Tipps für die Programmierung in Access

Franz Fiala ftp://pcnews.at/pcn/65/fiala/access/prg/

Während bei Word oder Excel auch Enduser ohne Programmiererfahrung sehr produktiv sein können, besteht bei der Verwendung von Datenbanken sehr bald die Notwendigkeit, dem Benutzer nur ein (programmiertes) Frontend zu überlassen und ihn alle Eingaben nicht direkt in Tabellen oder Abfragen sondern über Eingabeformulare machen zu lassen. Dazu ist es aber erforderlich, eine Verbindung zwischen dem eingebauten VBA (Visual Basic for Applications) und den Elementen einer Datenbank herzustellen. Die folgenden Beispiele zeigen exemplarisch den Umgang mit VBA in der Variante MS ACCESS.

1. Arbeiten mit Formularen
   Formulare als Programmstartflächen
2. Textausgabe
3. Arbeiten mit Tabellen
   Öffnen einer Tabelle und bearbeiten aller Datensätze
4. Arbeiten mit Abfragen
   Formulierung einer Abfrage mit SQL

1. Arbeiten mit Formularen

Eine ASCII-Datei enthält eine Liste mit Namen, z.B. eine Schülerliste; jede Zeile enthält einen Vor- und einen Familiennamen. Die ASCII-Datei wurde unabhängig von Access mit einem gewöhnlichen Texteditor erstellt und soll in eine Tabelle der Datenbank aufgenommen werden.

Der Programmstart erfolgt über ein Formular, das den Namen "`F_Namensliste-Erfassen`" erhält. In dem Formular gibt es ein Textfeld, in dem der Dateiname der ASCII-Datei einzugeben ist und eine Befehlsschaltfläche, die den Import der ASCII-Daten auslöst. Alle Zeilen der ASCII-Datei werden an eine bestehende Tabelle `T_NAMEN` angehängt.

Die folgenden Zeilen sind eine schrittweise Anleitung zum Herstellen des Formulars und der zugehörigen Programme.

Zur Bezeichnung

Achtung: ACCESS vergibt bei der Generierung eines jeden neuen Objekts einen Namen aus dem der Objekttyp hervorgeht und eine laufende Nummer. Man sollte sich von Beginn an eine geeignete Systematik zurechtlegen, die auf die Semantik des eigenen Projekts zugeschnitten ist und diesen Vorgabenamen dem Projekt anpassen. Man muss dabei beachten, dass Doppelbezeichnungen für Tabellen und Abfragen nicht zulässig sind und es auch sonst im Zuge von Programmen eine einfache Kontrollmaßnahme ist, wenn man aus dem Namen nicht nur die Stellung des Objekts im Projekt sondern auch den Typ erkennen kann.

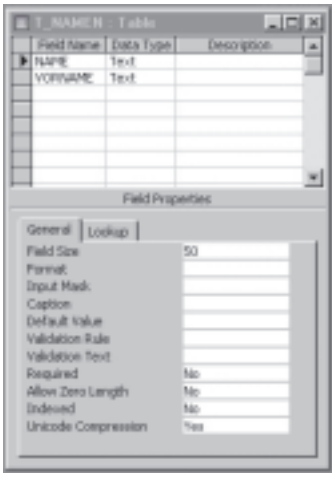
Beispiel

ACCESS vergibt den Namen `Tabelle1` für eine neue Tabelle. Wir ändern den Namen auf `T_NAMEN`. `T_` verweist darauf, dass es eine Tabelle ist, `NAMEN` verweist auf den Inhalt.

ACCESS vergibt den Namen `Form1` für ein neues Formular, wir ändern den Namen unmittelbar durch Speichern unter `F_NamenslisteErfassen`.

Diese unmittelbare Festlegung der Namen ist wichtig, denn bei Generierung von Prozedurnamen werden diese Namen wieder von den Basisnamen abgeleitet und eine spätere Änderung kann mühsam sein.

- ***Datei - Neue Datenbank anlegen***
- 2 Spalten `NAME` und `VORNAME` im Textformat anlegen, kein Primärschlüssel, zunächst keine Einträge (wäre der Autor konsequent gewesen, hätte er auch für die Spalten einen Typ-Präfix gewählt, wie das im Beitrag Webgenerator in diesem Heft der Fall ist).
- Die Tabelle bekommt die Bezeichnung `T_NAMEN`



Design-View der Tabelle T_NAMEN

- ***Neues Formular*** in ***Entwurfsansicht*** erzeugen und gleich auch unter dem Namen ***F_NamenslisteErfassen*** speichern
- Aus dem ***Werkzeugkasten(Toolbox)*** das ***Textfeld*** (nicht Bezeichnungsfeld (Label)) auswählen und rechteckigen Bereich markieren
- Das Textfeld erhält automatisch ein Bezeichnungsfeld mit dem provisorischen Namen `Text0`. Weiters ist das Textfeld "***Ungebunden (unbound)", d.h. nicht mit einem Tabellenfeld verbunden***
- Danach eine Befehlsschaltfläche im Formular zeichnen, z.B. unterhalb des Textfeldes

  *Hinweis: Normalerweise öffnet automatisch der Befehlsschaltflächenassistent, der sehr häufige Anwendungen für Schaltflächen bereithält. Hier nichts auswählen und auf "**Abbrechen**" drücken. Die Bezeichnung der Befehlsschaltfläche wird auf "**Command2**" voreingestellt (die laufende Nummer hängt von der Anzahl der vorangegangenen Versuche ab, auch jener, deren Objekt wieder gelöscht wurde). Die hier verwendete Access-Version stammt aus Office 2000 kann bei Office-97 vielleicht geringfügig anders aussehen, jedenfalls werden die Bezeichnungen nicht in englischer Sprache sein.*

Damit die beiden Bildschirmelemente das Gewünschte tun, müssen die "***Eigenschaften***" eingestellt werden.

Jedes Element auf der Arbeitsfläche besitzt "Eigenschaften" (Properties), die über die rechte Maustaste oder über den Menüpunkt "***Ansicht***"-"***Eigenschaften***" oder über das Symbol "***Eigenschaften***" angewählt werden können.

Entwurfsansicht des Formulars

Eigenschaften des Formulars

- Wechseln zur Entwurfsansicht
- Formular anwählen (Kreuzungspunkt des horizontalen und vertikalen Lineals)
- Im Ordner ***Format***
- Bildlaufleisten ausschalten
- Navigationsschaltflächen auf "***Nein***" stellen
- Min/Max-Buttons ausschalten
- Im Ordner "***Ereignis***" (***Event***) muss beim Laden des Dokuments eine Verbindung zur Tabelle hergestellt werden, an die die Daten aus der Textdatei anzuhängen sind. Daher das Ereignis "***OnLoad***" anwählen und dort den Eintrag "***Ereignisprozedur***" (***Event Procedure***). Ebenso wird eine Prozedur "***OnUnload***" angelegt, bei der Objekte wieder entfernt werden. Bei Kli-

cken auf das Symbol "..." wird VBA gestartet und man findet sich unmittelbar in der Prozedur. Der Prozedurname wird automatisch vergeben und im allgemeinen vom Formularnamen abgeleitet. Da wir im Zusammenhang mit diesem Formular immer mit der aktuellen Datenbank und mit einer Tabelle arbeiten wollen, vereinbaren wir beides als globale Variable. Im Abschnitt ***Declarations*** definieren wir:

```
Option Compare Text
Option Explicit
Dim DB As Database
Dim DS As Recordset
Const Pfad =
"D:\winnt\profiles\pcnews\desktop\65\fiala\a
ccess\"
```

`Option Compare Text` stellt bei Vergleichen von Text ein, dass Umlaute nach dem entsprechenden Grundlaut gereiht werden und nicht nach ihrer ASCII-Reihenfolge, also a<ä<b... und nicht a<b...<ä.

`Option Explizit` erzwingt, dass verwendete Variablen auch vorher definiert werden müssen (unbedingt einstellen, es kann sonst bei Schreibfehlern in langen Objekt- und Variablennamen zu unliebsamen und schwer auffindbaren Überraschungen kommen!)

`DB` ist der Name für ein Datenbankobjekt, `DS` ist der Name für ein Recordset-Objekt. `Pfad` reduziert die Eingabe des Namens auf den Dateinamen und wird später im Programm mit dem eingegebenen Dateinamen verkettet.

In der Prozedur `Form_Load` werden die beiden Objekte initialisiert und es wird der erste Datensatz eingestellt. In der Prozedur `Form_Unload` werden beide Objekte geschlossen.

```
Private Sub Form Load()
  Set DB = CurrentDb()
  Set DS = DB.OpenRecordset("T NAMEN")
  DS.MoveFirst
End Sub

Private Sub Form Unload(Cancel As Integer)
  DS.Close
  DB.Close
End Sub
```

Aufruf der Formulars und Eingabe des Dateinamens neu.txt

Eigenschaften des Textfeldes

- Textfeld auswählen und Eigenschaften anklicken
- Im Ordner "Other" den Namen `T_Dateiname` wählen
- Im Ordner Daten den Standardwert "*Hier den Dateinamen eingeben*"
- Das Textfeld ist mit keinen Programmen verbunden, es wird nur durch andere Programme abgefragt

Eigenschaften der Befehlsschaltfläche

- Befehlsschaltfläche auswählen und Eigenschaften anklicken
- Im Ordner "Other" den Namen `B_DateiLesen` wählen
- Im Ordner ***Ereignis Beim Klicken*** [Ereignisprozedur] auswählen, es erscheint rechts die Schaltfläche "..."
- Wählt man diese an, befindet man sich gleich in jener Prozedur, die ausgelöst wird, wenn der Benutzer diese Schaltfläche anklickt:

```
Private Sub B DateiLesen Click()
End Sub
```

Hier fügt man den gewünschten Kode ein:

```
Private Sub B_DateiLesen_Click()

  Dim Datei As String

  Datei = Pfad + B_Dateiname.Value
  If Dir(Datei) = "" Then
    MsgBox "Datei existiert nicht"
  Else
    Open Datei For Input Access Read As #1
    Dim Zeile As String
    Dim Trennstelle As Long

    Do While Not EOF(1)
      Line Input #1, Zeile
      Debug.Print Zeile
      Zeile = Trim(Zeile)
      Trennstelle = InStr(1, Zeile, " ")
      DS.AddNew
        DS!Vorname = _
          Left(Zeile, Trennstelle)
        DS!Name = _
          Mid(Zeile, Trennstelle + 1, 255)
      DS.Update
    Loop
    Close #1
  End If
End Sub
```

Anfangs wird der als Konstante vordefinierte `Pfad` mit dem eingegebenen Dateinamen `B_Dateiname.Value` zu `Datei` verbunden. Gibt es die Datei nicht, sieht der Benutzer eine Fehlermeldung. Sonst wird die Datei zum Lesen geöffnet und zeilenweise in Zeile eingelesen. Da es sich um Wortepaare handelt, wird nach einem Leerzeichen gesucht. Der erste Teil der Zeile kommt in `DS!Vorname`, der zweite Teil in `DS!Name`.

Anmerkung: Um das Beispiel nicht unübersichtlich werden zu lassen, wurde es unterlassen, mehrfache Leerzeichen oder Tabulatoren als *white space* zu suchen.

2. Anzeige von Text

Für Textausgaben gibt es mehrere Möglichkeiten:

Beispiel 1: Ausgabe des eingegebenen Textes in einer Message Box (F_Beispiel1)

```
MsgBox "Deine Eingabe: " + Text0.Value
```

Achtung: Jedes Element, das man in einem Formular erzeugt, bekommt einen Namen, den man verändern kann und soll. Das Textfeld hat den Namen `Text0` bekommen. Die Eigenschaften des Textfeldes können nicht nur beim Entwurf, sondern auch während des Programmablaufs gesteuert und abgelesen werden. Wichtig im Zusammenhang mit dem Textfeld ist sein Wert "`Value`".

Beispiel 2: Der einzugebende Text ist ein Dateiname, die Datei ist anzuzeigen (F_Beispiel2)

Od die Datei existiert, erfährt man mit dem Befehl `DIR`. Kennt man den Befehl und gibt ihn syntaktisch richtig ein, bekommt man im Kode-Editor gleich eine unmittelbare Hilfe, wie dieser Befehl weiter aufgebaut ist oder (bei Objekten), welche Methoden anwendbar sind. Bei `DIR` muss nur der richtige Pfad eingegeben werden. Am besten legt man sich dazu eine Hilfsdatei, z.B. `temp.txt` an und gibt bei Aufruf des Formulars ein `c:\temp\temp.txt`.

Wir erweitern den Kode um folgende Zeilen:

```
If Dir(Text0.Value) = "" Then
   MsgBox "Datei existiert nicht"
Else
   MsgBox "Datei existiert"
End If
```

Den zweiten Teil der Aufgabe, die Datei anzeigen, kann man auf mehrere Arten lösen:

1. Mit einem DOS-Befehl (`TYPE test.txt`)
2. Mit dem NOTEPAD
3. In das Debug-Fenster
4. Durch Ausgabe in ein anderes Feld

Ad. 1 Ausgabe mit DOS-Befehl `TYPE` (F_Beispiel21)

Bei der genauen Syntax kann man sich mit dem Befehlsschaltflächenassistenten ein bisschen helfen lassen:

Befehlsschaltfläche auf dem Formular aufziehen und jetzt den Befehlsschaltflächenassistenten nicht abbrechen sondern eingeben:

- ***"Anwendung", "Anwendung ausführen"***
- Befehlszeile:
  `CMD /K TYPE C:\TEMP\TEMP.TXT`
- ***"Text"***: Datei ausgeben
- Der Name der Schaltfläche ist `Befehl2`
- Fertigstellen

Der Assistent erzeugt etwa folgenden Kode

```
    Dim stAppName As String
    stAppName = _
```

```
        "CMD /K TYPE C:\TEMP\TEMP.TXT"
    Call Shell(stAppName, 1)
```

Bei Drücken der Schaltfläche wird der Code in einem DOS-Fenster ausgegeben. Das DOS-Fenster schließt man mit EXIT.

Ad. 2 Ausgabe mit NOTEPAD (F_Beispiel22)

Schreibt man `NOTEPAD C:\TEMP\TEMP.TXT`, wird die Datei auf Grund der Dateiendung mit dem Notepad aufgerufen.

Will man jetzt die Benutzereingabe mit den Ergebnissen dieser neuen Schaltfläche verbinden, schreibt man in die Ereignisprozedur der ersten Schaltfläche:

```
Private Sub Befehl2_Click()
 If Dir(Text0.Value) = "" Then
   MsgBox "Datei existiert nicht"
 Else
   Dim stAppName As String
   stAppName = "NOTEPAD " + Text0.Value
   Call Shell(stAppName, 1)
 End If
End Sub
```

Ad 3: Ausgabe in Debug-Fenster (F_Beispiel23)

Zunächst kopiert im Entwurfsmodus die Schaltfläche, oder man überschreibt die bestehende Ereignisprozedur wie folgt:

```
Private Sub Befehl2 Click()
 If Dir(Text0.Value) = "" Then
   MsgBox "Datei existiert nicht"
 Else
   Dim stAppName As String
   Open Text0.Value _
     For Input Access Read As #1
   Dim Zeile As String
   Do While Not EOF(1)
     Line Input #1, Zeile
     Debug.Print Zeile
   Loop
   Close #1
 End If
End Sub
```

Ad 4: Ausgabe in einem eigenen Fenster (F_Beispiel24)

Wenn man dieses Programm testet, darf man nicht vergessen, bei geöffnetem Programmeditor für die Ereignisprozedur mit "Ansicht", "Testfenster" das Testfenster zu öffnen, da die Daten in das Testfenster ausgegeben werden.

Will man statt dessen die Daten in ein Textfeld ausgeben, kann man dafür gleich das bereits bestehende Eingabe-Textfeld (oder ein neu anzulegendes) verwenden.

Statt `Debug.Print Zeile`

Schreibt man

```
Text0.value = Text0.Value + Zeile
```

Beziehungsweise bei einem Bezeichnungsfeld:

```
Bezeichnungsfeld8.Caption =
Bezeichnungsfeld8.Caption
  + Chr(10) + Chr(13) + Zeile
```

3. Arbeiten mit Tabellen

Dieses Beispiel zeigt, wie man die bestehende Tabelle T_NAMEN programmgesteuert bearbeiten kann. Alle Aktivitäten, die man in der Tabellenansicht händisch ausführen kann, können auch mit Visual-Basic durchgeführt werden.

Hier zwei Möglichkeiten:

- Bearbeitung aller Datensätze
- Register ***"Module"***
- ***Neuer Modul***
- ***Einfügen - Prozedur "Daten_editieren"***

Man erhält folgenden Prozedur-Rumpf

```
Public Sub Daten_editieren()
End Sub
```

Das folgende Gerüst kann man im Umgang mit Tabellen oder Abfragen immer verwenden:

```
Public Sub Daten editieren()
Dim Db As Database
Dim Rs As Recordset

Set Db = CurrentDb()
Set Rs = Db.OpenRecordset("T_NAMEN")

Do While Not Rs.EOF()
 Debug.Print Rs!NAME + " " + Rs!VORNAME
 Rs.MoveNext
Loop
Rs.Close
End Sub
```

`Database` ist der Typ einer Datenbank, die Datanbankvariable in diesem Programm ist `Db`. Im allgemeinen wird die eigene Datenbank verwendet (Funktion `CurrentDb()`). `Recordset` ist der Typ einer Tabelle oder Abfrage. `Db.OpenRecordset ("Tabelle1")` heißt, dass in der aktuellen Datenbank `Db` eine Tabelle oder Abfrage mit dem Namen "`T_NAMEN`" geöffnet wird.

Die allgemeine Bearbeitungsschleife für alle Datensätze lautet.

```
Do While Not Rs.EOF()
 'hier kommt die Bearbeitung herein
 Rs.MoveNext
Loop
```

Im obigen Programm werden `NAME` und `VORNAME` mit `debug.print` in das Testfenster augegeben. Will man die Daten editieren (Etwa, um alle `NAME`n auf Kleinbuchstaben zu verwandeln, schreibt man:

```
Rs.Edit
 Rs!NAME = LCase(Rs!NAME )
Rs.Update
```

Wichtig ist, dass man vor der Änderung die Edit-Methode und danach die Update-Methode einfügt.

Getestet wird die Funktion mit dem Testfenster durch Eingabe von `Daten_editieren` und Enter.

4. Arbeiten mit Abfragen

Die Abfrage soll alle Familiennamen von A-L, sortiert nach dem Familiennamen im Programm verarbeiten. Es gibt zwei Möglichkeiten, mit Abfragen zu arbeiten:

Man generiert die Abfrage (`A_NAMEN-AL`) grafisch im Register ***"Abfragen"*** - ***"Neu"***... und verwendet dann die Abfrage wie im obigen Beispiel die Tabelle (statt `T_NAMEN` verwendet man den Begriff `A_NAMEN-AL`). Diese Vorgangsweise ist zwar möglich, doch hat sie den Nachteil, dass jede Änderung der Abfrage sich auch im Programmkode auswirkt und unvorhersehbare Folgen haben kann.

Besser ist es, die Abfrage zuerst grafisch zu entwerfen (wie vorher beschrieben) und danach als SQL-Statement im Kode zu verwenden. (Damit lassen sich alle Arten von Abfragen, wie Tabellenerstellungsabfragen, Aktualisierungsabfragen, Löschabfragen, Auswahlabfragen und Aktualisierungsabfragen einheitlich ausführen.)

- Register ***Abfragen***
- ***Neue Abfrage***
- Entwurfsansicht
- `T_NAMEN` hinzufügen
- "*" in ein Ausgabefeld ziehen (alle Felder sollen bearbeitbar sein)
- `NAME` in ein Ausgabefeld ziehen und dabei Anzeige ausschalten und Sortierreihenfolge ***Aufsteigend*** einschalten
- Als Bedingung gibt man unter Kriterien im `NAME`-Feld an `<="L"`
- Jetzt überprüft man die Ausgabe mit ***"Ansicht" "Datenblattansicht"***
- Wenn alles zufriedenstellend ausschaut, geht man in die SQL-Ansicht und kopiert dien dort befindlichen SQL-Kode in die Zwischenablage:

```
SELECT T_NAMEN.*
FROM T_NAMEN
WHERE (((T_NAMEN.NAME)<="L"))
ORDER BY T_NAMEN.NAME;
```

Diese Kode kann auch ziemlich umfangreich sein aber egal, man muss ihn ja nicht selbst schreiben. Jetzt fügt man den Kode wie folgt in die Prozedur ein:

```
Dim SQLAbfrage As String
SQLAbfrage = _
  "SELECT T_NAMEN.* " &
  "FROM T_NAMEN " & _
  "WHERE (((T_NAMEN.NAME)<=" &
     Chr(34) + "L" + Chr(34) + ")) " & _
  "ORDER BY T_NAMEN.NAME;"
Set Db = CurrentDb()
Set Rs = Db.OpenRecordset(SQLAbfrage)
```

Alles Weitere bleibt gleich. Beim Übertragen der SQL-Anweisung in den String SQLAnweisung tritt das Problem auf, dass das Zeichen (") in BASIC als Begrenzer verwendet wird. Daher muss es mit `Chr(34)` in die Zeichenfolge eingebaut werden. Weiters ist es für die Übersichtlichkeit günstig, nicht eine einzige Zeile zu verwenden, sondern mehrere. Der Unterstrich verbindet die Zeilen, das Ampersand verkettet die Strings. Achtung auf die Leerzeichen dazwischen.