HARALD NAHRSTEDT

Excel + VBA

Ergänzungen

Kapitel

Einführung in VBA

Typisierte Random Dateien unter VBA

Erstellt am 12.02.2012

Beschreibung

Dieses Kapitel behandelt die Möglichkeit, Random-Dateien mit fester Datensatzlänge zu verwalten. Verwendet wird dazu ein typisierter Satzaufbau. Damit die Datensätze übersichtlich bleiben, wird ein kleiner Trick benutzt. In einem weiteren Schritt wird die Verwaltung der Datensätze nach Schlüsseln gezeigt. Der erste Schritt zum Aufbau von relationalen Datenbanken.

1 Typisierte Random Dateien

Random-Dateien lassen, wie der Name schon sagt, einen wahlfreien Zugriff auf Datensätze zu. Anders wie bei sequentiellen Dateien, kann ein beliebiger Datensatz jederzeit gelesen und beschrieben werden. Grundvoraussetzung ist eine feste Datensatzlänge. Da bietet sich die Definition eines Datensatz als benutzerdefinierten Typ an.

Als einfaches Beispiel benutzen wir den Aufbau eines Adress-Datensatzes. Alle Elemente sind vom Typ String und haben eine feste Länge (* n), so dass auch deren Summe und damit der Datensatz eine feste Länge besitzt. Ein kleiner Trick, damit die Datei später auch mit einem einfachen Texteditor lesbar ist, ist dass Anfügen der Steuerzeichen Crt für Carriage Return (Wagenrücklauf) und Lfd für Line Feed (Zeilenvorschub).

```
'Modul: mod_01_AdressDatei
Option Explicit

Type Adressen
   Nummer   As String * 3
   Name     As String * 20
   Vorname  As String * 20
   Strasse  As String * 30
   PLZ      As String * 10
   Ort      As String * 30
   Crt      As String * 1
   Lfd      As String * 1
End Type

Public Adr_Len As Long 'Länge des Datensatzes
Public Adr_Num As Long 'Aktuelle Dateinummer
Public Adr_Max As Long 'Maximale Anzahl vorhandener Datensätze
Public Adr_Rec As Long 'aktuelle Datensatznummer
```

Die vier Long-Definitionen beinhalten Werte, die den Umgang mit der Datei vereinfachen. Diesen Code setzen wir in ein Modul mit dem Dateinamen. Außerdem bekommt dieses Modul noch elementare Funktionen zur Handhabung der Datei.

Die allgemeine Syntax zum Öffnen einer Randomdatei lautet

```
Open Dateiname For Random Shared As Dateinummer Len = Satzlänge
```

Parameter	Beschreibung
Dateiname	Pfad und Dateiname der zu öffnenden Datei.
Dateinummer	Gültig im Bereich von 1 bis 511. Mit der *FreeFile*-Funktion erhält man die nächste verfügbare Dateinummer. Unter dieser Nummer erfolgen dann alle weitere Zugriffe des Programms auf die Datei.

Damit können wir nun eine Funktion schreiben, die Pfad und Dateinamen als Parameter bekommt. Außerdem ist noch die Angabe der Datensatzlänge erforderlich. In der Funktion

wird zunächst das Vorhandensein der Datei abgefragt, denn wir wollen nicht unbedingt jedesmal eine Datei erzeugen mit der ersten Abfrage, falls es sie noch nicht gibt. Für das Erstellen einer neuen Datei schreiben wir eine eigene Prozedur. Dazu öffnen wir lediglich die Datei kurz im Modus Output. In diesem Modus wird einfach nur eine Dateihülle angelegt. Sollte, wider Erwarten, eine Datei mit gleichem Namen existieren, wird sie zerstört. Natürlich muss auch eine Datei gelöscht werden können. Dies übernimmt die Funktion *AdrDatei_Löschen*.

```
Public Function AdrDatei_Öffnen(Datei As String, Länge As Long) _
   As Boolean
   Dim Data As Adressen

   If Not FileExist(Datei) Then
      AdrDatei_Öffnen = False
      Exit Function
   End If
   AdrDatei_Öffnen = True
   Adr_Num = FreeFile
   Open Datei For Random Shared As Adr_Num Len = Länge
   Adr_Max = LOF(Adr_Num) / Länge
   Do
      If Adr_Max > 0 Then
         Get Adr_Num, Adr_Max, Data
         If Data.Nummer = Space(Len(Data.Nummer)) Then
            Adr_Max = Adr_Max - 1
         End If
      End If
   Loop While Data.Nummer = Space(Len(Data.Nummer)) And _
      Adr_Max > 0
End Function

Public Function AdrDatei_Erstellen(Datei As String) As Boolean
   If FileExist(Datei) Then
      AdrDatei_Erstellen = False
      Exit Function
   End If
   Adr_Num = FreeFile
   Open Datei For Output As Adr_Num
   Close Adr_Num
   AdrDatei_Erstellen = True
End Function

Public Function AdrDatei_Löschen(Datei As String) As Boolean
   If FileExist(Datei) Then
      Kill Datei
      AdrDatei_Löschen = True
   Else
      AdrDatei_Löschen = False
   End If
End Function
```

Hilfsprozeduren bekommen ein eigenes Modul.

```
'Modul: mod_09_Hilfsprozeduren
Option Explicit

Public Function FileExist(Dateiname As String) As Boolean
   On Error GoTo ErrorHandler
   FileExist = Dir(Dateiname) <> ""
   Exit Function
ErrorHandler:
   FileExist = False
   Resume Next
End Function
```

Dermaßen ausgerüstet können wir nun beginnen, mit dieser Datei zu arbeiten. Diese Prozeduren setzen wir ebenfalls in ein eigenes Code-Modul. Die nachfolgende Prozedur öffnet die Datei und schreibt einen Datensatz in die Datei, und schließt sie wieder. Die Eingabe werden abgefragt und es werden daraus vereinfachte Datensätze geformt. Die zweite Prozedur stellt einen vorhandenen Datensatz in das Formblatt.

```
'Modul: mod_02_AdressVerwaltung
Option Explicit

Public Const FileName = "C:\Temp\Random\Daten.dat"

Public Sub Adresse_Erzeugen()
   Dim Data       As Adressen
   Dim Text       As String
   Dim Key        As String
   Dim Nummer     As Long

   If Not ActiveSheet.Name = "Adressen_Formular" Then Exit Sub

   If AdrDatei_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not AdrDatei_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         End If
      Else
         Exit Sub
      End If
   End If
   Nummer = InputBox("Schlüssel = ")
   Key = Right("000" & Trim(Str(Nummer)), 3)
   Adr_Rec = InputBox("Satznummer = ")
   LSet Data.Nummer = Key
   LSet Data.Name = "Name " & Key
   LSet Data.Vorname = "Vorname " & Key
   LSet Data.Strasse = "Strasse " & Key
   LSet Data.PLZ = "D-PLZ " & Key
   LSet Data.Ort = "Wohnort " & Key
   LSet Data.Crt = vbCr
```

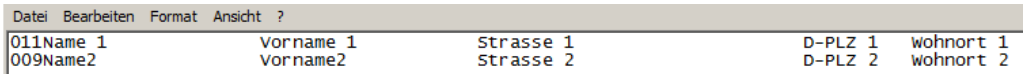
```
   LSet Data.Lfd = vbLf
   Put Adr_Num, Adr_Rec, Data
   Close Adr_Num
End Sub

Public Sub Adresse_Lesen()
   Dim Data       As Adressen
   Dim Text       As String
   Dim Key        As String
   Dim Nummer     As Long

   If Not ActiveSheet.Name = "Adressen_Formular" Then Exit Sub

   If AdrDatei_Öffnen(FileName, Len(Data)) = True Then
      Adr_Rec = InputBox("Satznummer = ")
      Get Adr_Num, Adr_Rec, Data
      With wshAdressen_Formular
         .Cells(1, 2) = Data.Nummer
         .Cells(2, 2) = Data.Name
         .Cells(3, 2) = Data.Vorname
         .Cells(4, 2) = Data.Strasse
         .Cells(5, 2) = Data.PLZ
         .Cells(6, 2) = Data.Ort
      End With
   End If
   Close Adr_Num
End Sub
```

Der Inhalt in der Datei, sieht mit einem einfachen Texteditor gelesen, dank der Steuerzeichen so aus:



Die nachfolgende Prozedur liest den Inhalt der Datei aus und schreibt ihn in ein Arbeitsblatt der Excel-Datei.

```
Public Sub Adressen_Lesen()
   Dim Data       As Adressen
   Dim lCount     As Long

   wshAdressen_Liste.Activate
   wshAdressen_Liste.UsedRange.Clear
   If AdrDatei_Öffnen(FileName, Len(Data)) = True Then
      For lCount = 1 To Adr_Max
         Get Adr_Num, lCount, Data
         With ActiveSheet
            .Cells(lCount, 1) = Data.Nummer
            .Cells(lCount, 2) = Data.Name
            .Cells(lCount, 3) = Data.Vorname
            .Cells(lCount, 4) = Data.Strasse
            .Cells(lCount, 5) = Data.PLZ
```

```
            .Cells(lCount, 6) = Data.Ort
         End With
      Next lCount
   End If
   Close Adr_Num
End Sub
```

Danach hat das Arbeitsblatt ebenfalls die Daten der Datei.

	A	B	C	D	E	F
1	11	Name 1	Vorname 1	Strasse 1	D-PLZ 1	Wohnort 1
2	9	Name2	Vorname2	Strasse 2	D-PLZ 2	Wohnort 2

Nun fehlt lediglich noch die Löschprozedur.

```
Public Sub Adressen_Löschen()
   Dim Text       As String

   If AdrDatei_Löschen(FileName) Then
      Text = "Datei " & FileName & " gelöscht!"
      MsgBox Text
   Else
      Text = "Datei " & FileName & " nicht gelöscht!"
      MsgBox Text
   End If
End Sub
```

2 Die Problematik des Wahlfreien Zugriffs

In dem vorangegangenen Beispiel haben wir die Datensätze hintereinander in die Datei geschrieben. Doch was ist, wenn wir die Daten des 2. Datensatz als 3. Datensatz speichern. Löschen wir also die Datei und führen wir einen Test aus.

Das Schreiben und das Lesen erfolgt problemlos. Wir finden auch die Datenelemente im Arbeitsblatt richtig dargestellt.

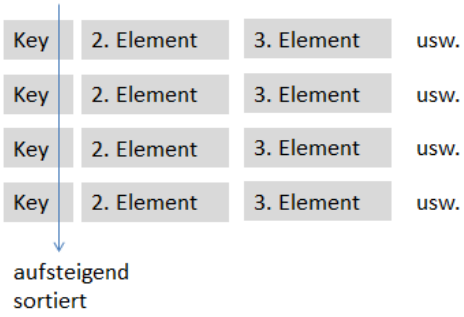
	A	B	C	D	E	F
1	11	Name 1	Vorname 1	Strasse 1	D-PLZ 1	Wohnort 1
2						
3	9	Name2	Vorname2	Strasse 2	D-PLZ 2	Wohnort 2

Nur das Lesen in der Datei mit dem Texteditor zeigt eine Veränderung.

Der Platz für den 2. Datensatz wird durch Leerzeichen besetzt. Leider gilt dies auch für die Steuerzeichen Crt und Lfd, so dass sich der 3. Datensatz nahtlos anschließt. Erst wenn auch der 2. Datensatz beschrieben wurde, ist auch die Darstellung im Texteditor wieder homogen.

3 Die Einführung eines Primärschlüssels

Natürlich kann man die Personendaten auch so speichern, dass ihre Personalnummer auch der Datensatznummer entsprechen. Doch dies kann unter Umständen sehr viele Leersätze erzeugen. Sinnvoller ist die Aneinanderreihung von Datensätzen in sortierter Form. Dazu muss ein Datensatzelement als Primärschlüssel definiert werden. Der Einfachheit halber nehmen wir dazu das erste Element. Das muss zwar nicht sein, macht die Programmierung aber etwas einfacher. Für unser Beispiel nehmen wir die bisher wenig genutzte Nummer. Es könnte aber auch der Name sein. Dann gibt es wieder das Problem der Namensgleichheit, also könnte man den Vornamen mit dazu nehmen. Sind die auch wieder gleich, muss man zusätzlich einen Schlüssel einführen. Zum Beispiel die ersten zwei Buchstaben vom Namen plus der ersten zwei Buchstaben vom Vornamen plus einer fortlaufenden Nummer. Letztlich geht es um die Eindeutigkeit eines Schlüssels. Denn dies ist die Grundvoraussetzung eines Primärschlüssels. Damit bekommt unsere Random Datei folgenden Aufbau.



4 Die Bisektionsmethode

Natürlich könnte man eine sortierte Datei sequentiell lesen, um den Ort zu finden, an der man einen neuen Schlüssel einfügen sollte. Doch diese Methode ist bei großen Datenmengen einfach zu langsam. Daher benutzen wir die Bisektionsmethode.

Ich will sie an einem einfachen Beispiel erklären. Angenommen, Sie suchen einen Namen im Telefonbuch. Dann schlagen Sie es in der Mitte auf. Entweder steht dann der Name in der linken oder rechten Hälfte. Nehmen wir an, er steht in der rechten. Dann teilen Sie diese Hälfte wieder in der Mitte und entscheiden wie zuvor. So kommen Sie relativ schnell auf die Seite mit dem Namen. In meinem Buch *Algorithmen für Ingenieure* finden Sie ein VBA-Beispiel zu diesem Thema, und nachfolgend sehen Sie daraus das Struktogramm.

Eingabe des Suchbegriffs		
v=1		
b=Anzahl Einträge		
	So lange b>=v	
	$m = Int(\frac{v+b}{2})$	
	Ist t > Listeneintrag in Zeile m	
	Ja	Nein
	v=m+1	b=m-1
	i=v	
Ausgabe Listeneintrag in Zeile i		

5 Sortiert Typisierte Random Dateien

In einem ersten Schritt benötigen wir eine Prozedur, die uns die Datensatznummer liefert, an der der Datensatz nach seinem Primärschlüssel einsortiert werden soll. Undzwar auch dann, wenn die Datei erst angelegt, also noch leer ist.

Die nachfolgende Funktion findet mithilfe der Bisektionsmethode diese Position und liefert sie als Parameter *Rec*. Die Funktion hat außerdem den Wert *True*, falls es den Schlüssel bereits gibt. Sie wird im Modul für die Dateifunktionen mit aufgenommen.

```
Public Function Adressen_Index _
   (Key As String, Max As Long, Rec As Long, Data As Adressen) As
Boolean
   Dim lVon As Long
   Dim lBis As Long
   Dim lNeu As Long

   lVon = 1
   lBis = Max
   Do While lBis >= lVon
      lNeu = Int((lVon + lBis) / 2)
      Get Dat_Num, lNeu, Data
      If Key > Data.Nummer Then
         lVon = lNeu + 1
      Else
         If Key = Data.Nummer Then
            lVon = lNeu
            lBis = lVon - 1
```

```
         Else
            lBis = lNeu - 1
         End If
      End If
   Loop
   Rec = lVon
   If Key = Data.Nummer Then
      Adressen_Index = True
   Else
      Adressen_Index = False
   End If
End Function
```

Die ursprüngliche Prozedur zum Schreiben in eine Random Datei wird um diese Funktion ergänzt und hat nun folgenden Aufbau. Alle Prozeduren zur Nutzung über den Primärschlüssel bekommen ein neues Code-Modul.

```
Public Sub Index_Schreiben_1()
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As Long
   Dim Existent   As Boolean

   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Nummer = InputBox("Schlüssel = ")
   Key = Right("000" & Trim(Str(Nummer)), 3)
   Existent = Adressen_Index(Key, Dat_Max, Dat_Rec, Data)
   If Existent = True Then
      Text = "Daten überschreiben?"
      If MsgBox(Text, vbYesNo) = vbNo Then
         Exit Sub
```

```
      End If
   End If
   LSet Data.Nummer = Key
   LSet Data.Name = "Name" & Key
   LSet Data.Vorname = "Vorname" & Key
   LSet Data.Strasse = "Strasse" & Key
   LSet Data.PLZ = "D-PLZ" & Key
   LSet Data.Ort = "Wohnort" & Key
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf
   Put Dat_Num, Dat_Rec, Data

   Close Dat_Num
End Sub
```

Die Prozedur hat eine vereinfachte Eingabe, damit alle Test ohne großen Aufwand durchgeführt werden können.

So weit, so gut. Die Prozedur, und damit auch die Funktion, arbeiten ohne Probleme. Nur ist es diesmal mit einem einfachen Put-Befehl nicht mehr getan. Denn für den ersten Datensatz mit dem Schlüssel *011* erhalten wir die Datensatznummer 1. Für den zweiten Datensatz mit dem Schlüssel *009* aber auch. Das ist korrekt, denn an der Stelle, an der der erste Datensatz steht, sollte jetzt der zweite stehen, und der erste auf der Datensatznummer 2. Doch hier wird es nur überschrieben. Wir benötigen also noch neben einer Funktion zum Schreiben noch eine Funktion zum Einfügen, mit der wir zukünftig alle Datensätze schreiben.

Zunächst die Funktion für das Schreiben eines Datensatzes an eine bestimmte Datensatznummer.

```
Public Function Adressen_Schreiben _
   (Rec As Long, Data As Adressen) As Boolean

   On Error GoTo Error_Schreiben
   Put Dat_Num, Rec, Data
   Adressen_Schreiben = True
   Exit Function
Error_Schreiben:
   Adressen_Schreiben = False
End Function
```

Und nun die Funktion zum Einfügen eines Datensatzes an eine bestimmte Datensatznummer.

```
Public Function Adressen_Einfügen _
   (Rec As Long, Max As Long, Data As Adressen) As Boolean
   Dim Datax As Adressen
   Dim lNeu  As Long

   On Error GoTo Error_Einfügen
   If Rec <= Max Then
      For lNeu = Max To Rec Step -1
         Get Dat_Num, lNeu, Datax
```

```
         Put Dat_Num, lNeu + 1, Datax
      Next lNeu
   End If
   Put Dat_Num, Rec, Data
   Max = Max + 1
   Adressen_Einfügen = True
   Exit Function
Error_Einfügen:
   Adressen_Einfügen = False
End Function
```

Um diese beiden Funktionen nutzen zu können, muss die Schreib-Prozedur noch einmal abgewandelt werden.

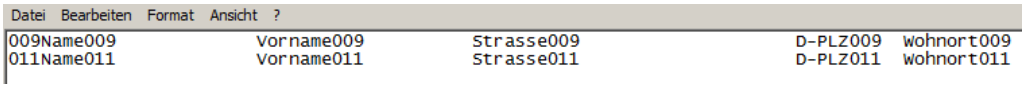
```
Public Sub Index_Schreiben()
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As Long
   Dim Existent   As Boolean

   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Nummer = InputBox("Schlüssel = ")
   Key = Right("000" & Trim(Str(Nummer)), 3)
   Existent = Adressen_Index(Key, Dat_Max, Dat_Rec, Data)
   If Existent = True Then
      Text = "Daten überschreiben?"
      If MsgBox(Text, vbYesNo) = vbNo Then
         Exit Sub
      End If
   End If
   LSet Data.Nummer = Key
```

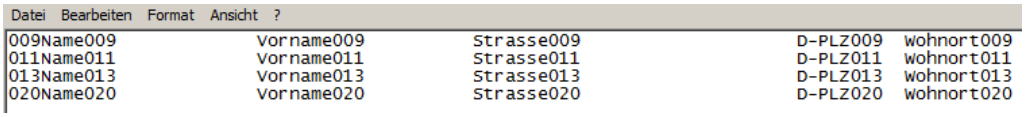
```
    LSet Data.Name = "Name" & Key
    LSet Data.Vorname = "Vorname" & Key
    LSet Data.Strasse = "Strasse" & Key
    LSet Data.PLZ = "D-PLZ" & Key
    LSet Data.Ort = "Wohnort" & Key
    LSet Data.Crt = vbCr
    LSet Data.Lfd = vbLf
    If Existent = True Then
       If Adressen_Schreiben(Dat_Rec, Data) = False Then
          Text = "Datensatz konnte nicht geschrieben werden!"
          MsgBox Text
       End If
    Else
       If Adressen_Einfügen(Dat_Rec, Dat_Max, Data) = False Then
          Text = "Datensatz konnte nicht eingefügt werden!"
          MsgBox Text
       End If
    End If
    Close Dat_Num
End Sub
```

Ein Blick mit einem Texteditor in die Datei zeigt, dass die Einfüge-Prozedur funktioniert.



```
Datei  Bearbeiten  Format  Ansicht  ?
009Name009          Vorname009      Strasse009          D-PLZ009  Wohnort009
011Name011          Vorname011      Strasse011          D-PLZ011  Wohnort011
```

Erneute Aufrufe der Schreib-Prozedur mit anderen Schlüsseln zeigen, dass auch weitere Datensätze richtig einsortiert werden.



```
Datei  Bearbeiten  Format  Ansicht  ?
009Name009          Vorname009      Strasse009          D-PLZ009  Wohnort009
011Name011          Vorname011      Strasse011          D-PLZ011  Wohnort011
013Name013          Vorname013      Strasse013          D-PLZ013  Wohnort013
020Name020          Vorname020      Strasse020          D-PLZ020  Wohnort020
```

Letztlich fehlt noch eine Funktion, die einen Datensatz auch wieder entfernen kann. Durch das Entfernen müssen natürlich allen nachfolgenden Datensätze vorgezogen werden.

```
Public Function Adressen_Entfernen _
    (Rec As Long, Max As Long) As Boolean
    Dim Datax As Adressen
    Dim lNeu  As Long

    On Error GoTo Error_Entfernen
    If Rec <= Max Then
       For lNeu = Rec + 1 To Max
          Get Dat_Num, lNeu, Datax
          Put Dat_Num, lNeu - 1, Datax
       Next lNeu
    End If
```

```
   LSet Datax.Nummer = ""
   LSet Datax.Name = ""
   LSet Datax.Vorname = ""
   LSet Datax.Strasse = ""
   LSet Datax.PLZ = ""
   LSet Datax.Ort = ""
   LSet Datax.Crt = vbCr
   LSet Datax.Lfd = vbLf
   Put Dat_Num, Max, Datax
   Max = Max - 1
   Adressen_Entfernen = True
   Exit Function
Error_Entfernen:
   Adressen_Entfernen = False
End Function
```

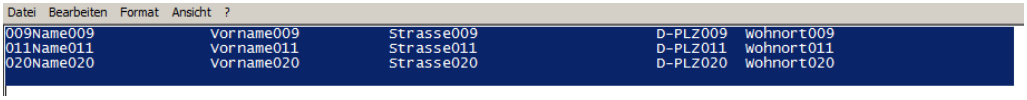
Getestet wird die Funktion mit der nachfolgenden Prozedur.

```
Public Sub Index_Entfernen()
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As String

   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht!"
      Exit Sub
   End If

   Nummer = InputBox("Schlüssel = ")
   Key = Right("000" & Trim(Str(Nummer)), 3)
   If Adressen_Index(Key, Dat_Max, Dat_Rec, Data) = True Then
      If Adressen_Entfernen(Dat_Rec, Dat_Max) = False Then
         Text = "Datensatz wurde nicht gelöscht!"
         MsgBox Text
      Else
         Text = "Datensatz gelöscht!"
         MsgBox Text
      End If
   Else
      Text = "Schlüssel nicht vorhanden!"
      MsgBox Text
   End If
End Sub
```

Das Ergebnis zeigt eine richtige Funktion und ein kleines Problem.



Durch das Entfernen von Datensätzen sammeln sich am Ende der Datei leere Datensätze, die aber nicht belegt sind. Sie dürfen bei der Bestimmung der Anzahl vorhandener Datensätze nicht berücksichtigt werden. Die in der Funktion Adressen_Erstellen vorhandene Anweisung

```
    Dat_Max = LOF(Dat_Num) / Länge
```

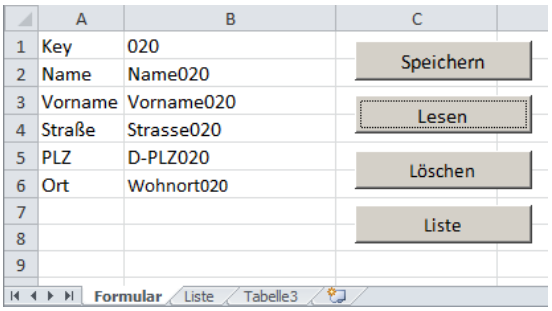
muss im Anschluss noch wie folgt korrigiert werden.

```
    Do
       If Dat_Max > 0 Then
          Get Dat_Num, Dat_Max, Datax
          If Datax.Nummer = Space(Len(Datax.Nummer)) Then
             Dat_Max = Dat_Max - 1
          End If
       End If
    Loop While Datax.Nummer = Space(Len(Datax.Nummer)) And _
       Dat_Max > 0
```

Nun wird beim nächsten Öffnen der Datei die richtige Satzanzahl erzeugt und weitere Datensätze können hinzugefügt und gelöscht werden.

6 Formulare und Listen

Natürlich macht es keinen Sinn, jedesmal eine Prozedur zu schreiben, wenn man einen Datensatz hinzufügen möchte. Dazu verwenden wir ein Arbeitsblatt als Formular. Es könnte auch eine UserForm dazu erstellt werden.



Das Formulat verfügt über 4 ActivX-Steuerelemente der Form CommandButton, die die nachfolgenden Event-Prozeduren im Codefenster des Arbeitsblattes Formular besitzen. Darin finden sich die Inhalte unserer Testprozeduren in etwas angewandelter Form wieder.

Die Schreibprozedur erwartet als Angaben Werte im Bereich B1:B6. Sie berücksichtigt auch bereits vorhandene Schlüssel.

```
Private Sub cbnSpeichern_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As Long
   Dim Existent   As Boolean

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   Existent = Adressen_Index(Key, Dat_Max, Dat_Rec, Data)
   If Existent = True Then
      Text = "Daten überschreiben?"
      If MsgBox(Text, vbYesNo) = vbNo Then
         Exit Sub
      End If
   End If
   LSet Data.Nummer = Key
   LSet Data.Name = Cells(2, 2)
   LSet Data.Vorname = Cells(3, 2)
   LSet Data.Strasse = Cells(4, 2)
   LSet Data.PLZ = Cells(5, 2)
   LSet Data.Ort = Cells(6, 2)
   LSet Data.Crt = vbCr
   LSet Data.Lfd = vbLf
   If Existent = True Then
      If Adressen_Schreiben(Dat_Rec, Data) = False Then
         Text = "Datensatz konnte nicht geschrieben werden!"
         MsgBox Text
```

```
      End If
   Else
      If Adressen_Einfügen(Dat_Rec, Dat_Max, Data) = False Then
         Text = "Datensatz konnte nicht eingefügt werden!"
         MsgBox Text
      End If
   End If
   Close Dat_Num
End Sub
```

Die Leseprozedur erwartet den zu lesenden Schlüssel in B1. Wird der Schlüssel gefunden, werden alle Datensatzelemente in die Zellen B2:B6 geschrieben.

```
Private Sub cbnLesen_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key As String

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht! Erstellen?"
      If MsgBox(Text, vbYesNo) = vbYes Then
         If Not Adressen_Erstellen(FileName) Then
            Text = " Datei " & FileName & " wurde nicht erstellt!"
            MsgBox Text
            Exit Sub
         Else
            If Adressen_Öffnen(FileName, Len(Data)) = False Then
               Text = "Datei " & FileName & _
                  " wurde nicht erstellen?"
               MsgBox Text
               Exit Sub
            End If
         End If
      Else
         Exit Sub
      End If
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   If Adressen_Index(Key, Dat_Max, Dat_Rec, Data) = True Then
      Cells(2, 2) = Data.Name
      Cells(3, 2) = Data.Vorname
      Cells(4, 2) = Data.Strasse
      Cells(5, 2) = Data.PLZ
      Cells(6, 2) = Data.Ort
   Else
      Cells(2, 2) = ""
      Cells(3, 2) = ""
      Cells(4, 2) = ""
```

```
      Cells(5, 2) = ""
      Cells(6, 2) = ""
      Text = "Datensatz nicht gefunden!"
      MsgBox Text
   End If
   Close Dat_Num
End Sub
```

Die Löschprozedur erwartet den zu löschenden Schlüssel in B1. Wird der Schlüssel gefunden, wird der Datensatz entfernt

```
Private Sub cbnLöschen_Click()
   Dim FileName   As String
   Dim Data       As Adressen
   Dim Text       As String
   Dim Dat_Rec    As Long
   Dim Dat_Txt    As String
   Dim Key        As String
   Dim Nummer     As String

   FileName = "C:\Temp\Random\Daten.dat"
   If Adressen_Öffnen(FileName, Len(Data)) = False Then
      Text = "Datei " & FileName & " existiert nicht!"
      Exit Sub
   End If

   Key = Right("000" & Trim(Str(Val(Cells(1, 2)))), 3)
   If Adressen_Index(Key, Dat_Max, Dat_Rec, Data) = True Then
      If Adressen_Entfernen(Dat_Rec, Dat_Max) = False Then
         Text = "Datensatz wurde nicht gelöscht!"
         MsgBox Text
      Else
         Text = "Datensatz gelöscht!"
         MsgBox Text
      End If
   Else
      Text = "Schlüssel nicht vorhanden!"
      MsgBox Text
   End If
End Sub
```

Die Listenprozedur erstellt im Arbeitsblatt Liste eine Übersicht aller vorhandenen Datensätze.

```
Private Sub cbnListe_Click()
   Dim wshListe   As Worksheet
   Dim FileName   As String
   Dim Data       As Adressen
   Dim lCount     As Long

   Set wshListe = Worksheets("Liste")
   wshListe.Activate
```

```
    wshListe.Cells.Clear
    FileName = "C:\Temp\Random\Daten.dat"
    If Adressen_Öffnen(FileName, Len(Data)) = True Then
       For lCount = 1 To Dat_Max
          Get Dat_Num, lCount, Data
          With wshListe
             .Cells(lCount, 1) = Data.Nummer
             .Cells(lCount, 2) = Data.Name
             .Cells(lCount, 3) = Data.Vorname
             .Cells(lCount, 4) = Data.Strasse
             .Cells(lCount, 5) = Data.PLZ
             .Cells(lCount, 6) = Data.Ort
          End With
       Next lCount
       wshListe.Range("A:F").Columns.AutoFit
    Else
       MsgBox "Lesevorgang abgebrochen!"
    End If
    Close Dat_Num
End Sub
```

Nachfolgend eine Ergebnis aus den Tests.

	A	B	C	D	E	F	G	H
1	9	Name009	Vorname009	Strasse009	D-PLZ009	Wohnort009	Formular	
2	11	Name011	Vorname011	Strasse011	D-PLZ011	Wohnort011		
3	20	Name020	Vorname020	Strasse020	D-PLZ020	Wohnort020		
4								

Das Arbeitsblatt der Liste enthält eine Schaltfläche Formular, mit der über die nachfolgende Event-Prozedur zurück zur Formularanzeige geschaltet werden kann. Die Prozedur beinhaltet auch, dass der markierte Datensatz (oder ein Element davon) beim Umschalten auf das Formular darin abgebildet wird. So lassen sich Änderungen am Datensatz, die Kopie eines Datensatzes oder dessen Löschung relativ schnell durchführen.

6 Formulare als UserForm

Natürlich lässt sich das Handling der Daten noch verbessern. Zum Beispiel, indem das Formular als Userform erstellt wird, die bei der Aktivierung der Liste automatisch eingeblendet wird.

Normalisierung und ER-Modell

Das relationale Datenmodell wurde 1970 von dem Mathematiker Codd entwickelt und mithilfe der Mengentheorie beschrieben. Dieses Modell bildet die Basis für Relationale Datenbanken.

Eine Relation ist in der DB-Sprache eine Beziehung zwischen Tabellen mit Daten in Spalten und Zeilen, die Informationen zu bestimmten Objekten beinhaltet. Genauso ist es auch bei den Excel-Tabellen.

Eine Tabelle ist gekennzeichnet durch:

- Einen eindeutigen Namen. Auch eine Excel-Tabelle besitzt in einer Arbeitsmappe einen eindeutigen Namen.
- Die Tabelle hat mindestens eine, meistens mehrere Attribute (Spalten).
- Eine Tabelle hat keine bis beliebig viele Datensätze (Zeilen).
- Den Schnittpunkt von Zeilen und Spalten bilden die Attribute (Zellen).
- Einen Primärschlüssel, der jeden Datensatz eindeutig identifiziert, und dessen Wert sich während der Existenz des Datensatzes nicht ändert. Bei einer Excel-Tabelle ist dies im einfachsten Fall die Zeilennummer.
- Keine oder mehrere Sekundärschlüssel, die auf Primärschlüssel verweisen.