzeitpunkt ansehen kann. Und auch dies bietet *vbWatchdog*! Wenn Sie nämlich auf die Schaltfläche mit der Funktion *<BUTTONACTION_SHOWVARIABLES>* klicken, erscheint der Dialog aus Abbildung 14.12.

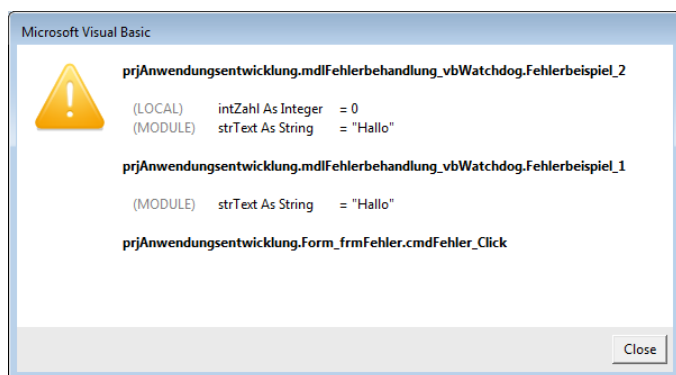


Abbildung 14.12: Ein weiterer Dialog zeigt die Variablen der verschiedenen Prozeduren und deren Inhalte zum Fehlerzeitpunkt.

Die hier dargestellten Daten und die Steuerelemente können Sie ebenfalls beeinflussen. Dazu stellt *vbWatchdog* ein eigenes Objekt namens *VariablesDialogOptions* bereit. Die folgenden Zeilen passen die Texte der Titelleiste und der ausgegebenen Variableninhalte an, entfernen alle Schaltflächen und fügen eine Schaltfläche zum Schließen des Dialogs hinzu.

```
With ErrEx.VariablesDialogOptions  
.WindowCaption = "Variablenliste"  
.HTML_CallStackItem = _  
    "<b><SOURCEPROJ>. <SOURCEMOD>. <SOURCEPROC></b><br><br><VARIABLEN><br>"  
.HTML_MainBody = "<CALLSTACK>"  
.HTML_MoreInfoBody = ""  
.HTML_VariableItem = _  
    "<font color=#808080><VARSCOPE></font>|<VARNAME> As <VARTYPE>| = <VARVALUE><br>"  
.RemoveAllButtons  
.AddButton "Schließen", BUTTONACTION_VARIABLES_CLOSE  
End With
```

Die vollständige Version der Fehlerbehandlung für den Entwicklermodus finden Sie im Modul *mdlFehlerbehandlung_vbWatchdog* unter *Errordialog_Entwickler*.

14.2.3 Fehlerdialog für den Endbenutzer

Der Endbenutzer soll einen ganz anderen Dialog vorfinden, wenn ein Fehler auftritt. Dieser darf natürlich nicht in den Debugging-Modus der Anwendung wechseln und er soll auch nicht die Aufrufeliste oder die Variablen anzeigen. Stattdessen soll der Benutzer aber zwei neue Schaltflächen vorfinden:

- » eine, mit der er die Fehlerbeschreibung direkt in einer neuen Outlook-Mail versenden kann, und
- » eine, welche die Fehlerbeschreibung in die Zwischenablage kopiert, damit er diese mit einem alternativen E-Mail-Programm an den Entwickler senden kann.

Beides soll mit einem Fehlerdialog erfolgen, der wie in Abbildung 14.13 aussieht. Dort finden Sie zwei Schaltflächen zum Aufrufen der oben genannten Aktionen sowie zum Beenden des Dialogs.

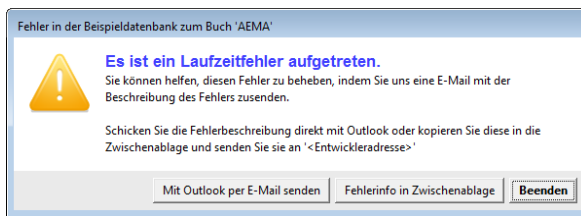


Abbildung 14.13: Fehlermeldung für den Endbenutzer

Damit der Dialog so aussieht wie in der Abbildung, rufen Sie beim Start der Anwendung die folgende Prozedur auf:

```
Public Sub Errordialog_Benutzer()  
    Dim strMainBody As String  
    ErrEx.Enable "DatenSammeln"  
    With ErrEx.DialogOptions  
        strMainBody = "<font face=Arial size=13pt color=""#4040FF"">" _  
            & "<b>Es ist ein Laufzeitfehler aufgetreten.</b></font><br>"  
        strMainBody = strMainBody & "Sie können helfen, diesen Fehler zu beheben, indem " _  
            & "Sie uns eine E-Mail mit der Beschreibung des Fehlers zusenden.<br><br>"  
        strMainBody = strMainBody & "Schicken Sie die Fehlerbeschreibung direkt " _  
            & "mit Outlook oder kopieren Sie diese in die Zwischenablage und senden " _  
            & "Sie sie an '<Entwickleradresse>'"  
        .HTML_MainBody = strMainBody  
        .RemoveAllButtons
```

Kapitel 14 Fehlerbehandlung

```
.ShowMoreInfoButton = False  
.AddCustomButton "Mit Outlook per E-Mail senden", "ErrorMail"  
.AddCustomButton "Fehlerinfo in Zwischenablage", "ErrorClipboard"  
.AddButton "Beenden", BUTTONACTION_ONERROREND  
.DefaultButtonID = 2  
.WindowCaption = "Fehler in der Beispieldatenbank zum Buch 'AEMA'"  
End With  
End Sub
```

Die Prozedur aktiviert zunächst die Fehlerbehandlung und legt fest, dass beim Auftreten eines Fehlers die Prozedur *DatenSammeln* aufgerufen wird. Diese Prozedur ist der sogenannte *Global Error Handler*. Danach stellen Sie das Erscheinungsbild des Dialogs ein.

Im Wesentlichen definieren Sie dabei den Meldungstext und weisen diesen der Eigenschaft *HTML_MainBody* zu. Danach entfernt die Methode *RemoveAllButtons* die vorhandenen Schaltflächen. Durch das Einstellen der Eigenschaft *ShowMoreInfoButton* auf den Wert *False* wird die Schaltfläche zum Erweitern des Dialogs ausgeblendet.

Nun folgt zwei Mal der Aufruf der Methode *AddCustomButton*, die komplett benutzerdefinierte Schaltflächen hinzufügt. Dies bedeutet, dass Sie als ersten Parameter die Beschriftung angeben und als zweiten den Namen der benutzerdefinierten VBA-Prozedur, die beim Anklicken ausgelöst werden soll.

Die beiden Prozeduren heißen *ErrorMail* und *ErrorClipboard*. Die Schaltfläche *Beenden* wiederum beendet den Dialog. Die Eigenschaft *DefaultButtonID* legt fest, dass die dritte Schaltfläche, also die zum Beenden des Dialogs, als Standardschaltfläche festgelegt wird (der Index für die Schaltflächen ist 0-basiert). Schließlich wird noch eine Titelbeschriftung zugewiesen.

14.2.4 Global Error Handler-Prozedur definieren

Im Falle eines Fehlers ruft vbWatchdog nun zunächst die als *Global Error Handler* festgelegte Prozedur auf, die in diesem Fall *DatenSammeln* lautet. Die folgende Prozedur trägt einige Informationen zum aufgetretenen Fehler zusammen und schreibt diese als Zeichenkette in ein Feld der Tabelle *tblOptionen*.

Dieses müssen Sie zunächst noch anlegen, es heißt *LetzterFehler* und hat den Felddatentyp *Memo*. Warum schreiben wir die Fehlerbeschreibung in eine Tabelle, wenn wir diese doch in die Zwischenablage kopieren oder per E-Mail versenden möchten?

Weil wir nur in der als *Global Error Handler* angegebenen Routine auf die Variablen zugreifen können, die uns die notwendigen Fehlerinformationen liefern.

Zu diesem Zeitpunkt steht jedoch noch nicht fest, ob der Benutzer die Fehlerinformationen per E-Mail versenden, in die Zwischenablage kopieren oder gar nicht verwerten möchte. Daher müssen wir die Fehlerinformationen zuvor noch an geeigneter Stelle zwischenspeichern.

Fehlerbehandlung mit vbWatchdog

Die Prozedur *DatenSammeln* sammelt die Fehlerinformationen zunächst in einer Variablen namens *strError*. Die ersten Informationen stammen aus dem Objekt *ErrEx* beziehungsweise liefern das Datum und die Uhrzeit des Fehlers. *ErrEx* liefert mit den beiden Eigenschaften *Number* und *Description* die Fehlernummer und die Fehlerbeschreibung.

Die Daten werden jeweils mit führendem Beschreibungstext (*Datum, Fehlernummer, Beschreibung*) in eine eigene Zeile geschrieben. Für den Zeilenumbruch sorgt die VBA-Konstante *vbCrLf*.

Danach folgt eine erste *Do...Loop While*-Schleife, die erst beendet wird, wenn die Eigenschaft *NextLevel* den Wert *False* liefert. Bis dahin liefern die Eigenschaften *ProjectName, ModuleName, ProcedureName, LineNumber* und *LineCode* die Informationen für das jeweils aktuelle Element der Aufrufeliste.

NextLevel liefert aber nicht nur die Information, ob es noch eine weitere Ebene gibt, sondern fügt auch gleich die Werte der nächsten Ebene für die oben genannten Eigenschaften zum Objekt *ErrEx.Callstack* hinzu.

Für jede Ebene der Aufrufeliste gibt es wiederum einen Satz von Variablen. Diese liefert das Objekt *ErrEx.Callstack.VariablesInspector*. Mit der Anweisung *FirstVar* wird die erste Variable eingelesen, mit der Methode *NextVar* die jeweils folgende. Dies geschieht so lange, bis die Eigenschaft *IsEnd* den Wert *True* zurückliefert.

Innerhalb der Schleife liest die Prozedur die Eigenschaften *Name, Value, TypeDesc* und *Scope* aus und fügt sie, um ein paar Leerzeichen eingerückt, zur *String*-Variablen *strError* mit den Fehlerinformationen hinzu. Bei der Eigenschaft *Value* kann es zu Problemen kommen, wenn es sich dabei um ein Objekt handelt und dieses auch noch den Wert *Nothing* enthält.

Für diesen Fall haben wir eine Prüfung des Datentyps mit der *Vartype*-Funktion eingefügt. In einer *Select Case*-Bedingung werden die verschiedenen Werte (9 für Objekte, 13 für *Nothing* und im *Else*-Zweig alle übrigen) geprüft und gegebenenfalls nur der Datentyp (zum Beispiel *Database* oder *Recordset*, ermittelt mit der *TypeName*-Funktion) beziehungsweise der Wert *Nothing* an die Zeichenkette angehängt.

Zum Schluss setzt die Prozedur eine SQL-Anweisung zusammen, die den Inhalt der Variablen *strError* in das Feld *LetzterFehler* der Tabelle *tbOptionen* schreibt, und speichert diese Anweisung in der Variablen *strSQL*.

Der Inhalt dieser Variablen dient wiederum als Argument der *Execute*-Methode des in *db* referenzierten *Database*-Objekts. Die *Execute*-Methode führt die als Parameter angegebene SQL-Aktionsabfrage aus und trägt somit die Fehlerinformationen in die Zieltabelle ein. Die komplette Prozedur sieht wie folgt aus – Sie finden sie im Modul *mdlFehlerbehandlung_vbWatchdog* der Beispieldatenbank:

```
Public Sub DatenSammeln()  
    Dim strError As String  
    Dim strSQL As String
```

Kapitel 14 Fehlerbehandlung

```
Dim db As DAO.Database
Set db = CurrentDb
With ErrEx
    strError = strError & "Datum: " & Now & vbCrLf
    strError = strError & "Fehlernummer: " & .Number & vbCrLf
    strError = strError & "Beschreibung: " & .Description & vbCrLf
    With ErrEx.Callstack
        Do
            strError = strError & "  Projektname: " & .ProjectName & vbCrLf
            strError = strError & "  Modulname: " & .ModuleName & vbCrLf
            strError = strError & "  Prozedurname: " & .ProcedureName & vbCrLf
            strError = strError & "  Zeilennummer: " & .LineNumber & vbCrLf
            strError = strError & "  Zeileninhalt: " & .LineCode & vbCrLf
            With ErrEx.Callstack.VariablesInspector
                .FirstVar
                Do While Not .IsEnd
                    strError = strError & "    Variablenname: " & .Name & vbCrLf
                    Select Case VarType(.Value)
                        Case 9
                            strError = strError & "      Variablenwert: " _
                                & TypeName(.Value) & vbCrLf
                        Case 13
                            strError = strError & "      Variablenwert: Nothing" _
                                & vbCrLf
                        Case Else
                            strError = strError & "      Variablenwert: " _
                                & .Value & vbCrLf
                    End Select
                    strError = strError & "      Datentyp: " & .TypeDesc & vbCrLf
                    strError = strError & "      Gultigkeitsbereich: " _
                        & .Scope & vbCrLf
                    .NextVar
                Loop
            End With
        Loop While .NextLevel
    End With
End With
strSQL = "UPDATE tblOptionen SET LetzterFehler = '" _
    & Replace(strError, "'", "''") & "'"
db.Execute strSQL, dbFailOnError
End Sub
```

14.2.5 Globale Fehlerbehandlung bei On Error Resume Next

Gegebenenfalls möchten oder müssen Sie im Code der Anwendung mit der *On Error Resume Next*-Anweisung arbeiten, um Anwendungen auszuführen, die gegebenenfalls (aber nicht zwingend) einen Fehler auslösen.

Standardmäßig ignoriert *vbWatchdog* diese und ähnliche Anweisungen und löst dennoch die vorgesehenen Mechanismen aus.

In unserem Fall bedeutet dies, dass etwa der folgende Code zwangsläufig die Prozedur *DatenSammeln* startet und somit auch die Daten des Fehlers zusammenträgt:

```
On Error Resume Next
Debug.Print 1/0
If Err.Number = 0 Then
    ...
End If
```

Das ist natürlich nicht gewünscht – der Fehler sollte einfach übergangen werden, damit die folgende *If...Then*-Bedingung prüfen kann, ob die betroffene Anweisung einen Fehler ausgelöst hat. *vbWatchdog* bietet jedoch die Gelegenheit, zu prüfen, ob der Fehler erfolgte, nachdem die Fehlerbehandlung mit *On Error Resume Next* deaktiviert wurde.

Diese platzieren Sie optimalerweise ganz oben in der Prozedur *DatenSammeln*, damit die folgenden Anweisungen gar nicht erst ausgeführt werden:

```
Public Sub DatenSammeln()
    Dim strError As String
    Dim strSQL As String
    Dim db As DAO.Database
    If ErrEx.State = OnErrorResumeNext Then
        Exit Sub
    End If
    ...
End Sub
```

Auf die gleiche Weise können Sie auch andere Zustände geprüft werden – *ErrEx.State* kann auch Werte wie *OnErrorGoto0* oder *OnErrorGotoLabel* annehmen.

14.2.6 Benutzeraktionen bei Laufzeitfehlern

Nun fehlen noch die beiden Prozeduren, die der Benutzer durch einen Mausklick auf die beiden Schaltflächen *Mit Outlook per E-Mail senden* oder *Fehlerinfo in Zwischenablage* auslösen kann. Diese legen Sie ebenfalls als öffentliche Prozeduren im Standardmodul *mdlFehlerbehandlung_vbWatchdog* an.

Kapitel 14 Fehlerbehandlung

Fehlerinfo in Zwischenablage kopieren

Das Kopieren der Fehlermeldung ist die weniger aufwendige Variante – wenn man davon absieht, dass dazu einiger zusätzlicher Code notwendig ist, der im Standardmodul *mdlZwischenablage* liegt. Die Details dazu sind für uns uninteressant – wichtig ist nur, dass wir dort eine Prozedur namens *InZwischenablage* finden, die den Inhalt der als Parameter angegebenen Variablen in der Zwischenablage speichert. Die Prozedur *ErrorClipboard* sieht dementsprechend wie folgt aus:

```
Public Sub ErrorClipboard()  
    Dim strError As String  
    Dim db As DAO.Database  
    Set db = CurrentDb  
    strError = db.OpenRecordset("SELECT LetzterFehler FROM tblOptionen", _  
        dbOpenDynaset).Fields(0)  
    InZwischenablage strError  
    Set db = Nothing  
End Sub
```

Die Prozedur öffnet zunächst ein Recordset, das lediglich das Feld *LetzterFehler* der Tabelle *tblOptionen* zurückliefert. Mit der Eigenschaft *Fields(0)* greift die entsprechende Anweisung auf den Inhalt dieses Feldes für den ersten Datensatz dieses Recordsets zu. Das ist okay, da diese Tabelle ohnehin nur einen einzigen Datensatz enthält.

Der Inhalt dieses Feldes wird in der Variablen *strError* gespeichert und von dort aus mit der Routine *InZwischenablage* direkt in die Zwischenablage kopiert. Nun liefert der Inhalt von *strError* noch keinen Hinweis auf weitere interessante Informationen wie beispielsweise die Version der Anwendung, von Access und von Windows. Außerdem wäre es noch interessant, ob die Anwendung mit der Vollversion oder mit der Runtime-Version von Access geöffnet wurde. Damit auch diese Informationen in der Zwischenablage landen, ersetzen Sie die Zeile

```
strError = db.OpenRecordset("SELECT LetzterFehler FROM tblOptionen", _  
    dbOpenDynaset).Fields(0)
```

durch die folgenden:

```
strError = strError & "Version FE:      " & Nz(DLookup("Version_FE", "tblOptionen"), _  
    "") & vbCrLf  
strError = strError & "Version Access:  " & _  
    & Access.Version & " " & SysCmd(acSysCmdAccessVer) & vbCrLf  
strError = strError & "Runtime:        " & SysCmd(acSysCmdRuntime) & vbCrLf  
strError = strError & "Version Windows: " & GetWindowsVersion & vbCrLf  
strError = strError & "Datum:         " & _  
    & Format(Now, "yyyy-mm-dd, hh:nn:ss") & vbCrLf  
strError = strError & "Datenbank:     " & CodeDb.Name & vbCrLf
```

Fehlerbehandlung mit vbWatchdog

```
strError = strError & db.OpenRecordset("SELECT LetzterFehler FROM tblOptionen". dbOpen-  
Dynaset).Fields(0)
```

Die erste Zeile liest die Version der Datenbank aus dem Feld *Version_FE* der Tabelle *tblOptionen* aus. Die zweite ermittelt mit der *SysCmd*-Funktion mit dem Parameter *acSysCmdAccessVer* die Access-Version, die dritte zeigt auf, ob die Anwendung mit der Runtime- oder der Vollversion von Access gestartet wurde.

Die Funktion *GetWindowsVersion* aus der vierten Zeile wird im Standardmodul *mdlSysteminformation* definiert und liefert die Windows-Version. Schließlich folgen noch der Ordner und der Name der Datenbank, die den Fehler ausgelöst hat, und das aktuelle Datum. *strError* wird durch diese Anweisungen beispielsweise um die folgenden Informationen angereichert:

```
Version FE:          0.9.0.0  
Version Access:     14.0 14.0  
Runtime:            Falsch  
Version Windows:    Windows 7 (6.1.7601 - Service Pack 1)  
Datum:              2012-08-26, 13:02:39  
Datenbank:          C:\Daten\Buchprojekte\Access_Anwendungsentwicklung\Anwendung\AEMA.  
accdb  
Datum:              26.08.2012 13:02:37
```

Fehlerbeschreibung direkt per E-Mail versenden

Die zweite Variante ist ein Mausklick auf die Schaltfläche *Mit Outlook per E-Mail senden*. Dies löst die Funktion *ErrorMail* aus. Warum eine Funktion und keine Prozedur wie zuvor? Weil wir diesmal demonstrieren möchten, wie die Fehlermeldung nach dem Ausführen einer benutzerdefinierten Aktion automatisch geschlossen wird – beim Speichern des Fehlers in der Zwischenablage ist das ja nicht der Fall.

Die Prozedur ist prinzipiell so aufgebaut wie die zum Kopieren der Fehlerinformationen in die Zwischenablage. Der Unterschied ist, dass sie die Daten nicht in die Zwischenablage kopiert, sondern diese per Mail versendet.

Dazu verwendet die Funktion die Klasse *clsMail*, die im Detail unter 13.2 beschrieben wird.

In diesem Fall instanziiert die Funktion eine neue Instanz dieser Klasse und füllt die Eigenschaften *Betreff*, *Inhalt* und *ToHinzufuegen* mit den entsprechenden Informationen. In der Zeile mit der Empfängeradresse tragen Sie Ihre eigene E-Mail-Adresse ein.

Damit der Benutzer sieht, welche Informationen an den Entwickler der Anwendung gesendet werden, soll die E-Mail nicht direkt versendet, sondern angezeigt werden. Dies erledigt die Funktion mit der *Anzeigen*-Methode des Klasse *clsMail*:

```
Public Function ErrorMail() As OnErrorStatus  
    Dim db As DAO.Database
```

Kapitel 14 Fehlerbehandlung

```
Dim strError As String
Dim objMail As clsMail
Set objMail = New clsMail
Set db = CurrentDb
strError = strError & "Version FE:          " _
    & Nz(DLookup("Version_FE", "tblOptionen"), "") & vbCrLf
strError = strError & "Version Access:      " _
    & Access.Version & " " & SysCmd(715) & vbCrLf
strError = strError & "Runtime:                " & SysCmd(acSysCmdRuntime) & vbCrLf
strError = strError & "Version Windows:       " & GetWindowsVersion & vbCrLf
strError = strError & "Datum:                  " _
    & Format(Now, "yyyy-mm-dd. hh:nn:ss") & vbCrLf
strError = strError & "Datenbank:             " & CodeDb.Name & vbCrLf
strError = strError & db.OpenRecordset("SELECT LetzterFehler FROM tblOptionen", _
    dbOpenDynaset).Fields(0)
With objMail
    .Betreff = "Fehlermeldung 'AEMA'"
    .Inhalt = strError
    .ToHinzufuegen "andre@minhorst.com"
    .Anzeigen
End With
Set db = Nothing
ErrorMail = OnErrorEnd
End Function
```

Die Funktion gibt einen Wert mit dem Datentyp *OnErrorStatus* zurück, in diesem Fall *OnErrorEnd*. Das bedeutet, dass die aufrufende Fehlermeldung nach dem Ausführen der benutzerdefinierten Fehlerbehandlung geschlossen wird.

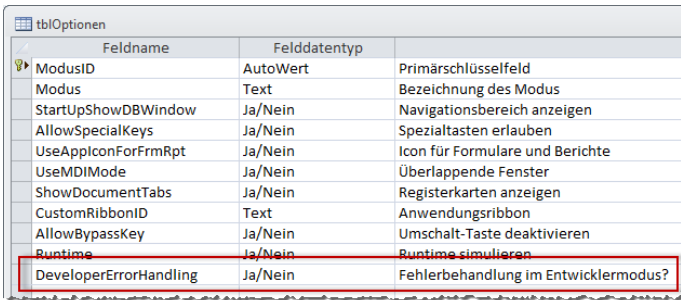
14.3 Fehlerbehandlungsmodus einstellen

Je nachdem, ob Sie gerade an der Datenbankanwendung arbeiten oder ob der Benutzer diese bereits nutzt, soll die jeweils benötigte Fehlerbehandlung aktiviert werden. Dies erledigt jeweils ein Aufruf der Prozeduren *ErrorDialog_Entwickler* oder *ErrorDialog_Benutzer*.

Wie aber erfährt die Anwendung, welcher Modus zum Einsatz kommen soll? Im Abschnitt »Anwendung für Entwicklung und Test starten« (Seite 42) haben Sie bereits die Starter-Datenbank kennengelernt, mit der Sie die Anwendung mit verschiedenen Einstellungen öffnen können. Diese Datenbank erweitern wir nun um eine weitere Option.

Dazu starten Sie die Datenbank *AEMA_Developer.accdb* und öffnen die Tabelle *tblOptionen* im Entwurfsmodus. Fügen Sie ein Feld wie in Abbildung 14.14 hinzu.

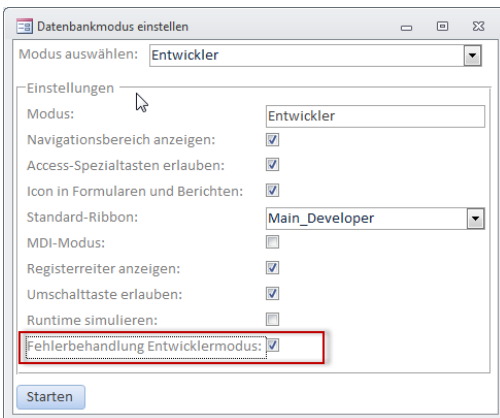
Fehlerbehandlungsmodus einstellen



| Feldname | Felddatentyp | Primärschlüsselfeld |
|------------------------|--------------|--------------------------------------|
| ModusID | AutoWert | Primärschlüsselfeld |
| Modus | Text | Bezeichnung des Modus |
| StartUpShowDBWindow | Ja/Nein | Navigationsbereich anzeigen |
| AllowSpecialKeys | Ja/Nein | Spezialtasten erlauben |
| UseApplIconForFrmRpt | Ja/Nein | Icon für Formulare und Berichte |
| UseMDIMode | Ja/Nein | Überlappende Fenster |
| ShowDocumentTabs | Ja/Nein | Registerkarten anzeigen |
| CustomRibbonID | Text | Anwendungsribbon |
| AllowBypassKey | Ja/Nein | Umschalt-Taste deaktivieren |
| Runtime | Ja/Nein | Runtime simulieren |
| DeveloperErrorHandling | Ja/Nein | Fehlerbehandlung im Entwicklermodus? |

Abbildung 14.14: Option zum Einstellen der Fehlerbehandlung

Auch im Formular zum Einstellen der einzelnen Optionen soll dieses Feld erscheinen (siehe Abbildung 14.15).



Modus auswählen: **Entwickler**

Einstellungen

Modus: **Entwickler**

Navigationsbereich anzeigen:

Access-Spezialtasten erlauben:

Icon in Formularen und Berichten:

Standard-Ribbon: **Main_Developer**

MDI-Modus:

Registerreiter anzeigen:

Umschalttaste erlauben:

Runtime simulieren:

Fehlerbehandlung Entwicklermodus:

Starten

Abbildung 14.15: Aktivieren der Fehlerbehandlung im Entwicklermodus

Fügen Sie zur Prozedur `cmdStart_Click` die folgenden Zeilen hinzu:

```
...  
If Me!DeveloperErrorHandling Then  
    strRuntime = " /cmd ""Developer""  
End If  
Shell Chr(34) & strAccess & Chr(34) & " " & Chr(34) & strPfad & strDatenbank _  
    & Chr(34) & strRuntime  
...
```

Dadurch erweitert die Prozedur den Aufruf der Access-Datenbank gegebenenfalls wie folgt:

```
"C:\Program Files (x86)\Microsoft Office\Office14\MSAccess.exe" "C:\Daten\Buchprojekte\  
Access_Anwendungsentwicklung\Anwendung\AEMA.accdb" /cmd "Developer"
```

Kapitel 14 Fehlerbehandlung

Der Aufruf wird also um den Ausdruck */cmd "Developer"* erweitert. Wie aber sollen wir diesen in der Anwendung so auswerten, dass die für den Entwicklungsmodus beziehungsweise den Produktivmodus vorgesehene Fehlerbehandlung aktiviert wird?

Ganz einfach: Wenn Sie den Parameter */cmd* beim Aufruf einer Access-Anwendung setzen, können Sie von dieser Anwendung aus den dabei übergebenen Wert mit der *Command()*-Funktion auslesen. Dies erledigen wir in der Prozedur, die durch das Ereignis *Beim Laden* des Formulars *frmStart* ausgelöst wird:

```
Private Sub Form_Load()  
    If Command() = "Developer" Then  
        Errordialog_Entwickler  
    Else  
        Errordialog_Benutzer  
    End If  
End Sub
```

Wenn Sie in der Start-Anwendung *AEMA_Developer.accdb* also die Option *Fehlerbehandlung Entwickler* aktiviert haben, übergibt diese mit */cmd* den Wert *Developer*. Die Prozedur *Form_Load* des Formulars *frm_Start* ruft in diesem Fall die Prozedur *Errordialog_Entwickler* auf, welche die Fehlerbehandlung für den Entwicklermodus startet. Wenn *Command()* nicht den Wert *Developer* liefert, startet die Prozedur hingegen die Fehlerbehandlung für den Endbenutzer.