

19. Access-Entwickler-Konferenz
Nürnberg 24./25.09. / Düsseldorf 08./09.10. / Hannover 15./16.10.2016

FEHLER FÜR FORTGESCHRITTENE

Mehr als nur Debug.Print



Thomas Möller, www.Team-Moeller.de

Vorstellung

- Thomas Möller
 - dipl. Sparkassenbetriebswirt
- 2000 bis 2013 hauptberuflich als Access Entwickler tätig
- Schwerpunkte
 - Access mit DB/2 als BackEnd
 - Web-Entwicklung mit ASP.Net
- Seit 2013 Data Governance/Qualitätsmanagement
 - Aktuell ApEx mit Oracle
- Nebenberuflich: Team-Moeller.de
 - Add-Ins
- Seit 1.1.2007 MVP für Access



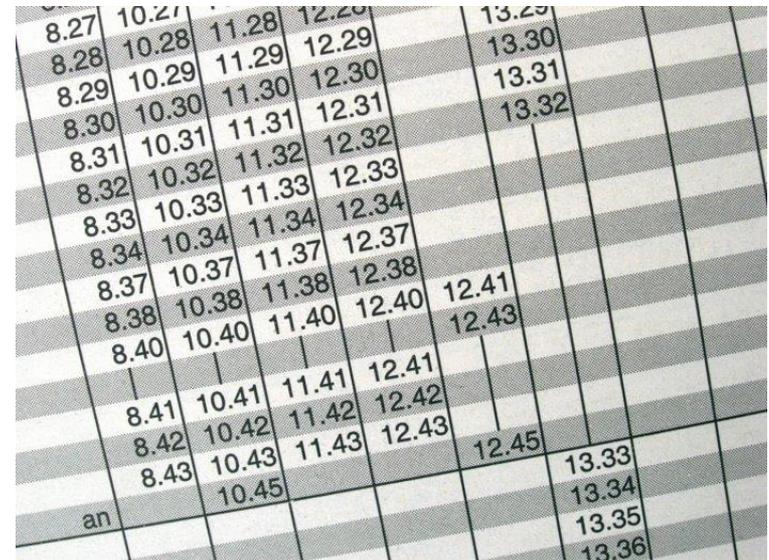
Kontaktdaten

- Webseite www.Team-Moeller.de
- Blog Blog.Team-Moeller.de
- E-Mail Thomas@Team-Moeller.de
- Twitter <https://twitter.com/ThomasMoeller>
- XING [https://www.xing.com/profile/Thomas Moeller40](https://www.xing.com/profile/Thomas_Moeller40)
- Linked In <https://www.linkedin.com/in/TeamMoeller>
- Facebook <https://www.facebook.com/TeamMoeller>
- Google Plus <https://plus.google.com/101082101965686277484>
- MVP <http://mvp.microsoft.com/de-de/mvp/Thomas%20Moeller-37548>



Fahrplan

- Grundlagen
 - Optionen, Fenster, Haltepunkte, Assert
- Error-Handling
 - Zentraler ErrorHandler, Zeilennummern, Error Propagation
- vbWatchdog
 - Funktionsweise, Implementierung



The image shows a tilted photograph of a train schedule table. The table has columns representing different time slots and rows representing different stations. The times listed are in HH:MM format. The stations are partially visible, with 'an' being clearly legible at the bottom left.

8.27	10.27	11.28	12.29	13.29	
8.28	10.28	11.29	12.29	13.30	
8.29	10.29	11.30	12.30	13.31	
8.30	10.30	11.31	12.31	13.32	
8.31	10.31	11.32	12.32		
8.32	10.32	11.33	12.33		
8.33	10.33	11.34	12.34		
8.34	10.34	11.37	12.37		
8.37	10.37	11.38	12.38	12.41	
8.38	10.38	11.40	12.40	12.43	
8.40	10.40				
8.41	10.41	11.41	12.41		
8.42	10.42	11.42	12.42		
8.43	10.43	11.43	12.43	12.45	13.33
an	10.45				13.34
					13.35
					13.36

Umfrage

- Wer hat in **jeder** Prozedur einen Error-Handler?
- Wer fügt Zeilennummern in den VBA-Code ein?
- Wer nutzt den vbWatchDog?



Optionen
Debug.Print
Resume
Überwachungsfenster
F8
Debug.Assert
Direktfenster
Lokal-Fenster
Shift+F8
Einzelschritt
Haltepunkte
Definition
Call-Stack
Prozedurschritt
Aufrufliste

GRUNDLAGEN

Grundlagen

- Kein Programm ist fehlerfrei
- Was passiert im Fehlerfall?
- Keine Fehlerbehandlung:
 - Programmausführung wird angehalten
 - Wechsel in den VBA-Code
 - Nicht möglich in *.accde / *.mde
- Mit Fehlerbehandlung:
 - Programmierer hat Kontrolle über weiteren Programmfluss
 - Fehler können protokolliert werden

Einfacher Error-Handler

- On Error GoTo Handle_Error
 - ' Super Programmcode
 - Exit_Here:
 - On Error Resume Next
 - Exit Sub
 - Handle_Error:
 - Select Case Err.Number
 - Case 0
 - Resume Next
 - Case 2501
 - Resume Next
 - Case Else
 - MsgBox Err.Description, vbExclamation, Err.Number
 - Resume Exit_Here
 - End Select
- Aufräumen:
Objekte schließen etc.
- Gezielte Reaktion auf
bestimmte Fehler
- Anzeige, E-Mail,
Protokollierung,...
-
- ```
graph TD; A[Exit Sub] --> B[Handle_Error]; B --> C[Case 0]; B --> D[Case 2501]; B --> E[Case Else]; C --> F[Aufräumen: Objekte schließen etc.]; D --> G[Gezielte Reaktion auf bestimmte Fehler]; E --> H[Anzeige, E-Mail, Protokollierung, ...];
```

# On Error und seine Freunde

- Wie das Error-Handling definieren?
- On Error Goto Label
  - Im Fehlerfall Sprung zum Label
- On Error Resume Next
  - Error-Handling ist deaktiviert, Code läuft weiter
- On Error Goto 0
  - Fehler löschen und zu Standardverhalten zurückkehren

# Resume und seine Freunde

- Was tun nach einem Fehler?
- Resume
  - Probier es noch einmal
- Resume Next
  - Nimm die nächste Zeile
- Resume LABEL
  - Sprung zum Label (analog Goto)

# Mehrere Error-Handler

- Mehrere Error-Handler in einer Prozedur möglich?

On Error Goto Handler\_1  
Code

On Error Goto Handler\_2  
Code

Exit\_Here:

Handler\_1:

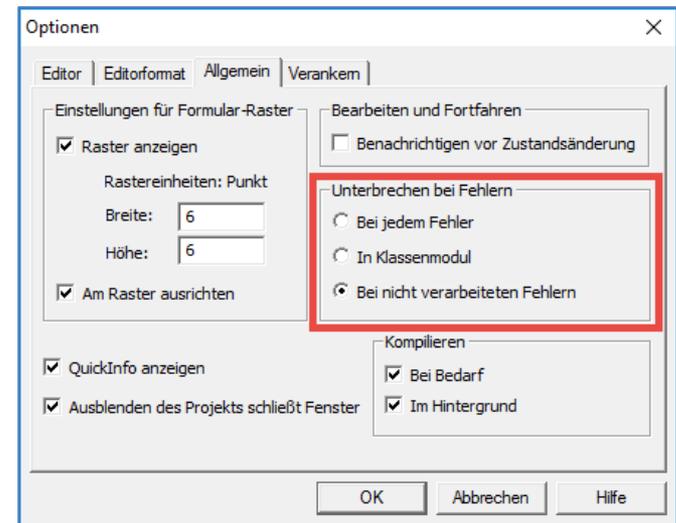
Handler\_2:

- Zwei Prozeduren anstreben



# Optionen Error-Handling

- Extras / Optionen / Allgemein / Unterbrechen bei Fehlern
- Einstellung gilt VBA-weit
- Kann vom User angepasst werden
- „Bei jedem Fehler“ unterbricht auch dann, wenn Fehlerbehandlung vorhanden ist



# Optionen abfragen und setzen

- Option abfragen:
  - `Application.GetOption("Error Trapping")`
- Option setzen:
  - `Application.SetOption "Error Trapping", myOption`

[DEMO](#)

- Option beim Start der Anwendung gezielt anpassen
  - schützt vor unerwarteten Situationen



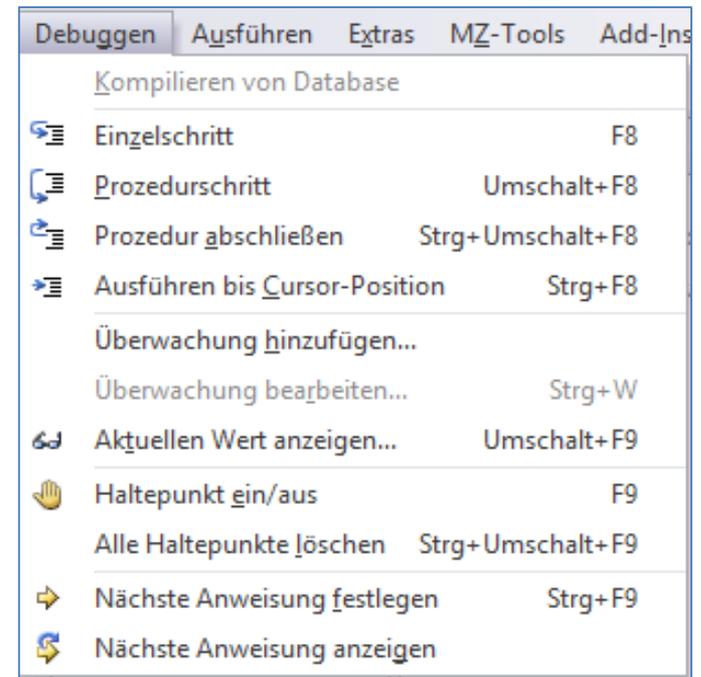
# Während der Entwicklung

- Fehlerbehandlung während der Entwicklung deaktivieren
- Anstatt:
  - `On Error Goto Handle_Error`
- Diese Variante:
  - `If DatenbankFertig Then On Error Goto Handle_Error`
- Zusätzliche Konstante erforderlich
  - `Public Const DatenbankFertig As Boolean = False`

Besserer Weg:  
Option "Bei jedem Fehler"

# Schrittweise durch den Code

- F8:
  - Einzelschritt
- Shift+F8:
  - Prozedurschritt
- Strg+Shift+F8:
  - Prozedur abschließen
- Strg+F8:
  - Ausführen bis Cursor



# Haltepunkte

- Haltepunkt (mit der Maus) definieren
- F9: setzt bzw. löscht Haltepunkt
- Strg+F9: Nächste Anweisung festlegen
  - Alternativ: Pfeil (mit der Maus) verschieben

```
Public Sub Demo_2()
On Error GoTo Handle_Error

 MsgBox 1 / 0
 MsgBox "Hier geht es weiter!"

Exit_Here:
 On Error Resume Next
 Exit Sub

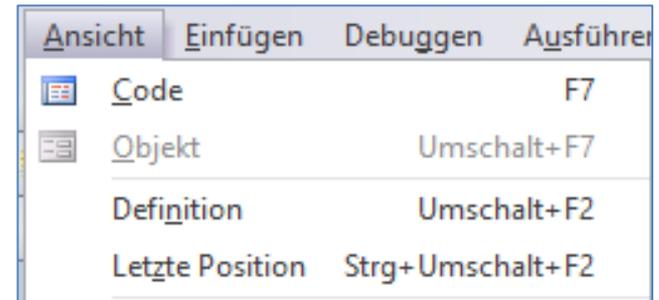
Handle_Error:
 Select Case Err.Number
 Case 0
 Resume Next
```

# Exkurs: The extra Resume

- Video von Armen Stein
  - <https://www.youtube.com/watch?v=0Xahwtk1Bk&feature=youtu.be>
- Resume nach Resume `Exit_Proc` eingefügen
- Wird nie ausgeführt
- Unterbrechungsmodus (Strg+Pause)
- Resume als nächsten Befehl festlegen
- F8 drücken
- Zeile, die den Fehler verursacht hat, ist markiert

# Definition

- Wo und wie ist ein Objekt (Variable, Funktion) definiert?
- Sprung zur Definition
  - Shift+F2: Gehe zu Definition
- Rücksprung zum Code
  - Strg+Shift+F2: Letzte Position
  - Geht über mehrere Ebenen



# Direktfenster

- Aufruf mit STRG+G
  - Shortcut funktioniert in VBA-IDE und in Access
- Aufruf von Befehlen möglich
  - Im Unterbrechungsmodus Ausgabe von Variablen oder Ausdrücken möglich
- Ausgabe von Ergebnissen
  - Ausgabe mit Debug.Print
  - Ideal zur Ausgabe von Zwischenergebnissen und Statusmeldungen

Direktbereich

```
? Currentdb.Name
D:\Users\Thomas\Documents\Präsentationen\Vortrag AEK 19\Demos\Demo_01_Grundlagen.accdb
|
```

# Tipps zu Debug.Print

- Gibt Zeile im Direktfenster aus
- Komma an Ende: Tabulator
- Semikolon am Ende: keine Zeilenschaltung

Debug.Print "Doing something ... " ;

‘ Lange laufender Code

Debug.Print "Done"

D  
E  
M  
O

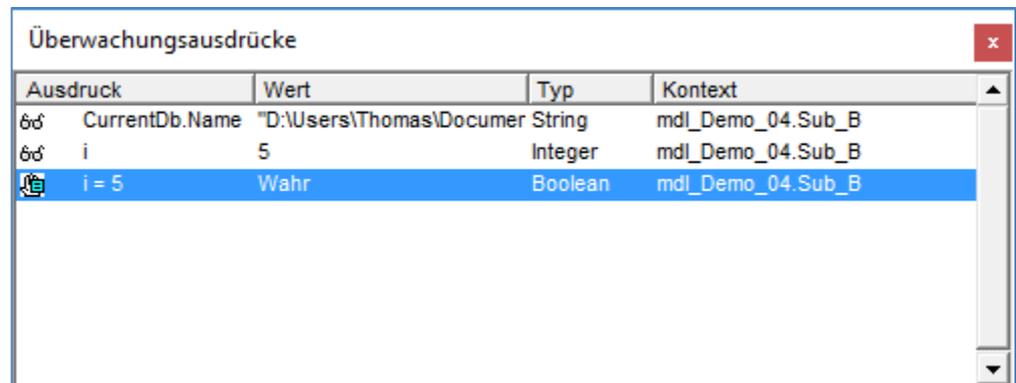
Direktbereich

```
Doing something ... Done!
Doing something else ... Done!
|
```

# Überwachungsfenster

- Dient zur Überwachung von Variablen und Ausdrücken
  - Überwachung
  - Unterbrechen, wenn Wert True ist
  - Unterbrechen, wenn Wert geändert wurde
- Hilfreich, um Code bis zum Fehler laufen zu lassen

**DEMO**



| Ausdruck       | Wert                     | Typ     | Kontext           |
|----------------|--------------------------|---------|-------------------|
| CurrentDb.Name | "D:\Users\Thomas\Documen | String  | mdl_Demo_04.Sub_B |
| i              | 5                        | Integer | mdl_Demo_04.Sub_B |
| i = 5          | Wahr                     | Boolean | mdl_Demo_04.Sub_B |

# Debug.Assert

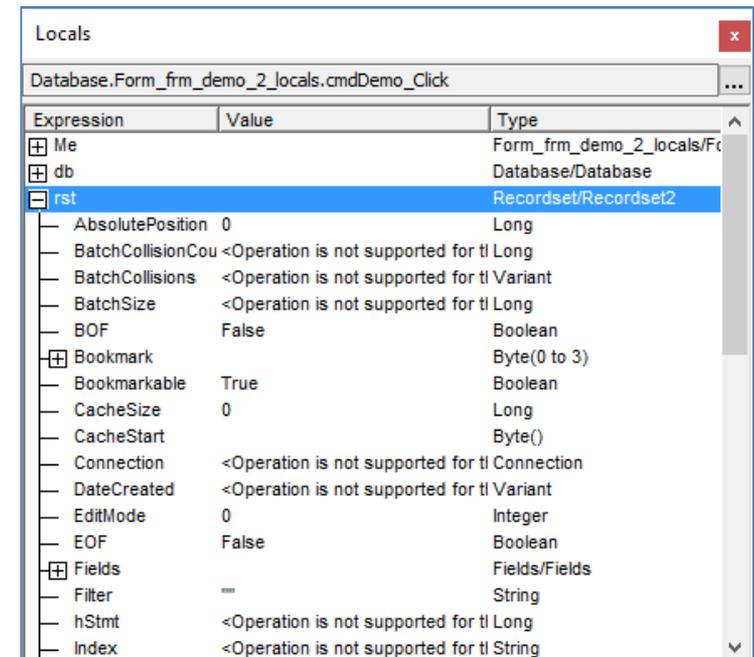
- Alternativen zu Überwachung:
  - If  $x = 5$  Then Stop
    - Oder
  - Debug.Assert  $x \neq 5$
- Debug.Assert hält Code an, wenn Bedingung **nicht** erfüllt ist.
- Unbedingt vor Produktivnahme der Anwendung aus Code entfernen!!



# Lokal-Fenster

- Anzeige von Objekten und Variablen
- ... incl. aller Eigenschaften und deren Werte
- Verschachtelung
- Hilfreich:
  - Eigenschaft finden
  - Daten überprüfen

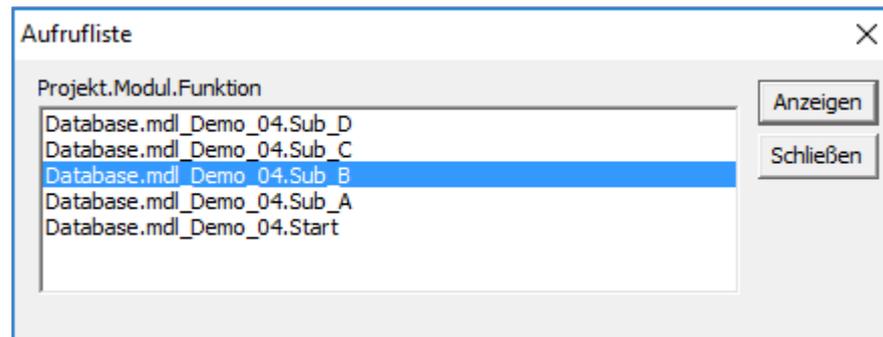
[DEMO](#)



| Expression        | Value                              | Type                         |
|-------------------|------------------------------------|------------------------------|
| Me                |                                    | Form_frm_demo_2_locals/Fc... |
| db                |                                    | Database/Database            |
| rst               |                                    | Recordset/Recordset2         |
| AbsolutePosition  | 0                                  | Long                         |
| BatchCollisionCou | <Operation is not supported for t! | Long                         |
| BatchCollisions   | <Operation is not supported for t! | Variant                      |
| BatchSize         | <Operation is not supported for t! | Long                         |
| BOF               | False                              | Boolean                      |
| Bookmark          |                                    | Byte(0 to 3)                 |
| Bookmarkable      | True                               | Boolean                      |
| CacheSize         | 0                                  | Long                         |
| CacheStart        |                                    | Byte()                       |
| Connection        | <Operation is not supported for t! | Connection                   |
| DateCreated       | <Operation is not supported for t! | Variant                      |
| EditMode          | 0                                  | Integer                      |
| EOF               | False                              | Boolean                      |
| Fields            |                                    | Fields/Fields                |
| Filter            | ==                                 | String                       |
| hStmt             | <Operation is not supported for t! | Long                         |
| Index             | <Operation is not supported for t! | String                       |

# Aufrufliste / Call-Stack

- Ansicht / Aufrufliste ... (Strg+L)
- Ermöglicht Blick auf Weg durch den Code
- Anzeigen: setzt Fokus auf entsprechenden Aufruf



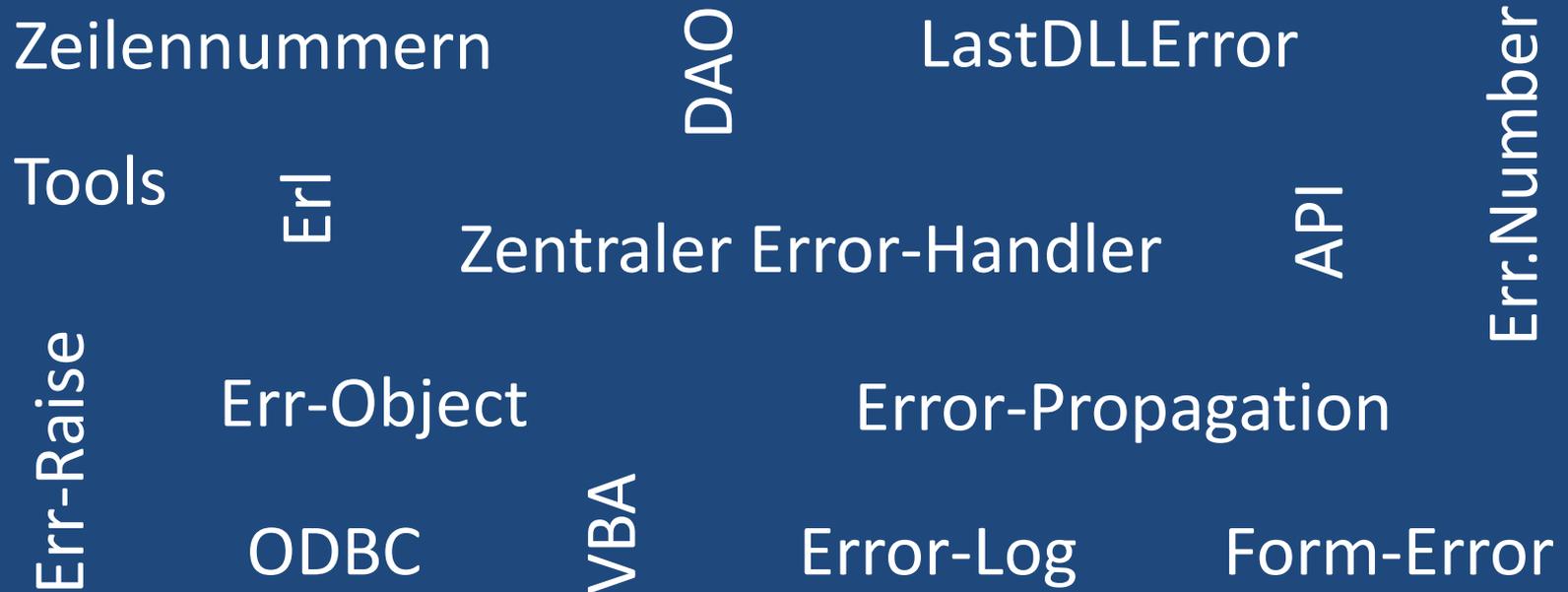
- Kann nicht per VBA abgerufen werden

# Eigener Call-Stack

- Kann im Fehlerfall abgerufen werden
- Implementierung über Array
- Bei Prozedurstart: Push
- Bei Prozedurende: Pop
- Vorsicht:  
Jeder Ausgang der Prozedur benötigt ein Pop

D  
E  
M  
O





---

# ERROR-HANDLING

# Das Err-Objekt

- Number                      Fehlernummer
- Source                        Name des VBA-Projekts
- Description                 Fehlerbeschreibung
- LastDllError                Letzte Fehler-Nummer aus API-Aufruf
  
- Clear                         Setzt Err-Objekt zurück
- Raise                         Wirft (eigenen) Fehler

# Err.Raise

- Fehler erneut werfen
- Eigene Fehler werfen

```
Select Case X
```

```
 Case 1
```

```
 Call Something
```

```
 Case 2
```

```
 Call SomethingElse
```

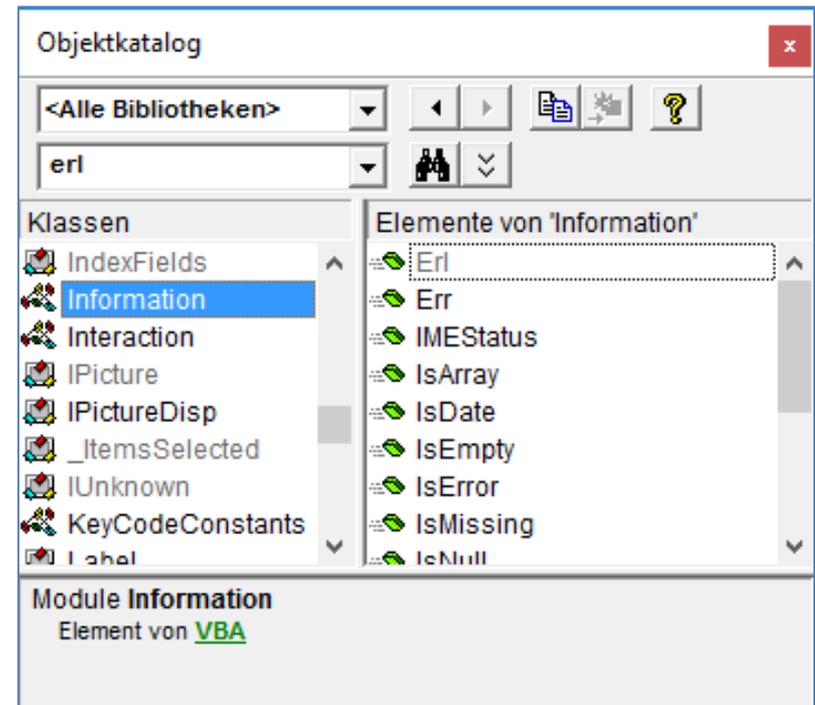
```
 Case Else
```

```
 Err.Raise vbObjectError, , "Not implemented yet"
```

```
End Select
```

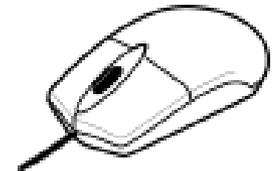
# Erl => ErrorLine

- Erl => Nummer der Zeile, in der der Fehler aufgetreten ist
- Verborgene Eigenschaft in VBA
- Relikt aus den Anfängen von (Visual) Basic



# Zeilennummern einfügen

- Von Hand einfügen
  - aufwändig
  - hinderlich beim Programmieren
- per Tool
  - Aufwand bei Entwicklung (aus, coden, ein)
  - oder Aufwand vor Auslieferung
- per VBA
  - vor Auslieferung der Anwendung



# TM-VBAZeilenNummern Steckbrief

- Name des Tools
  - TM-VBAZeilenNummerieren
- Autor
  - Thomas Möller
- Download
  - <http://www.team-moeller.de/?Add-Ins:TM-VBAZeilenNummerieren>
- Lizenz / Preis
  - Freeware
- Zweck / Funktion
  - Zeilennummern in VBA-Code einfügen bzw. entfernen



# Eigener ErrorHandler

- Zentrale Implementierung
- Ermöglicht Protokollierung aller Fehler
- Versand per E-Mail an den Entwickler
  
- Wichtig:  
Kein Error-Handling im zentralen Error-Handler
  
- Beispiele:
  - [FMS: Global Error Handler](#)
  - [Allen Browne: LogError](#)

# Tabelle Error-Log

- Welche Informationen sollen gespeichert werden?
  - Fehlernummer, Fehler-Text      Err.Number, Description
  - Benutzer      vom Betriebssystem
  - Datum, Uhrzeit      =Now()
  - Modulname, Prozedurname      Konstanten
  - Zeilennummer      ERL
  - Call-Stack      selbst implementiert
  - Name der Anwendung      Konstante
  - Version der Anwendung      Konstante

# Error-Propagation

- Bei verschachtelten Aufrufen wird Fehler so lange nach „oben“ weitergegeben, bis Error-Handler vorhanden
- Kein Error-Handling in Aufrufkette vorhanden
  - Fehler wird in aktueller Prozedur behandelt

[DEMO](#)

# Fortgeschrittener Ansatz

- Ausgangssituation:  
Verschachtelte Funktionsaufrufe
- Error-Propagation:  
ERL von aufrufender Funktion mit Err-Handling
- Alternative:  
Jede Funktion hat Error-Handler und gibt Erfolg als Boolean zurück => Schwer lesbarer Code
- Bessere Alternative:  
Error-Handler wirft Fehler bei Bedarf erneut
  - <http://www.dymeng.com/techblog/rethrowing-vba-errors/>

# VBA, DAO, ADO, ODBC

- In VBA wird immer nur ein Fehler gespeichert
- Bei Aufruf per ODBC können mehrere Fehler zurückgegeben werden
- Unterscheidung über Errors-Collection
- 1 Fehler: VBA, mehrere Fehler ODBC
  - For Each errX in DAO.Errors
  - For Each errADO in conn.Errors
  - errADO.SQLState

# Handling ODBC Errors

- Tipp von Tom van Stiphout für gebundene Formulare
  - <http://www.accessmvp.com/TomvanStiphout/OdbcErrors.htm>
- Fehler kann nur über Form\_Error abgefangen werden
  - Nur allgemeine Fehlermeldung (3146)
  - ODBC-Fehlerdialog erscheint trotzdem
- Lösung:
  - Timer setzen
  - in Timer-Event Fenster mit API-Aufruf ermitteln
  - Fehlertext mit API abfragen
  - durch eigenen Text ersetzen ...
  - und mit API vor Fenster einblenden

# API: LastDLLErr

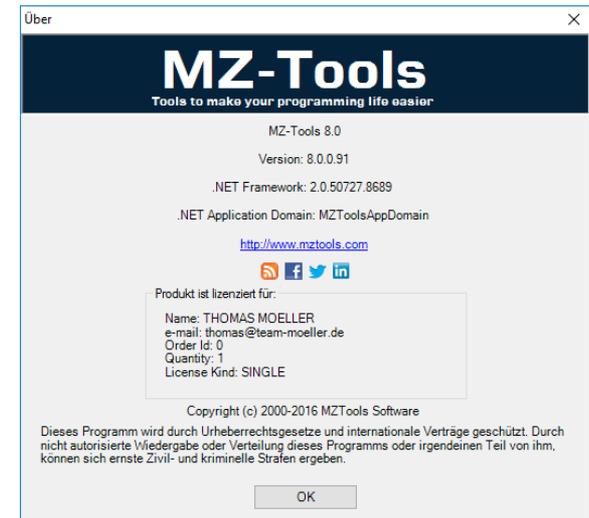
- (Viele) APIs liefern Return-Code
  - Positiver Wert = Ergebnis oder Erfolg
  - 0 als Rückgabe = Fehler
- Weitere Info zu Fehler muss erfragt werden
  - API: [GetLastError](#)
  - [Err.LastDLLError](#)
- Liste mit Error-Codes
  - [http://www.megos.ch/files/content/diverses/doserrors\\_e.txt](http://www.megos.ch/files/content/diverses/doserrors_e.txt)
- API-Funktion für Fehlertext
  - [FormatMessage](#)

# Form-Error

- Tritt bei fehlerhafter Eingabe ein
- Fehlermeldung kann unterdrückt werden
  - `acDataErrContinue`
- oder durch eigene, aussagekräftige Meldung ersetzt werden
  
- Exkurs: Sondereingabe zulassen
  - Beispiel von [P. Rohorzka, AEK 14](#)

# MZ-Tools - Steckbrief

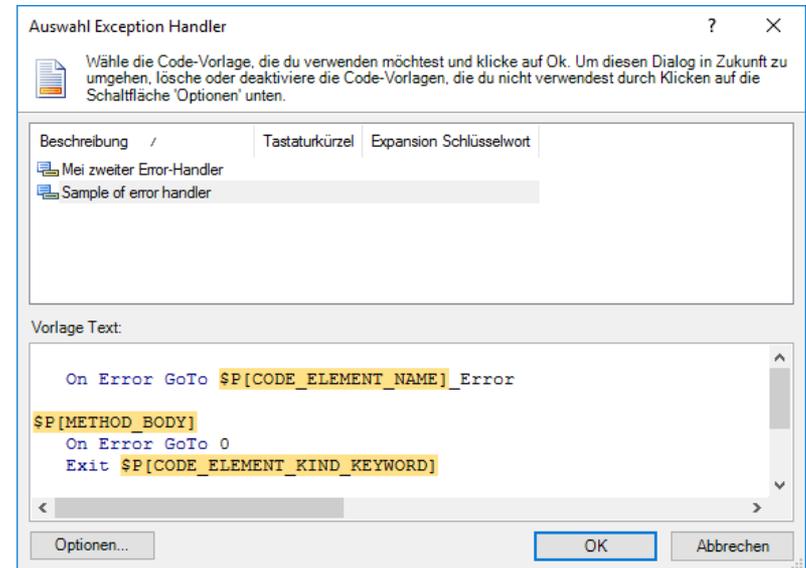
- Name des Tools
  - MZ-Tools
- Autor
  - Carlos Quintero
- Download
  - [http://www.mztools.com/v8/download\\_trial.aspx](http://www.mztools.com/v8/download_trial.aspx)
- Lizenz / Preis
  - Kommerziell: 74,05 €
- Zweck / Funktion
  - Umfangreiche Toolsammlung



30-Tage-Testversion

# MZ-Tools: Features für Error-Handling

- Error-Handler einfügen
- Mehrere Error-Handler
  - Auswahl bei Bedarf
- Vorlagen können deaktiviert werden
  - Nicht löschen, nur deaktivieren
- Zeilennummer hinzufügen
- Zeilennummern entfernen
- Zeilennummern per Skript einfügen



# Steckbrief

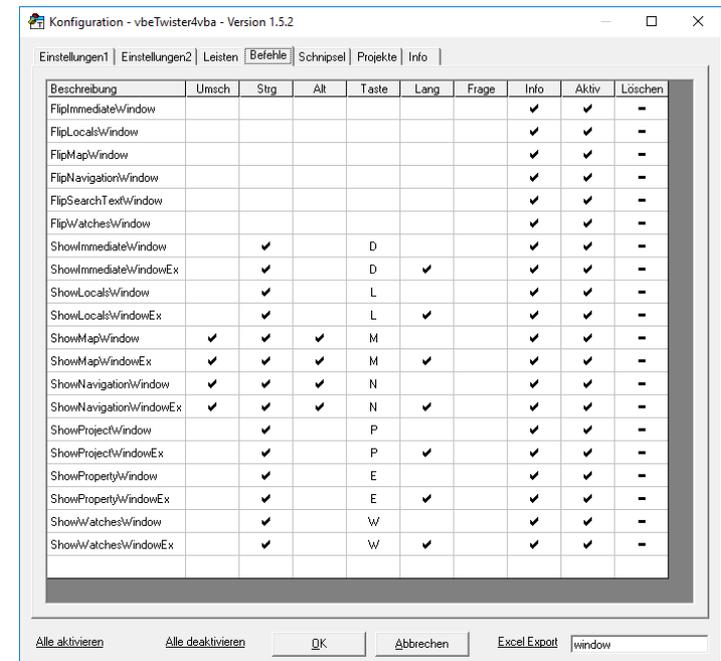
- Name des Tools
  - vbeTwister
- Autor
  - Martin Knotzer
- Download
  - <http://www.vbetwister.com>
- Lizenz / Preis
  - Kommerziell 99 €
- Zweck / Funktion
  - Zeitgemäße IDE und Editor für VBA



30-Tage-Testversion

# vbeTwister: Features für Debugging

- Error-Handler über „Schnipsel“ einfügen
  - oern => On Error Resume Next
  - oegl => On Error Goto Label
  - dbox => MsgBox Form-Name, Methodenname
  - Selbst definierte Schnipsel
- Diverse Shortcuts für
  - Überwachungsfenster
  - Lokalfenster
  - Aufrufliste
  - Direktfenster



ErrEx.Enable

Finally

ErrEx.Disable

Funktionsweise

Installation

OnErrorGotoLabel

Catch

OnErrorCatch

CatchAll

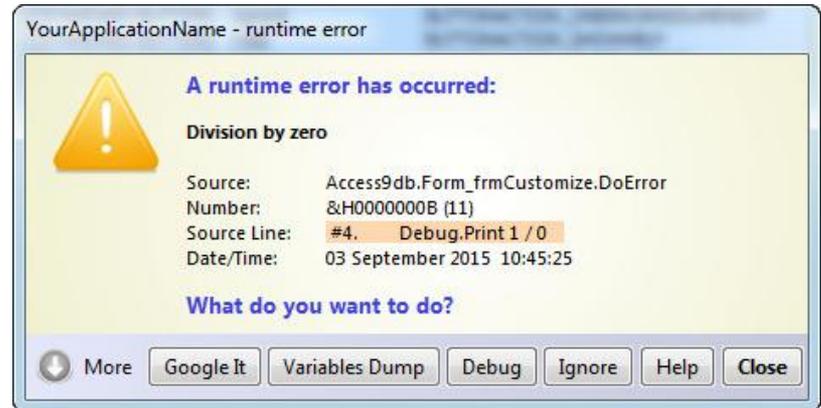
Implementierung

---

# VBWATCHDOG

# vbWatchDog - Steckbrief

- Name des Tools
  - vbWatchDog
- Autor
  - Wayne Phillips
- Download
  - <http://www.everythingaccess.com/vbwatchdog.asp>
- Lizenz / Preis
  - Enterprise / 135 € Ultimate / 165 €
- Zweck / Funktion
  - Professionelle Fehlerbehandlung in VBA



# Installation

- Installation nur auf dem Entwicklungsrechner
- Keine (!!!) Installation auf Rechner der User notwendig
- Es werden Module in den Code eingefügt
- Diese stellen die Funktionalität bereit

# Erste Schritte

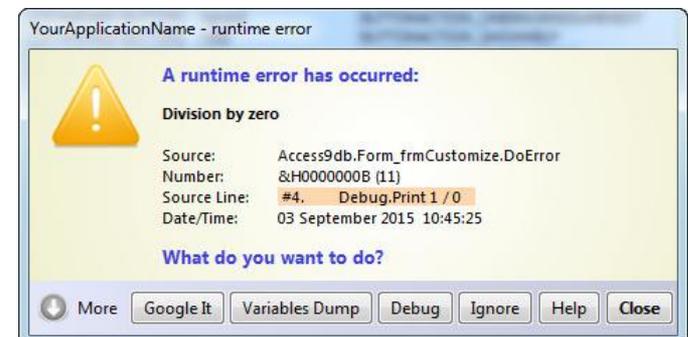
- Module hinzufügen
- Aktivieren
  - `ErrEx.Enable(,,“)`
- Ab sofort steht `vbWatchDog` mit Standardeinstellungen zur Verfügung
- Eigene zentrale Fehlerbehandlung kann definiert werden
  - `ErrEx.Enable(,,MeinZentralerErrorHandler“)`
- Deaktivieren
  - `errEx.Disable`

# Zentraler Error-Handler

- Prozedur, die bei jedem Fehler durchlaufen wird
- Reaktion auf verschiedene Wege möglich
  - OnErrorGoto0
  - OnErrorGotoLabel
  - OnErrorCatch etc
- Idealer Ort, um Fehler im Error-Log zu speichern
- Zentrale Behandlung von Fehlern hier möglich, z.B. 2501 (DoCmd abgebrochen)

# Fehlerdialog

- Kann individuell konfiguriert werden
  - With ErrEx.DialogOptions
- Konfiguration analog zu HTML
- Schlüsselworte werden mit Werten aus dem konkreten Fehler ersetzt
- Buttons können hinzugefügt werden
  - Vorgefertigte Aktionen
    - Debuggen, Ignorieren, Hilfe anzeigen
  - Eigene Aktionen
    - Aufruf Prozedur, z.B. Versand E-Mail



# Verfügbare Informationen

- Folgende Informationen sind (ohne besondere Vorbereitung) verfügbar
  - Modulname
  - Prozedurname
  - Zeilennummer
  - Call-Stack
  - Inhalt aller „aktiven“ Variablen



# Zusätzliche Sprachkonstrukte

- In Anlehnung an .Net
- `ErrEx.Catch ##`
  - Gezielt eine Fehlernummer behandeln
- `ErrEx.CatchAll`
  - Alle anderen Fehlernummern behandeln
- `ErrEx.Finally`
  - Der ideale Ort, zum Aufzuräumen und um geöffnete Objekte zu schließen

```
ErrEx.Catch 11
 'Fehler Nr. 11

ErrEx.CatchAll
 ' Alle anderen Fehler

ErrEx.Finally|
 'Aufräumen
```

# In Anwendung einbauen

- Einfügen in bestehende Anwendung ohne Probleme möglich
- Kein Alles oder Nichts
- Bestehendes Error-Handling und globales Error-Handling arbeiten Hand in Hand
- Anpassung kann Schrittweise erfolgen



# In Anwendung einbauen (II)

- Module einfügen
  - Add-Ins / vbWatchdog / Add vbWatchdog to this project
- Tabelle importieren
- Abfrage importieren
- Modul importieren
- Aktivierung in erster ausgeführter Funktion
  - `Call EnableErrorHandler` einfügen
- Optional: Bei Bedarf / Je nach Wunsch  
Bisheriges Error-Handling Schritt für Schritt anpassen



Links

Links

Links

---

RESSOURCEN

# Links

- FMS: Error Handling and Debugging
  - <http://www.fmsinc.com/tpapers/vbocode/Debug.asp>
- Allen Browne: Error Handling in VBA
  - <http://allenbrowne.com/ser-23a.html>
- H. Habermacher: Error Handling in Access
  - <http://www.dbdev.org/downloads.html#Error>
- Armen Stein: The Extra Resume
  - [https://www.youtube.com/watch?v=\\_0Xahwtk1Bk&feature=youtu.be](https://www.youtube.com/watch?v=_0Xahwtk1Bk&feature=youtu.be)
- UtterAccess: Error Handling
  - [http://www.utteraccess.com/wiki/index.php/Error\\_Handling](http://www.utteraccess.com/wiki/index.php/Error_Handling)
- Jack D. Leach: Rethrowing VBA Errors
  - <http://www.dymeng.com/techblog/rethrowing-vba-errors/>

# Links

- Paul Rohorzka: Form\_Error
  - [http://www.donkarl.com/Downloads/AEK/AEK14\\_Fehlerbehandlung.zip](http://www.donkarl.com/Downloads/AEK/AEK14_Fehlerbehandlung.zip)
- Tom van Stiphout: Handling ODBC Errors
  - <http://www.accessmvp.com/TomvanStiphout/OdbcErrors.htm>
- LastDllError-Eigenschaft
  - [https://msdn.microsoft.com/de-de/library/yb8afh2h\(v=vs.90\).aspx](https://msdn.microsoft.com/de-de/library/yb8afh2h(v=vs.90).aspx)
- API-Funktion FormatMessage
  - [http://www.vbarchiv.net/api/api\\_formatmessage.html](http://www.vbarchiv.net/api/api_formatmessage.html)
- Stack Item Class
  - [https://msdn.microsoft.com/en-us/library/aa227567\(v=vs.60\).aspx](https://msdn.microsoft.com/en-us/library/aa227567(v=vs.60).aspx)

19. Access-Entwickler-Konferenz  
Nürnberg 24./25.09. / Düsseldorf 08./09.10. / Hannover 15./16.10.2016

# FEHLER FÜR FORTGESCHRITTENE

---

Mehr als nur Debug.Print



Thomas Möller, [www.Team-Moeller.de](http://www.Team-Moeller.de)