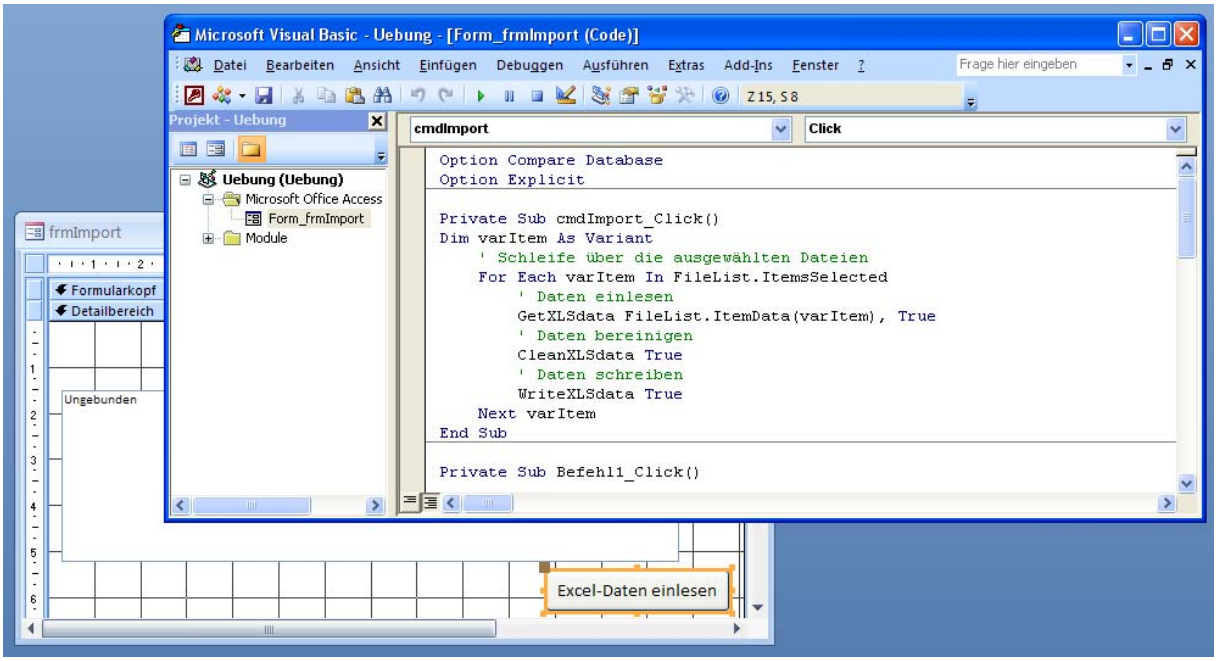
Handbuch zum Selbststudium
und für Schulungskurse

David Niegisch

Visual Basic for Applications (VBA) in Access-Datenbanken



Mai 2010, Version 0.2

Inhaltsverzeichnis

1 Einführung

1.1 An die Leser

Dieses Handbuch soll Ihnen einen möglichst einfachen, praxisnahen und schnellen Einstieg in die VBA-Programmierung unter Access ermöglichen. Anhand ausgewählter Übungen sollen grundsätzliche Funktionen und Methoden beim Einsatz von VBA in einer Access-Umgebung dargestellt werden.

Für ausführliche Informationen zu den einzelnen Themen, insbesondere eine ausführliche Einführung in die VBA-Programmierung, beachten Sie bitte die Linksammlung im Anhang.

Dieses Handbuch sowie sämtliche Beispiel-Daten, die für die folgenden Übungen benötigt werden, finden Sie online unter:

http://yahg.net/kurse/access/vba/

Im Verlauf der folgenden Übungen werden Sie eine Access-Datenbank zur Verwaltung einer Basketball-Liga erstellen. In diesem Beispiel wurde die Liga-Verwaltung bislang lediglich in und mit Excel gepflegt. Diese bereits vorhandenen Daten sollen jedoch in der künftigen Datenbank-Lösung ebenfalls abrufbar sein, müssen also in die Datenbank importiert werden. Ebenso werden stets neue Excel-Dateien zu importieren sein, weshalb der Importvorgang größtenteils automatisiert werden soll.

Sie werden auch eine Methode zur Erstellung einer Benutzungsoberfläche für Ihre Access-Datenbanken kennenlernen, die sehr leicht an die eigenen Wünsche angepasst werden kann und daher für die meisten Access-Projekte gut geeignet ist. In diesem Zusammenhang werden Sie einige anspruchsvolle Suchmöglichkeiten und entsprechende Ergebnisdarstellungen umsetzen.

Das Skript endet mit dem Export beliebiger Datensätze als Excel-Dateien.

Für sämtliche Aufgaben werden Lösungen mit Hilfe von VBA aufgezeigt und der VBA-Code erklärt.

1.2 Visual Basic vs. Visual Basic for Applications

Visual Basic (VB) ist eine eigenständige Programmiersprache zur Erstellung voneinander unabhängiger Softwarekomponenten, wie z. B. ausführbaren Programmen.

Visual Basic for Applications (VBA) ist eine auf Visual Basic basierende interpretierte Skriptsprache. Sie wurde speziell zum Automatisieren wiederkehrender Aufgaben innerhalb anderer Programme entwickelt, bspw. in MS Excel oder Word. VBA wird in Microsoft Office Umgebungen verwendet, da die integrierte Entwicklungsumgebung standardmäßig in den Office-Programmen enthalten ist.

Visual Basic Script (VBScript oder VBS) ist ebenfalls eine interpretierte Visual-Basic-Variante, die vor allem zum Erstellen von dynamischen Webseiten eingesetzt wird.

1.3 Der Visual Basic-Editor

Die integrierte Entwicklungsumgebung (IDE) stellt die benötigte Infrastruktur und sämtliche Hilfsmittel zur Erstellung von VBA-Programmen zur Verfügung. Diese VBA-Entwicklungsumgebung wird auch Visual Basic-Editor genannt. Der Visual Basic-Editor stellt alle Werkzeuge zur Entwicklung eigener VBA-Programme zur Verfügung.

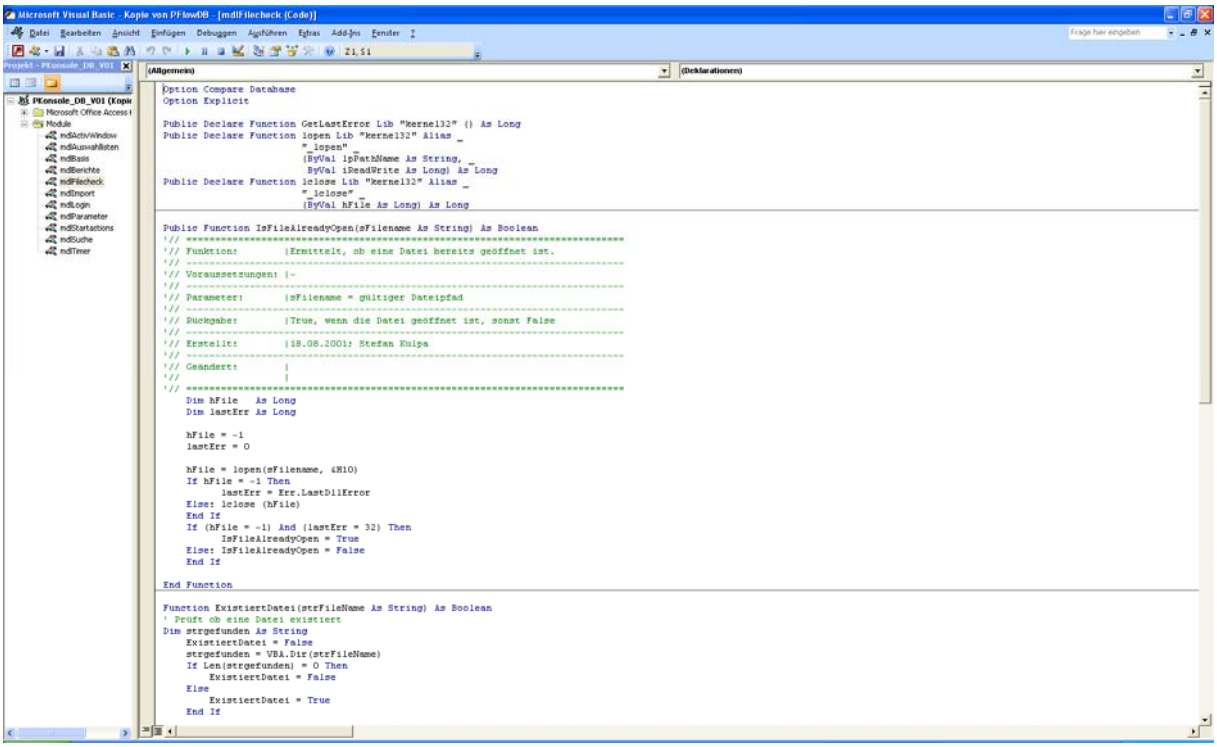


Abbildung 1: Visual Basic-Editor in MS Access 2007

Der Visual Basic-Editor wird aus Access 2007 über die Registerkarte Datenbanktools → Visual Basic oder über die Tastenkombination Alt + F11 aufgerufen.

1.4 Objektorientierte Programmierung mit VBA

Visual Basic ist eine objektorientierte Programmiersprache. Für Programmier-Neulinge kann die Arbeit mit Objekten eine besondere Herausforderung darstellen, jedoch lohnt sich die Mühe.

1.4.1 Was ist ein Objekt?

Objekte bieten dem Programmierer einerseits Zugang zur Funktionalität der zugrunde liegenden VBA-Anwendung bspw. Access oder Excel. Ebenfalls kann auf Objekte aus anderen kompatiblen Anwendungen zugegriffen werden und es ist sogar möglich, eigene Objekte zu erschaffen.

In der Praxis ist ein Objekt einfach ein benanntes Element. Dieses Element verfügt über ganz bestimmte:

- ***Eigenschaften***: Einstellungen, die geändert oder überprüft werden können.
- ***Methoden***: Aktionen, die vom Objekt ausgeführt werden können.
- ***Ereignisse***: Das Objekt reagiert auf bestimmte Ereignisse, indem es automatisch eine vorher definierte Aktion durchführt.

Ein weiteres Merkmal von Objekten besteht darin, dass sie in der Lage sind von anderen Objekten Nachrichten zu empfangen bzw. selbst Nachrichten an andere Objekte zu senden.

1.4.2 Objekte als Komponenten von VBA-Anwendungen

In einer VBA-Anwendung, bspw. Access, stellt jede Tabelle, Abfrage, Formular oder Bericht ein Objekt dar. Das Navigationsfenster in Access visualisiert diesen Umstand sehr gut, jedoch auch das Navigationsfenster ist ein Objekt. Ebenso ist die Multifunktionsleiste, mit allen darin enthaltenen Befehlen, ein Objekt und die einzelnen Befehle sind ebenfalls Objekte.

VBA-Objekte existieren in Hierarchien, in denen bestimmte Objekte andere Arten von Objekten enthalten. Diese Objekt-Hierarchien werden Objektmodelle genannt.

Ein wichtiges Prinzip der Objektmodelle ist die Vererbung. Untergeordnete Objekte erben Eigenschaften ihrer „Eltern“ und können erweiterte Funktionalität erhalten was wiederum der Spezialisierung dient. Ebenso können geerbte Eigenschaften und Funktionen vom untergeordneten Objekt bei Bedarf überschrieben werden.

Beim objektorientierten Programmieren sollten Sie darauf achten, möglichst viel wiederverwertbaren VBA-Code zu erstellen. VBA-Code kann in den einzelnen Objekten selbst existieren, also in den Formularen oder Berichten selbst. Wenn nun bspw. auf einem Formular eine Schaltfläche zur Dateiauswahl existiert, dann sollte der Code der diese Funktionalität bereit stellt nicht im Formular selbst sondern besser in einem eigenständigen Modul gespeichert werden. Die Schaltfläche wird dann lediglich mit dem Aufruf des VBA-Codes und der benötigten Parameter für den Aufruf versehen. Auf diese Weise lässt sich der Code problemlos von weiteren Formularen aus aufrufen und muss nicht in jedem einzelnem Formular gespeichert werden.

2 Das erste Programm

2.1 „Hallo Welt!“-Programm

Das folgende Beispiel soll auf möglichst einfache Weise zeigen, welche Anweisungen oder Bestandteile für ein vollständiges Programm in VBA benötigt werden und somit einen ersten Einblick in die Syntax geben.

Die Aufgabe des Programms besteht darin, den Text "Hallo Welt!" am Bildschirm auszugeben.

Bekanntlich führen mehrere Wege nach Rom und so ist es auch bei den folgenden drei Lösungsvorschlägen für diese Aufgabe, wobei noch zahlreiche weitere Lösungen denkbar wären.

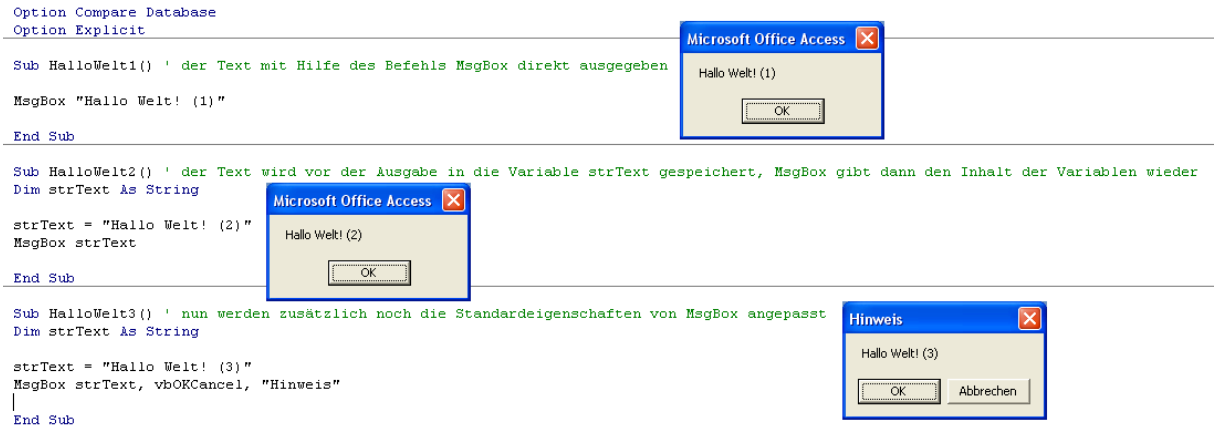


Abbildung 2: „Hallo Welt!“-Programm in drei Variationen mit dem jeweiligen Ergebnis

2.2 Option Explicit

VBA-Code kann in Access in Berichten, Formularen und Modulen untergebracht werden. In diesen Objekten kann durch die Anweisung „Option Explicit“ am Anfang des Programmcodes eine genaue Deklaration aller Variablen in diesem Modul/Formular/Bericht erzwungen werden.

Dadurch werden alle Variablen bereits beim Kompilieren, d.h. noch vor der eigentlichen Ausführung des Codes, auf eine Deklaration überprüft. So können falsch geschriebene Namen bereits vorhandener Variablen oder Verwechslungen im Code bei unklarem Gültigkeitsbereich von Variablen vermieden werden.

Der Visual Basic-Editor kann so eingestellt werden, dass „Option Explicit“ automatisch am Anfang des Codes eingefügt wird. Diese Einstellungsmöglichkeit versteckt sich hinter dem Menü Extras → Optionen → Registerkarte Editor. Hier muss nun vor "Variablendeklaration erforderlich" ein Häkchen gesetzt werden. Nach dem nächsten Neustart des Visual Basic-Editors sind die Änderungen wirksam.

2.3 Code ausführen oder “Wie wird das Programm gestartet?“

Der selbst programmierte Code kann direkt aus dem Visual Basic Editor heraus ausgeführt werden, im Menü Ausführen → Sub/UserForm ausführen. Alternativ kann auch die Taste F5 dazu verwendet werden. Wichtig dabei ist, dass sich der Cursor innerhalb des Codes der auszuführenden Prozedur befindet.

Eine weitere, für den Anfänger womöglich leichter nachzuvollziehende Möglichkeit um den eigenen Programmcode auszuführen ist, eine Schaltfläche auf einem Formular zu erstellen. Der Code wird bei einem Klick auf diese Schaltfläche ausgeführt.

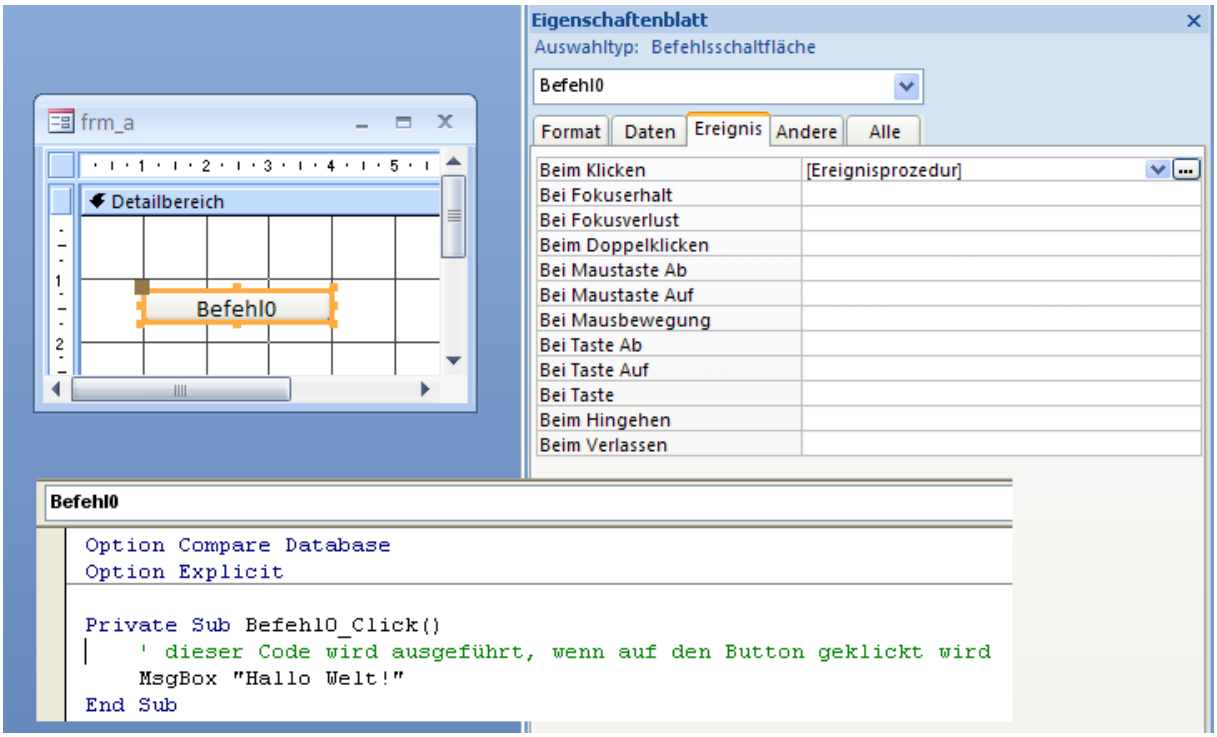


Abbildung 3: Eine Schaltfläche, um den selbst programmierten Code auszuführen

2.4 Bedingungen: If … Then … Else

Eine der elementarsten Fähigkeiten jeder Programmiersprache ist die Prüfung, ob bestimmte Bedingungen erfüllt sind. Abhängig vom Resultat einer solchen Prüfung kann Code ausgeführt oder angehalten werden.

Das folgende Beispiel demonstriert eine einfache Bedingungsprüfung anhand des bereits bekannten „Hallo Welt!“-Programms.

```
Sub Bedingungen() ' Ausgabe eines Textes mit Bedingung, nur wenn boolChecker = TRUE ist
Dim strText As String
Dim boolChecker As Boolean

boolChecker = True ' Die Variable wird auf TRUE gesetzt

If boolChecker Then ' Bei Variablen des Typs Boolean reicht eine solche Prüfung
    strText = "Hallo Welt mit Bedingung!"
    MsgBox strText, vbOKCancel, "Hinweis"
Else
    ' es passiert nichts
End If

End Sub
```

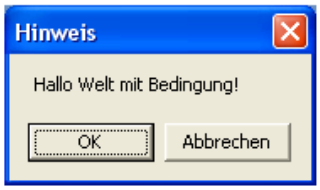


Abbildung 4: Beispiel für eine Bedingungsprüfung

Die Variable boolChecker wird erst deklariert und dann auf den Wert TRUE gesetzt. Die IF-Bedingung prüft danach, welchen Wert die Variable boolChecker besitzt. Bei TRUE wird der bekannte Text ausgegeben, bei FALSE passiert einfach gar nichts.

Dieses Codebeispiel zeigt auch ein paar geläufige Abkürzungen beim Programmieren. Bei der IF-Bedingung wurde bereits die Kurzform verwendet. D.h. statt der Schreibweise „If boolCheker = True Then“ ist in diesem Fall auch ein schlichtes „If boolChecker Then“ erlaubt. Das liegt an der Variablendeklaration von boolChecker: Variablen des Typs Boolean können nur zwei Werte besitzen, TRUE oder FALSE. Der Wert TRUE entspricht dabei der Aussage, dass die Variable existiert, während der Wert FALSE das Gegenteil meint. Insofern prüft die IF-Bedingung die Existenz der Variablen.

Eine weitere Möglichkeit um in diesem Beispiel Schreibarbeit zu sparen wäre es, den ELSE-Teil wegzulassen. Da dort sowieso nichts passiert, ist dieser Teil des Codes eigentlich unnötig.

3 Schleifen

Mit Hilfe von Schleifen werden Programmanweisungen mehrmals wiederholt. Um sich dem Thema zu nähern ist es hilfreich, noch etwas mit dem bereits erstellten „Hallo Welt!“-Programm weiter zu arbeiten.

3.1 Zählergesteuerte Schleifen

Der Text „Hallo Welt!“ soll nun mehrmals untereinander angezeigt werden. Jedes „Hallo Welt!“ soll dabei seine eigene Zeile in dem PopUp-Fenster erhalten. Für den Anfang reichen 3 Zeilen aus.

```
Sub HalloWelt_mehrmals1() ' der Text wird nun mehrmals untereinander ausgegeben, ohne Schleife
Dim strText As String

strText = "Hallo Welt!"
MsgBox strText & vbCrLf & strText & vbCrLf & strText

End Sub
```

```
Sub HalloWelt_mehrmals2() ' der Text wird wieder mehrmals untereinander ausgegeben, dieses Mal mit Schleife
Dim strText1 As String
Dim strText2 As String
Dim i As Integer

strText1 = "Hallo Welt!"
For i = 1 To 3
    strText2 = strText2 & vbCrLf & strText1
Next i
MsgBox strText2

End Sub
```

Abbildung 5: Zwei Variationen mit dem gleichen Ergebnis

Beide Programme aus Abbildung 3 machen exakt das Gleiche und was auch in der Aufgabenstellung gefordert war. Jedoch ist das erste Programm sehr statisch und dadurch in seiner Funktionalität und Erweiterbarkeit stark eingeschränkt. Würde sich bspw. die Aufgabenstellung dahingehend ändern, dass der Text nun 15 Mal angezeigt werden soll, würde dies beim ersten Programm einen erheblichen Programmieraufwand bedeuten. Während im zweiten Programm lediglich eine winzige Änderung nötig wäre, nämlich statt „For i = 1 To 3“ einfach nur „For i = 1 To 15“ zu schreiben.

Damit die Schleife in diesem Beispiel korrekt funktioniert, muss neben einer Zahl-Variablen (i) auch eine weitere Text-Variable (strText2) deklariert werden. Diese Variable dient quasi als Container, in dem der spätere Ausdruck Stück für Stück „zusammengebaut“ wird. Dies geschieht bei den einzelnen Durchläufen der Schleife. Wenn der Wert der Zahl-Variablen (i) das Maximum erreicht hat, ist die Schleife beendet und erst dann wird der Inhalt von strText2 mittels dem Befehl MsgBox ausgegeben.

3.2 Schleifen mit nachgestellter oder vorangestellter Bedingungsprüfung

Schleifen können auch eine unbestimmte Anzahl von Durchläufen besitzen, d.h. die Schleife wiederholt sich so lange, bis eine bestimmte Bedingung erfüllt ist. Die „Do ... Loop“ Schleife ist so eine Schleife, die entweder mit nachgestellter Bedingungsprüfung oder vorangestellter Bedingungsprüfung eingesetzt werden kann.

Die Syntax von „Do ... Loop“ Schleifen sieht wie folgt aus:

mit nachgestellter Bedingungsprüfung:	mit vorangestellter Bedingungsprüfung:
• Do • Anweisungen • Loop [While Bedingung]	• Do [While Bedingung] • Anweisungen • Loop
oder	oder
• Do • Anweisungen • Loop [Until Bedingung]	• Do [Until Bedingung] • Anweisungen • Loop

Bei der Entscheidung mit Until oder While zu arbeiten, sollte darauf geachtet werden, dass sich beide Klauseln gegensätzlich zueinander verhalten.

```
Sub Schleife_DoWhile()
Dim x As Integer
x = 0
    Do While x < 10
        x = x + 1
    Loop
MsgBox "Mit Do While Loop: " & x
End Sub
```

```
Sub Schleife_DoUntil()
Dim x As Integer
x = 0
    Do Until x < 10
        x = x + 1
    Loop
MsgBox "Mit Do Until Loop: " & x
End Sub
```

Abbildung 6: Schleifen mit Do While- und Do Until Loop

Die Do While Schleife wird so oft wiederholt, so lange x kleiner als 10 ist. Sobald x den Wert 10 erreicht (oder überschreitet), wird die Schleife beendet. Da x am Anfang des Codes auf den Wert 0 gesetzt wird, kann die Schleife mehrere Male durchlaufen werden.

Die Do Until Schleife hingegen wird genau so lange ausgeführt bis x kleiner als 10 ist. Sobald x kleiner als 10 ist, wird sie ebenfalls beendet. Somit wird diese Schleife kein einziges Mal durchlaufen, weil die Bedingung für einen Durchlauf von Anfang an nicht erfüllt ist.

4 Import von Excel-Dateien

Access bietet Ihnen viele verschiedene Möglichkeiten, um Daten von Excel nach Access zu bringen. Die einfachste Lösung für diese Aufgabe ist wohl der Befehl Importieren, der jedoch so gut wie keine Einstellungsmöglichkeiten bietet. Die Inhalte der Excel-Tabellen werden dabei lediglich in entsprechenden Access-Tabellen gespeichert. Aber es findet keinerlei Aufteilung der Daten statt, damit diese in Access in einem korrekten relationalen Modell gespeichert werden können.

In den meisten Fällen ist es daher nötig, eigene Import-Routinen in VBA zu programmieren, damit die Daten beim Import bereits korrekt aufgeteilt und in die entsprechenden Tabellen eingetragen werden. Diese Lösung ist natürlich dementsprechend anspruchsvoller und aufwändiger zu realisieren: Die Struktur der Access-Tabellen, in die die Excel-Daten später eingefügt werden, muss erstellt werden und der Code für den Import muss programmiert sowie getestet werden.

4.1 Vorbereitung der Tabellen

Die Übungsdaten bestehen aus insgesamt 6 Excel-Dateien. Die beiden Tabellenblätter der Datei "Teams.xls" können Sie einfach mit Hilfe des Befehls Importieren als jeweils eigene Tabelle in eine neue Datenbank einfügen. Die Tabelle mit den Spielern enthält noch eine Spalte mit Teamnamen. Statt dieser Teamnamen sollen dort jedoch die entsprechenden Schlüsselwerte aus der anderen Tabelle verwendet werden. Diese Daten-Bereinigung kann bspw. mit Hilfe einer Tabellenerstellungsabfrage geschehen. Nachdem die Teamnamen mit den entsprechenden Schlüsselwerten ausgetauscht wurden, kann und muss auch die Beziehung zwischen den beiden Tabellen erstellt werden. Die folgende Abbildung zeigt das gewünschte Ergebnis:

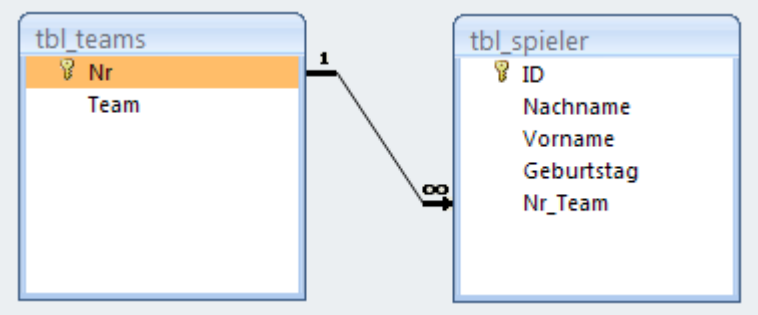


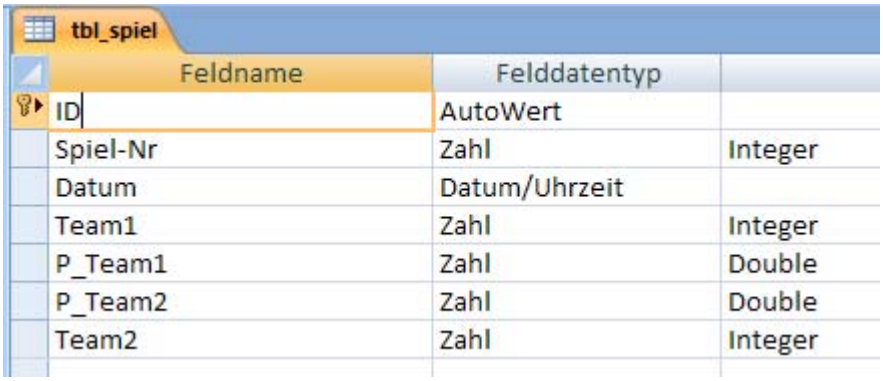
Abbildung 7: Die Beziehung zwischen den ersten 2 Tabellen nach manuellem Daten-Import und -Bereinigung

Falls Sie diese Aufgabe ohne Schritt-für-Schritt-Anleitung nicht lösen können, sollten Sie unbedingt die Access-Grundlagen wiederholen, insbesondere Abfragen!

Nun fehlt noch eine letzte Tabelle, in der sämtliche Spiele der Basketball-Liga gespeichert werden. Die Daten zu den Spielen werden jeweils pro Saison in einer Excel-Datei geliefert. Dieser Daten-Import soll weitgehend automatisiert werden, so dass nur noch die entsprechende Excel-Datei ausgewählt werden muss und die Daten automatisch und korrekt angefügt werden. Voraussetzung dafür ist, dass die Excel-Dateien stets gleich aufgebaut sind, was bei dieser Übung natürlich der Fall ist.

Sie sollten zu Testzwecken die Daten einer Saison auch mit Hilfe des Befehls Importieren in eine neue Tabelle der Datenbank einfügen. Bei den Daten fällt auf, dass wieder die Teamnamen statt der Schlüsselwerte gespeichert sind. Auch das Anfügen von Daten einer weiteren Saison mit derselben Methode gestaltet sich relativ unkomfortabel, da beim Importvorgang stets mehrere Parameter festgelegt werden müssen.

Neben diesen Hinweisen können Sie aus dem Versuch noch einen praktischen Nutzen ziehen und die Struktur der Spiel-Tabelle festlegen. Dabei sollte berücksichtigt werden, dass in der Datenbank statt der Teamnamen die entsprechenden Schlüsselwerte aus tbl_teams gespeichert werden sollen.


tbl_spiel

Feldname	Felddatentyp	
ID	AutoWert	
Spiel-Nr	Zahl	Integer
Datum	Datum/Uhrzeit	
Team1	Zahl	Integer
P_Team1	Zahl	Double
P_Team2	Zahl	Double
Team2	Zahl	Integer

Abbildung 8: Struktur der Spiel-Tabelle

Die neue Tabelle muss natürlich auch noch über Beziehungen mit der Tabelle tbl_teams verknüpft werden. Hierbei ist es nötig, eine weitere Instanz der Tabelle tbl_teams in das Beziehungsfenster zu holen, um die Beziehungen richtig einstellen zu können.

Im Beziehungsfenster kann ein Feld einer Tabelle immer nur mit einer Beziehung versehen werden. Um wie in unserem Beispiel ein Feld einer Tabelle (tbl_teams.Nr) mit zwei Feldern einer anderen Tabelle (tbl_spiel.Team1 und tbl_spiel.Team2) zu verknüpfen, können Sie einfach die Tabelle tbl_teams mittels des Befehls „Tabelle anzeigen“ erneut auswählen.

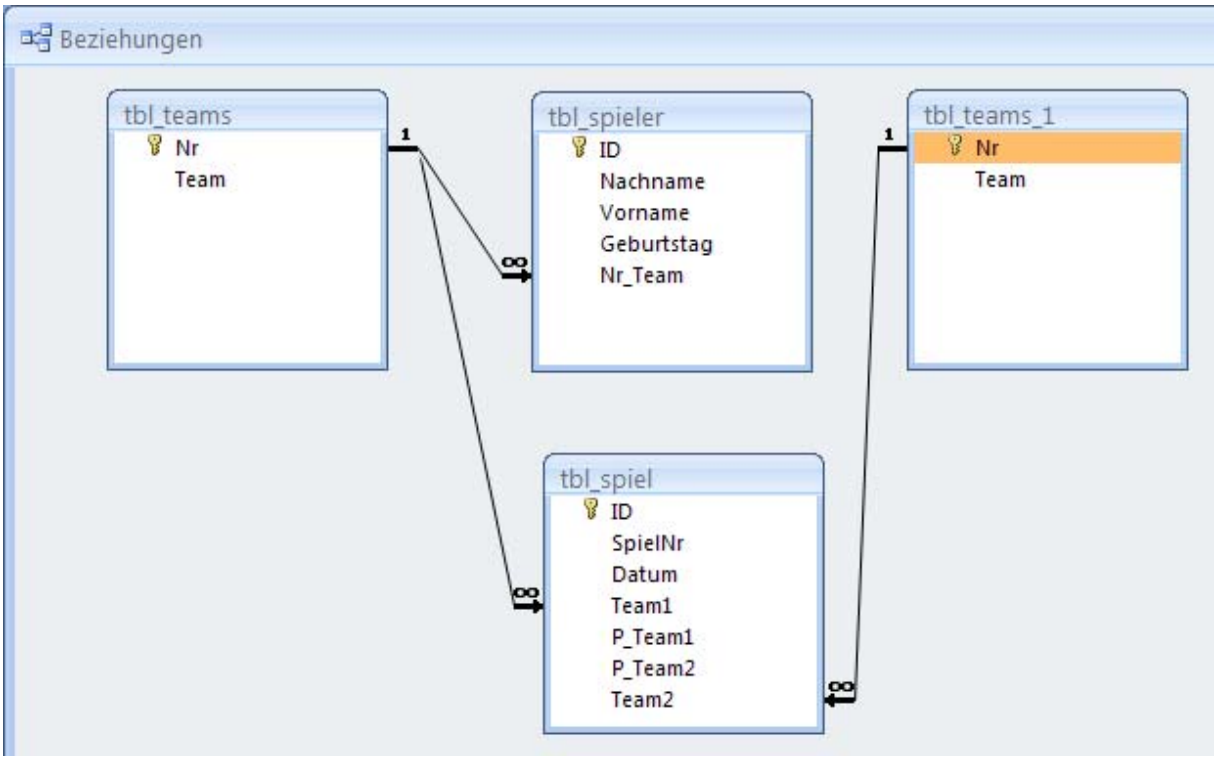


Abbildung 9: Die Beziehungen zwischen allen 3 Tabellen (tbl_teams_1 ist nur eine Instanz im Beziehungsfenster)

4.2 Programmierung der Import-Prozedur

Die automatische Import-Prozedur, mit der später die eben erstellte Tabelle tbl_spiel gefüllt wird, besteht aus mehreren Einzelschritten:

1. Zuerst muss ein Dateiauswahldialog für die User geschaffen werden, damit diese die zu importierende Excel-Datei auswählen können.
2. Danach muss Access die ausgewählte Excel-Datei öffnen und auslesen.
3. Im nächsten Schritt müssen die Teamnamen durch die entsprechenden Schlüsselwerte aus tbl_teams ersetzt werden
4. und erst dann können die Daten in die Tabelle tbl_spiel eingetragen werden.
5. Zuletzt sollte die verwendete Excel-Instanz noch ordnungsgemäß beendet werden.

4.2.1 Dateiauswahldialog

Seit Access 2003 bietet das FileDialog-Objekt von Office eine einfache und stabile Möglichkeit um Dateiauswahldialoge zu erstellen. Die Access-Hilfe (Taste F1) enthält auch bereits ein kleines Beispiel für die Verwendung des FileDialog-Objektes.

Zu beachten ist dabei, dass das FileDialog-Objekt nur dann zur Verfügung steht, wenn mindestens die „Microsoft Office 11.0 Object Library“ im VBA-Editor unter Extras → Verweise eingebunden ist.

Solche Verweise müssen bei jeder Access-Datenbank einmalig vom Entwickler selbst eingestellt werden.

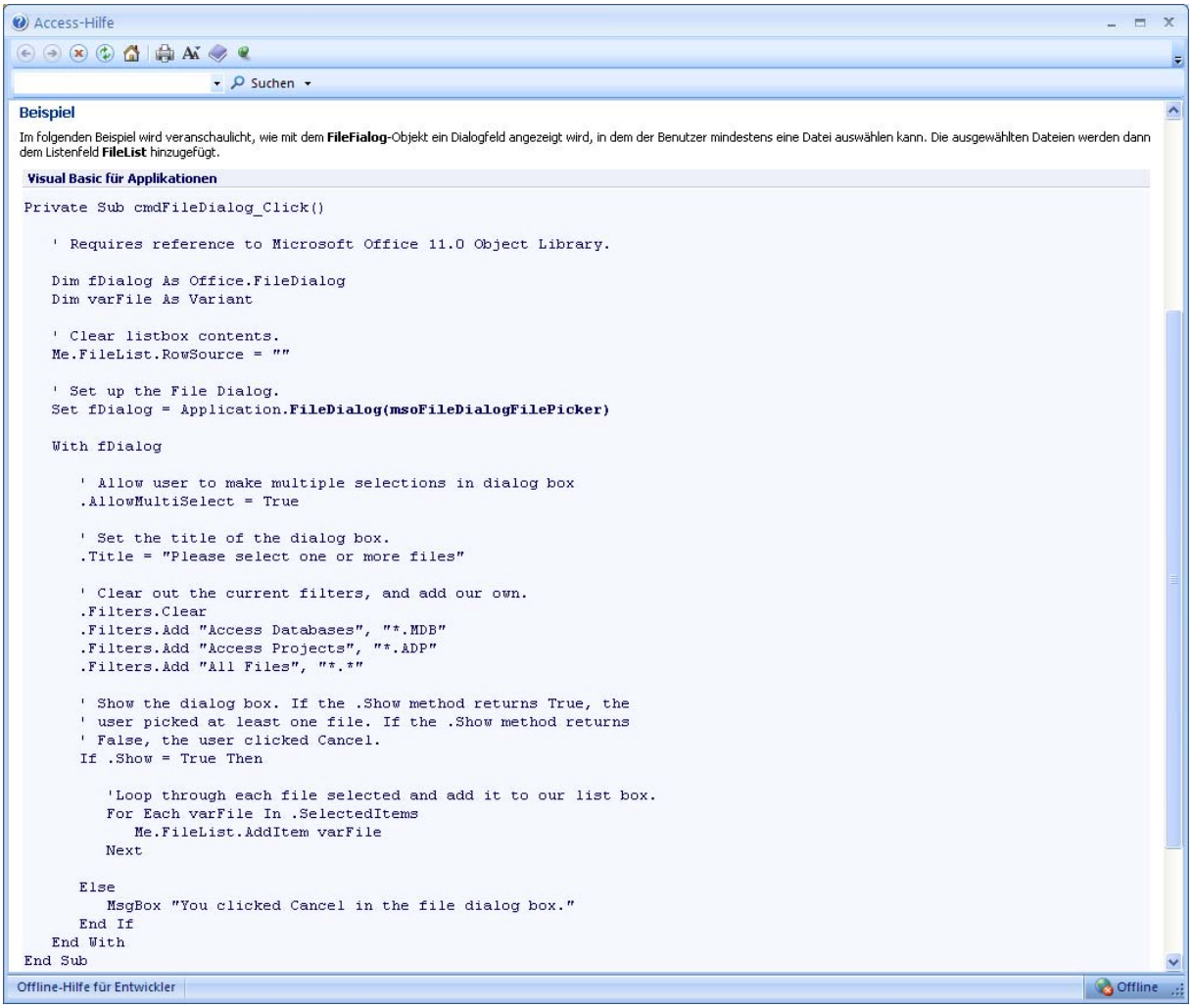


Abbildung 10: Beispiel für die Verwendung des FileDialog-Objektes

Sie können den Code aus der Hilfe kopieren und im VBA-Editor an geeigneter Stelle einfügen, bspw. in einem neuen leeren Formular. Erstellen Sie dazu ein ungebundenes Formular und fügen Sie auf diesem Formular eine Schaltfläche ein. Den Beispiel-Code weisen Sie der Schaltfläche zu dem Ereignis „Beim Klicken“ zu. Nun benötigen Sie noch ein Listenfeld dem Sie den Namen „FileList“ zuweisen. Außerdem muss der Herkunftstyp des Listenfeldes „FileList“ auf „Wertliste“ eingestellt werden.

Wenn Sie alles richtig eingestellt haben, sollte sich nach einem Klick auf die Schaltfläche ein Dateiauswahldialog öffnen. Alle dort ausgewählten Dateien werden dabei in das Listenfeld auf dem Formular eingetragen.

Diesen Beispielcode werden Sie später vielleicht noch geringfügig anpassen müssen, aber für den Moment reicht diese Lösung vollkommen aus und Sie können sich dem Excel-Zugriff widmen.

4.2.2 Zugriff auf Excel: Daten lesen

Die vom Nutzer ausgewählte Excel-Datei soll nun verarbeitet werden, d.h. Access soll prüfen ob es sich um die richtige Datei handelt, die Daten der Datei einlesen und in einer Variablen in Access speichern.

Der scheinbare Umweg über eine Variable zur Zwischenspeicherung der Daten hat viele Vorteile, u.a. verbessert sich dadurch die Performance der Anwendung enorm. Auch kann auf diese Weise die Datenbereinigung (statt Teamnamen die entsprechenden Schlüsselwerte) bereits in der Variablen stattfinden, so dass beim Übertrag der Daten aus der Variablen in die Tabelle tbl_spiele nichts weiter beachtet werden muss und auch keine nachträgliche Korrektur bereits eingetragener Daten mehr stattfinden muss.

Der folgende Code enthält die Anweisungen für den Zugriff auf Excel, um die Daten auszulesen. Sie können den VBA-Code direkt aus diesem Handbuch kopieren und in ein neues Modul einfügen. Der Name des neuen Moduls spielt dabei keine Rolle.

```
Option Compare Database
Option Explicit
'++++++++++++++++++++++++++++++
' Dieses Modul enthält sämtliche Importfunktionen
'++++++++++++++++++++++++++++++
''' Struktur der Excel-Dateien
Type ExcelStruc
   intSpielNr As Integer
   dateDatum As Date
   strTeam1 As String
   intPTeam1 As Integer
   intPTeam2 As Integer
   strTeam2 As String
End Type
Public globalXLSdata() As ExcelStruc
Public globalXLSmax As Integer

Public Sub GetXLSdata(sxlsFile As String, bShowInfo As Boolean)
''' Daten aus Excel werden spaltenweise eingelesen und in einer globalen Variablen abgelegt
Dim i As Integer
Dim iRowS As Integer
Dim iRowL As Integer
Dim sCol As String
Dim vSpielNr As Variant
Dim vDatum As Variant
Dim vTeam1 As Variant
Dim vTeam2 As Variant
Dim vPTeam1 As Variant
Dim vPTeam2 As Variant
' Verweis auf Excel-Bibliothek muss gesetzt sein
Dim xlsApp  As Excel.Application
Dim Blatt As Excel.Worksheet
Dim MsgAntw As Integer
' Konstante: Name des einzulesenden Arbeitsblattes
Const sWSname As String = "Ergebnisse"

   ' Excel vorbereiten
   On Error Resume Next
   Set xlsApp = GetObject(, "Excel.Application")
   If xlsApp Is Nothing Then
      Set xlsApp = CreateObject("Excel.Application")
   End If
   On Error GoTo 0

   ' Exceldatei readonly öffnen
   xlsApp.Workbooks.Open sxlsFile, , True
```

```
    ' Namen aller Tabellenblätter prüfen
    For Each Blatt In xlsApp.ActiveWorkbook.Sheets
        Select Case Blatt.Name
            ' Blatt existiert
            Case sWSname
                ' Erste Zeile wird statisch angegeben
                iRowS = 2
                ' Letzte Zeile auf Tabellenblatt wird dynamisch ermittelt
                iRowL = xlsApp.Worksheets(sWSname).Cells(xlsApp.Rows.Count, 1).End(xlUp).Row
                "' Spalten einlesen
                ' Spalte mit Spiel-Nr. einlesen
                sCol = "A"
                vSpielNr = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Spalte mit Datum einlesen
                sCol = "B"
                vDatum = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Spalte mit Team1 einlesen
                sCol = "C"
                vTeam1 = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Spalte mit PunkteTeam1 einlesen
                sCol = "D"
                vPTeam1 = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Spalte mit PunkteTeam1 einlesen
                sCol = "E"
                vPTeam2 = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Spalte mit Team1 einlesen
                sCol = "F"
                vTeam2 = xlsApp.Worksheets(sWSname).Range(sCol & iRowS & ":" & sCol & iRowL)
                ' Daten einlesen Ende
                '--------------------
                ' Werte in globaler Variable speichern
                globalXLSmax = UBound(vSpielNr)
                ReDim globalXLSdata(1 To globalXLSmax)
                For i = 1 To globalXLSmax
                    globalXLSdata(i).intSpielNr = vSpielNr(i, 1)
                    globalXLSdata(i).dateDatum = vDatum(i, 1)
                    globalXLSdata(i).strTeam1 = vTeam1(i, 1)
                    globalXLSdata(i).intPTeam1 = vPTeam1(i, 1)
                    globalXLSdata(i).intPTeam2 = vPTeam2(i, 1)
                    globalXLSdata(i).strTeam2 = vTeam2(i, 1)
                Next i
        End Select
    Next

    ' Meldung
    If bShowInfo Then
        ' Prüfung ob Daten geladen wurden
        If globalXLSmax > 0 Then
            MsgAntw = MsgBox("Die Daten aus der angegebenen Excel-Datei wurden eingelesen.", vbInformation, "Daten
geladen")
        Else
            MsgAntw = MsgBox("Die angegebene Excel-Datei enthielt keine Daten.", vbInformation, "Keine Daten geladen")
        End If
    End If

    ' Excel reset
    Set xlsApp = Nothing

End Sub
```

Was passiert bislang in diesem Modul?

Zuerst wird die Struktur einer Variablen festgelegt, in der die Excel-Daten zwischengespeichert werden. Diese Struktur wird in einer globalen Variant-Variablen als Array deklariert, wodurch die Speicherung mehrerer Datensätze (=mehrere Zeilen in Excel) möglich ist. Um später die Dimensionen des Arrays, d.h. die Anzahl der Datensätze im Array, bestimmen zu können, wird noch eine Zahl-Variable deklariert.

In der eigentlichen Import-Routine wird zuerst auf das Programm Excel zugegriffen. Falls bereits eine Excel-Instanz existiert, wird auf diese zugegriffen, sonst wird eine neue Instanz gestartet. Ohne weitere Prüfung wird dann direkt die ausgewählte Excel-Datei geöffnet.

An dieser Stelle besteht noch viel Optimierungspotential, da mögliche Fehler überhaupt nicht abgefangen werden. Dies ist beim Zugriff auf das richtige Tabellenblatt bereits besser umgesetzt worden, hier können so gut wie keine unvorhergesehenen Fehler mehr eintreten.

Nun werden mit Hilfe von Excel-Funktionen jeweils die gesamten Daten einer Spalte in das Array einer dafür reservierten Variant-Variablen geschrieben.

Nachdem alle Spalten eingelesen wurden, wird die globale Variable entsprechend dimensioniert und die Daten werden in einer Schleife dort abgelegt.

Die Routine endet mit einer Meldung die Auskunft über den Import gibt.

Die Routine wurde so programmiert, dass der Dateiname der einzulesenden Excel-Datei beim Aufruf mit übergeben werden muss.

Dies erleichtert den nächsten Schritt der darin besteht, den Dateiauswahldialog in die Prozedur mit einzubinden. Zu diesem Zweck können Sie das bereits erstellte Formular zum Testen des Dateiauswahldialogs verwenden. Dort genügt eine zusätzliche Schaltfläche für den Import, welche die Importroutine mit den benötigten Parametern startet. Dies kann bspw. mit folgendem Code geschehen, der bereits für eine Mehrfachauswahl von Dateien geeignet ist:

```
Private Sub cmdImport_Click()
Dim varItem As Variant
    For Each varItem In FileList.ItemsSelected
        GetXLSdata FileList.ItemData(varItem), True
    Next varItem
End Sub
```

Abbildung 11: Aufruf der Importroutine mit Übergabe der Parameter

Auf dem gleichen Formular können Sie zu Kontrollzwecken eine weitere Schaltfläche einfügen, die die Anzahl der eingelesenen Datensätze in der globalen Variable ausgibt. Das ist eine gute Übung um das Verständnis für den bisher verwendeten VBA-Code zu verbessern.

4.2.3 Datenbereinigung: Team-Namen auflösen

Die importierten Daten, die zuvor in der globalen Variablen gespeichert wurden, sollen nun bereinigt werden. Die Team-Namen müssen dabei in die entsprechenden Schlüsselwerte aus tbl_teams umgewandelt werden.

Der folgende Code erledigt diese Aufgabe und kann in das bereits erstellte Modul eingefügt werden.

```
Public Sub CleanXLSdata(bShowInfo As Boolean)
''' Datenbereinigung innerhalb der Variablen
Dim i As Integer
Dim iTeam1 As String
Dim iTeam2 As String
Dim sSQL As String
Dim paraDB As Database
Dim paraTab As Recordset
Dim MsgAntw As Integer
   For i = 1 To globalXLSmax ' Schleife über alle Datensätze in der Variablen
      ' Aktuelle Datenbank auswählen
      Set paraDB = DBEngine.OpenDatabase(CurrentDb.Name)
      ' Schlüsselwerte aus tbl_teams einlesen und in Variable speichern
      Set paraTab = paraDB.OpenRecordset("SELECT Nr, Team FROM tbl_teams WHERE Team = '" &
globalXLSdata(i).strTeam1 & "';")
         If Not paraTab.EOF Then
            iTeam1 = paraTab("Nr")
         End If
      Set paraTab = paraDB.OpenRecordset("SELECT Nr, Team FROM tbl_teams WHERE Team = '" &
globalXLSdata(i).strTeam2 & "';")
         If Not paraTab.EOF Then
            iTeam2 = paraTab("Nr")
         End If
      ' DB und Recordset resetten
      Set paraTab = Nothing
      Set paraDB = Nothing
      ' Teamnamen mit den Schlüsselwerten in globaler Variablen ersetzen
      globalXLSdata(i).strTeam1 = iTeam1
      globalXLSdata(i).strTeam2 = iTeam2
   Next i
   ' Meldung
   If bShowInfo Then
      MsgAntw = MsgBox("Die Daten wurden bereinigt.", vbInformation, "Daten bereinigt")
   End If
End Sub
```

Diese Routine durchläuft in einer Schleife sämtliche in der Variablen vorhandenen Datensätze. Für jeden Datensatz werden die entsprechenden Schlüsselwerte aus der Tabelle tbl_teams ausgelesen und innerhalb der Variablen an Stelle der Teamnamen eingetragen.

4.2.4 Eintragen der Daten in die Access-Tabelle

Da nach der Datenbereinigung aus dem vorigen Kapitel die Daten bereits in der richtigen Form vorliegen, gestaltet sich das nun folgende Eintragen der Daten sehr einfach. Die Daten aus der Variablen werden in einer Schleife durchlaufen und mit Hilfe des SQL-Befehls INSERT in die Tabelle tbl_spiel eingetragen.

```
Public Sub WriteXLSdata(bShowInfo As Boolean)
''' Daten aus Variable in Tabelle übertragen
Dim i As Integer
Dim sSQL As String
Dim MsgAntw As Integer
   For i = 1 To globalXLSmax ' Schleife über alle Datensätze in der Variablen
      ' SQL-String erstellen und Daten schreiben
      sSQL = "INSERT INTO tbl_spiel ( SpielNr, Datum, Team1, P_Team1, P_Team2, Team2 ) VALUES (" &
globalXLSdata(i).intSpielNr & ",'" & globalXLSdata(i).dateDatum & "'," & globalXLSdata(i).strTeam1 & "," &
globalXLSdata(i).intPTeam1 & "," & globalXLSdata(i).intPTeam2 & "," & globalXLSdata(i).strTeam2 & ");"
      DoCmd.SetWarnings False
      DoCmd.RunSQL sSQL
      DoCmd.SetWarnings True
   Next i
   ' Meldung
   If bShowInfo Then
      MsgAntw = MsgBox("Die Daten wurden in die Tabelle tbl_spiel eingetragen.", vbInformation, "Daten eingetragen")
   End If
End Sub
```

Diese Prozedur sollte die Daten korrekt in die Tabelle tbl_spiel eintragen. Damit wäre eine der schwierigsten Aufgaben im Verlauf dieser Übung bereits erledigt. Nun gilt es möglichst flexible Suchmöglichkeiten und entsprechend dynamische Ergebnisanzeigen für die importierten Daten zu erstellen.

4.2.5 Excel-Instanz beenden

Nach dem Import existiert bislang weiterhin eine versteckte Excel-Instanz. Das können Sie selbst prüfen, indem Sie nach dem Import die Datenbank schließen und einen Blick in den Windows Task-Manager werfen. Diese Excel-Instanz besteht weiter, weil bislang nirgends im Code der Befehl zum Beenden von Excel gegeben wurde. Dies könnte bspw. mit einer zusätzlichen Zeile in der Subroutine GetXLSdata geschehen. Am Ende des Codes muss es dann lauten:

```
' Excel reset und Instanz beenden
xlsApp.Quit
Set xlsApp = Nothing
```

Da jedoch der bisherige Code der Importroutine bereits für den Import mehrerer ausgewählter Excel-Dateien vorgesehen ist, sollte auch die Excel-Instanz erst nach dem Einlesen der letzten Datei geschlossen werden. Sie benötigen also eine eigene Routine zum Beenden der Excel-Instanz, die nach dem Import der letzten Excel-Datei ausgeführt werden soll.

```
Public Sub CloseXLSapp(bShowInfo As Boolean)
'" Excel-Instanz beenden
' Verweis auf Excel-Bibliothek muss gesetzt sein
Dim xlsApp  As Excel.Application
Dim MsgAntw As Integer
   ' Excel-Instanz suchen
   On Error Resume Next
   Set xlsApp = GetObject(, "Excel.Application")
   If xlsApp Is Nothing Then
      ' keine Excel-Instanz vorhanden
      ' Meldung
      If bShowInfo Then
         MsgAntw = MsgBox("Es wurde keine Excel-Instanz gefunden.", vbInformation, "Excel-Instanz beenden")
      End If
      ' Ende
      Exit Sub
   End If
   On Error GoTo 0
   ' Excel schließen und resetten
   xlsApp.Quit
   Set xlsApp = Nothing
   ' Meldung
   If bShowInfo Then
      MsgAntw = MsgBox("Die Excel-Instanz wurde beendet.", vbInformation, "Excel-Instanz beenden")
   End If
End Sub
```

Nun da alle Routinen programmiert wurden, müssen diese lediglich in der richtigen Reihenfolge gestartet werden damit ein korrekter Import stattfindet. Falls Ihnen hierzu gar nichts einfallen will, werfen Sie doch mal einen Blick auf das Titelblatt dieses Handbuchs ☺

5 Anspruchsvolle Formulare

Bei der Gestaltung von Formularen werden Sie VBA-Code in der Regel am häufigsten verwenden. Formulare dienen der Darstellung von Daten und Benutzungsoberflächen – da bietet sich die rege Verwendung von VBA natürlich an.

In Kapitel 4 haben Sie bereits das Datenmodell umgesetzt, eine automatische Import-Routine programmiert und nach den Testläufen sollten in der Tabelle tbl_spiel auch Daten vorhanden sein, mit denen gearbeitet werden kann. Die nächste Aufgabe lautet nun, eine Benutzungsoberfläche zu erstellen. Dies wird mit Hilfe des Übersichts-Managers realisiert werden.

5.1 Der Übersichts-Manager

Wenn der Übersichts-Manager in einem Access 2007 Projekt eingesetzt wird, dann enthält das Formular „Übersicht“ keinerlei VBA-Code mehr. Die gesamte Funktionalität des Übersichts-Managers wird seit Programmversion 2007 nur noch mit Hilfe von automatisch erstellten Makros umgesetzt. Dies erschwert die Anpassung der Funktionalitäten und das Einbringen von selbstgeschriebenem Code.

Sie können sich jedoch einen Übersichts-Manager in einer Access 2003 Datenbank erstellen und das Formular „Übersicht“ von dort in die gewünschte Access 2007 Datenbank importieren. Damit haben Sie wieder Zugriff auf den reinen VBA-Code und können diesen leichter anpassen.

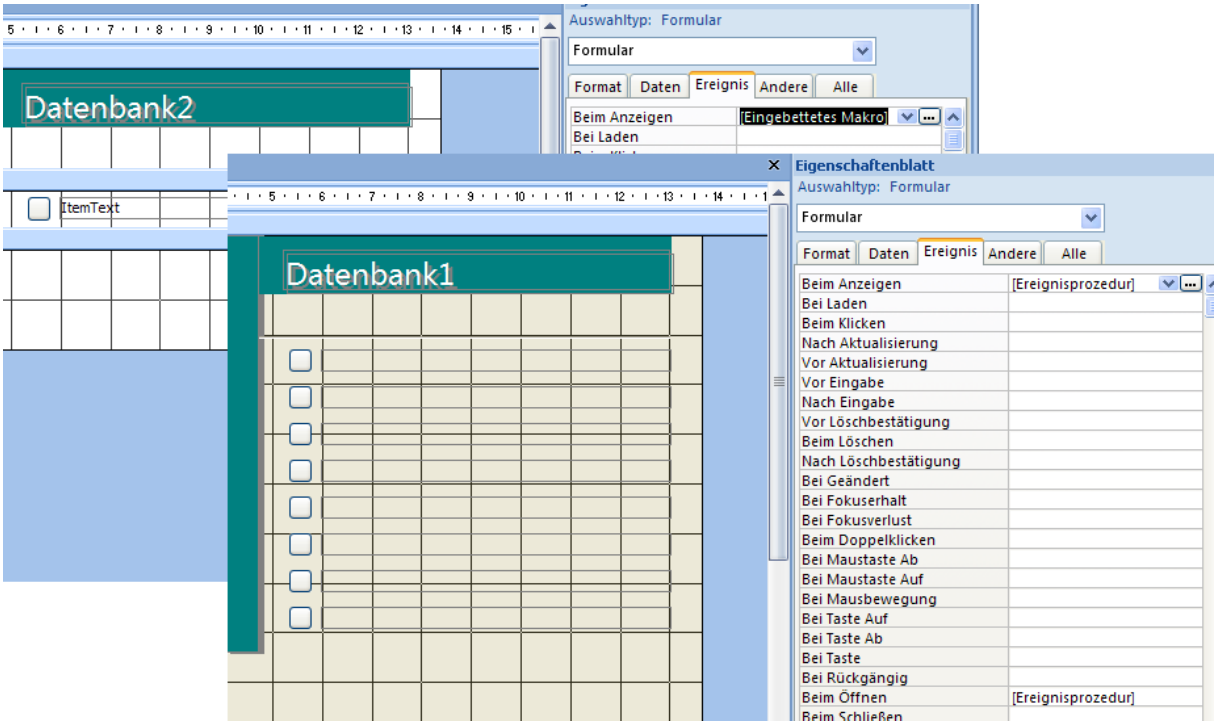


Abbildung 12: Verschiedene Versionen des Übersichts-Managers in Access 2007 und davor

Wenn Sie die alte Version des Übersichts-Managers verwenden, ist es relativ einfach die Übersicht um weitere Funktionalitäten zu erweitern. Sie könnten bspw. in der Tabelle „Switchboard Items“ ein weiteres Feld zur Angabe von Farbwerten hinzufügen. Im VBA-Code des Übersichts-Managers könnte dann für jedes angezeigte Element der Farbwert ausgelesen werden um damit bspw. die Hintergrundfarbe des Formulars oder der Überschrift zu verändern.

```
Private Sub Form_Current()
Dim strUsername As String
Dim strRGBValue As String
Dim fldRGBValue() As String
Dim lngRGBValue As Long
Dim dbsPDatbank As Database
Dim tblCurrentUsers As Recordset

' Update the caption and fill in the list of options.
    Me.Caption = Nz(Me![ItemText], "")
    FillOptions

' Einfärben der Kopfzeile je nach Menü-Level
    strRGBValue = Me![MenueColor]
    fldRGBValue = Split(strRGBValue, ",")
    lngRGBValue = RGB(fldRGBValue(0), fldRGBValue(1), fldRGBValue(2))
    Me!HorizontalHeaderBox.BackColor = lngRGBValue
    Me!HorizontalHeaderBox.BorderColor = lngRGBValue
```

Abbildung 13: Beispiel für die Anpassung des Übersichts-Managers, vgl. Abb. 12

Dieser Code wird dem Formular „Übersicht“ bei dem Ereignis „Beim Anzeigen“ zugewiesen.

Damit der Code ordnungsgemäß funktioniert, müssen Sie das Feld „MenueColor“ in der Tabelle „Switchboard Items“ ergänzen. In das neue Feld können Sie RGB-Farbwerte eintragen, als Trennzeichen sollten Sie das Komma verwenden (vgl. Abb. 12)

Ebenso müssen Sie darauf achten, dass im Formular „Übersicht“ ein Steuerelement mit dem Namen „HorizontalHeaderBox“ existiert.

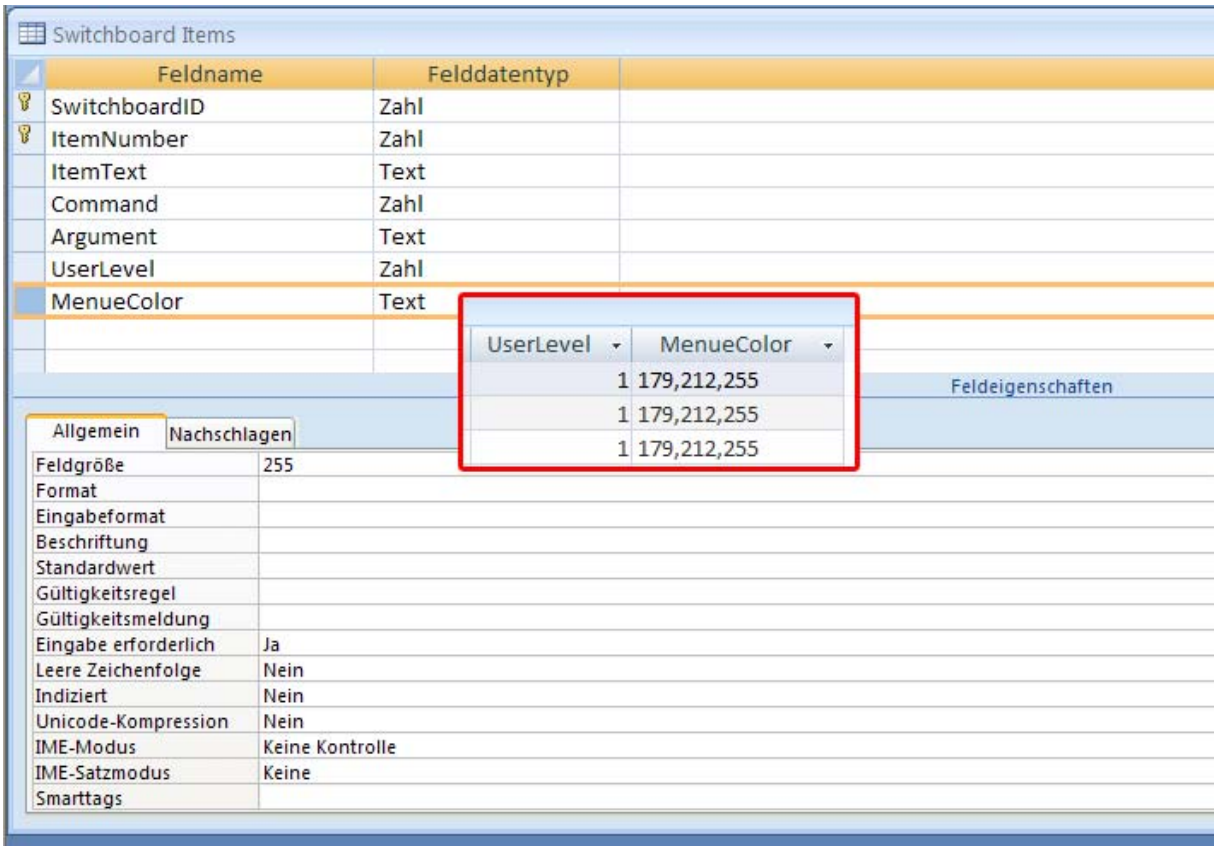


Abbildung 14: Beispiel für die Anpassung des Übersichts-Managers, vgl. Abb. 12

Es sind noch viele weitere Anpassungen denkbar, bspw. ein eigenes Login-System mit dem Übersichts-Manager zu verbinden.

5.2 Dynamische Berichte und Formulare

Im Gegensatz zu einer statischen Lösung, bei der für jedes Ergebnis einer Auswahl oder Suche ein eigenes Ergebnisformular erstellt wird, bietet der Einsatz von VBA viele dynamische Möglichkeiten, um verschiedenste Ergebnisse in nur einem Ergebnisformular oder –bericht anzuzeigen. Dadurch können Sie komplizierte Anforderungen mit nur wenigen Access-Objekten umsetzen.

5.2.1 Formulare zur Suche über mehrere Felder mit Auswahl der Felder

Dynamisch bedeutet in diesem Zusammenhang, dass sich sowohl das Suchformular als auch das Ergebnisformular an die Eigenschaften der Datensatzquelle und die Such-Parameter des Users anpassen. Dies werden Sie nun beispielhaft an einer Suche für die Tabelle tbl_spieler üben.

Erstellen Sie ein neues leeres Formular in der Entwurfsansicht und versehen Sie dieses Formular mit einem Kombinationsfeld, einem Listenfeld und einem Textfeld. Die Bezeichnungsfelder können Sie bei allen Feldern außer dem Kombinationsfeld löschen.

Das Kombinationsfeld wird die Feldnamen der Tabelle tbl_spieler enthalten, die jedoch dynamisch per VBA ausgelesen werden. Im Listenfeld werden nur die Einträge „=“ und „LIKE“ erscheinen und diese Liste wird sich auch nicht mehr ändern, daher können diese Werte beim Erstellen des Listenfeldes direkt von Hand eingetragen werden. Das Textfeld bleibt ungebunden. Die drei Felder sollten wie in der Abbildung horizontal angeordnet werden. Darunter können Sie bereits eine Schaltfläche setzen, der Code dafür folgt später.

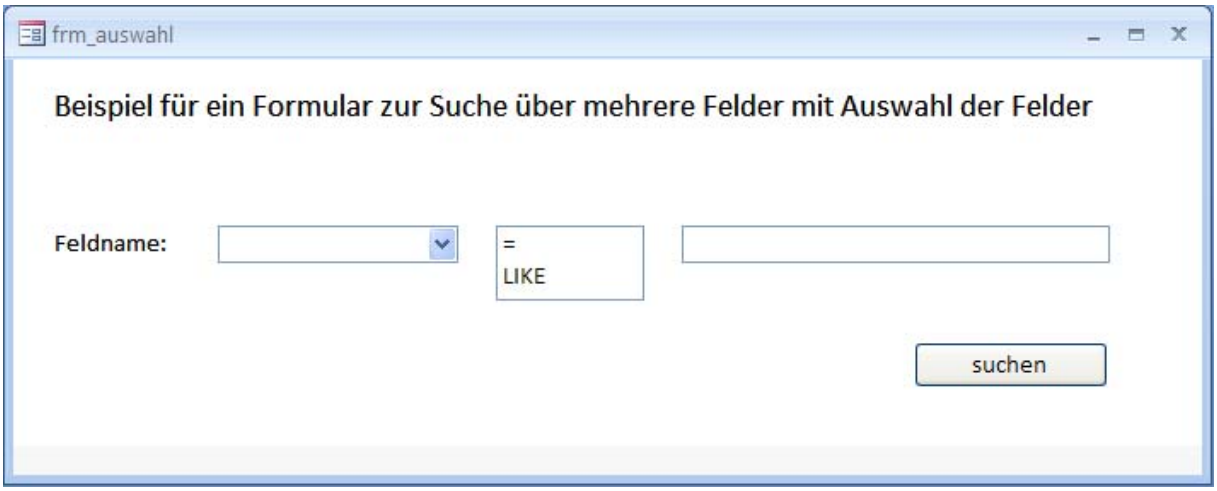


Abbildung 15: Beispiel für das Layout des dynamischen Such-Formulars

Dem Kombinationsfeld sollten Sie den Namen „feldauswahl“ geben. Außerdem muss in den Eigenschaften des Kombinationsfelds der Herkunftstyp auf Werteliste gesetzt werden, sonst lassen sich dem Kombinationsfeld keine Einträge per VBA hinzufügen.

Der Name des Kombinationsfeldes gibt dem nun folgenden VBA-Code ein Ziel für die automatisch ermittelten Feldnamen von tbl_spieler an. Die Feldnamen lassen sich sehr einfach mit Hilfe der Informationen in der Fields-Auflistung ermitteln.

```
Private Sub Form_Open(Cancel As Integer) ' Ereignis Beim Öffnen des Formulars
Dim i As Integer
Dim rs As Recordset

' tbl_spieler wird in in Recordset eingelesen
Set rs = CurrentDb.OpenRecordset("tbl_spieler", dbOpenSnapshot)

' Alles Spalten des Recordsets werden durchlaufen
For i = 0 To rs.Fields.Count - 1
  ' Jeder Feldname wird der Liste im Kombinationsfeld feldauswahl hinzugefügt
  Me.feldauswahl.AddItem Item:=rs.Fields(i).Name
Next

End Sub
```

Abbildung 16: Beispiel für das Layout des dynamischen Such-Formulars

Dieser VBA-Code kopiert als Erstes die Tabelle tbl_spieler in ein Recordset. Dieses Recordset wird in einer For … Next-Schleife durchlaufen und die Feldnamen einer nach dem anderen der Werteliste des Kombinationsfeldes hinzugefügt.

NOCH IN BEARBEITUNG !!!

5.3 Verweise auf Haupt- und Unterformulare

Beim Setzen von Verweisen auf Haupt- und Unterformulare ist die richtige Syntax unabdingbar. Syntaktisch fehlerhafte Verweise im VBA-Code werden von Access nicht erkannt und mit bösen Fehlermeldungen bestraft.

Die folgende Tabelle enthält einige häufige Fälle möglicher Verweise. Eine umfassendere Liste finden Sie in der Linksammlung im Anhang dieses Handbuchs.

	Sie befinden sich auf dem Hauptformular	1. Unterformular
Verweis auf eine Eigenschaft des Formulars, bspw. RecordSource:		
Im Hauptformular	Me.RecordSource	Me.Parent.RecordSource
Im 1. Unterformular	Me!Subform1.Form.RecordSource	Me.RecordSource
Im 2. Unterformular	Me!Subform1.Form!Subform2.Form.RecordSource	Me!Subform2.Form.RecordSource
Verweis auf ein Steuerelement:		
Im Hauptformular	Me!ControlName	Me.Parent!ControlName
Im 1. Unterformular	Me!Subform1.Form!ControlName	Me!ControlName
Im 2. Unterformular	Me!Subform1.Form!Subform2.Form!ControlName	Me!Subform2.Form!ControlName
Verweis auf die Eigenschaften eines Steuerelements, bspw.Enabled:		
Im Hauptformular	Me!ControlName.Enabled	Me.Parent!ControlName.Enabled
Im 1. Unterformular	Me!Subform1.Form!ControlName.Enabled	Me!ControlName.Enabled
Im 2. Unterformular	Me!Subform1.Form!Subform2.Form!ControlName.Enabled	Me!Subform2.Form!ControlName.Enabled
Verweis auf die Eigenschaften eines Steuerelements im Unterformular, bspw.SourceObject:		
Im Hauptformular	NICHT VERFÜGBAR	NICHT VERFÜGBAR
Im 1. Unterformular	Me!Subform1.SourceObject	NICHT VERFÜGBAR
Im 2. Unterformular	Me!Subform1.Form!Subform2.SourceObject	Me!Subform2.SourceObject

Quelle: Dev Ashish, http://www.mvps.org/access/forms/frm0031.htm

6 Export von Access-Daten

Der Export von Access-Daten gestaltet sich über den Befehl Exportieren ebenso einfach wie das Importieren. Jedes Objekt aus Access kann mit nur einem Mausklick exportiert werden, aber es fehlen wieder viele Einstellungsmöglichkeiten gegenüber der Umsetzung eines Exports mit Hilfe von VBA.

In Kapitel 5 haben Sie ein Formular zur Suche über mehrere Felder mit Auswahl der Felder erstellt. Es wäre sicher sinnvoll, auf dem Ergebnisformular dieser Suche eine Schaltfläche für den Export nach Excel zu platzieren. Für diese Schaltfläche wird natürlich wieder etwas VBA-Code benötigt.

6.1 Export mit DoCmd.OutputTo-Methode

Mit der OutputTo-Methode können die Daten eines angegebenen Datenbankobjekts (Datenblatt, Formular, Bericht, Modul oder Datenzugriffsseite) in mehrere Ausgabeformate ausgegeben werden. In der Access-Hilfe findet sich dazu ein Beispiel, das für die eigenen Bedürfnisse entsprechend angepasst werden kann.

Beispiel

Dieses Beispiel gibt die Tabelle **Employees** im Rich Text-Format (RTF) aus und öffnet die Datei in Microsoft Word für Windows.

Visual Basic für Applikationen

```
DoCmd.OutputTo acOutputTable, "Employees", _
    acFormatRTF, "Employee.rtf", True
```

Abbildung 17: Beispiel für DoCmd.OutputTo aus der Access-Hilfe

Dieser VBA-Code könnte in abgewandelter Form bspw. einer Schaltfläche bei dem Ereignis „Beim Klicken“ zugewiesen werden, um Daten nach Excel zu exportieren. Voraussetzung ist, dass das zu exportierende Objekt in der Datenbank vorhanden ist.

Sie sollten diese DoCmd.OutputTo-Methode nun an verschiedenen Stellen bzw. für verschiedene Objekte Ihrer Datenbank ausprobieren. Beim Testen wird Ihnen wahrscheinlich das eine oder andere Problem bei Verwendung dieser Methode auffallen. Dies soll Sie für die Notwendigkeit der nun folgenden Exportprozedur sensibilisieren, mit Hilfe derer der Export beliebiger Daten problemlos möglich ist.

6.2 Export beliebiger Recordsets

Die bislang behandelten Methoden zum Export von Daten haben den Nachteil, dass lediglich die in Access vorhandenen Objekte exportiert werden können. Dynamische Formulare zur Ergebnisanzeige, deren Datenquelle zur Laufzeit per VBA ermittelt wird, können damit nicht oder nur sehr umständlich exportiert werden. Wenn bspw. per VBA beim Aufruf eines Formulars Filter gesetzt werden, kann die DoCmd.OutputTo-Methode nicht mehr verwendet werden, weil hierbei die Filter fehlen würden.

Solche und ähnliche Probleme lassen sich über die sehr flexible Methode CopyFromRecordset in VBA lösen. Das Recordset eines Formulars oder Berichts enthält exakt nur die Daten, die auch am Bildschirm angezeigt werden. Es ist damit also ohne weiteres möglich, gefilterte Ergebnisse korrekt zu exportieren.

Der folgende Code ist für den Export beliebiger Recordsets gedacht. Das Recordset wird beim Aufruf an die VBA-Prozedur übergeben.

```
Public Sub XLSExportRS(rs As Recordset)
' Export aller Daten des Unterformulars nach Excel mit CopyFromRecordset-Methode
' David Niegisch, 25.02.2010
Dim gsMsgAntw As Integer
Dim iCols As Integer
Dim ws As Object ' falls Verweise auf die Excel-Bibl. gesetzt sind, ohne late binding: Excel.Worksheet
Dim wb As Object ' ohne late binding: Excel.Workbook
Dim ExcelApp As Object ' ohne late binding: Excel.Application

    ' MoveFirst sorgt dafür, dass die gleichen Daten auch mehrmals hintereinander exportiert werden können
    rs.MoveFirst

    ' Nur ausführen wenn Daten vorhanden sind
    If rs.EOF Then
        gsMsgAntw = MsgBox("Es sind keine Ergebnisse für einen Export vorhanden!", vbInformation, "Export nicht moeglich")
    Else
        ' Neue Excel Datei, Arbeitsmappe und Worksheet erstellen
        Set ExcelApp = CreateObject("Excel.Application")
        Set wb = ExcelApp.Workbooks.Add
        Set ws = ExcelApp.Worksheets.Add
        ' Daten nach Excel kopieren
        ' Feld- bzw. Spaltennamen
        For iCols = 0 To rs.Fields.Count - 1
            ws.Cells(1, iCols + 1).Value = rs.Fields(iCols).Name
        Next
        ' Daten
        ws.Range(ws.Cells(1, 1), _
            ws.Cells(1, rs.Fields.Count)).Font.Bold = True
        ws.Range("A2").CopyFromRecordset rs
        ' Spaltenbreite anpassen
        ws.Columns.AutoFit
        ' Excel sichtbar machen und alles resetten
        ExcelApp.Visible = True
        Set ws = Nothing
        Set wb = Nothing
        Set ExcelApp = Nothing
    End If

End Sub
```

Abbildung 18: VBA-Code zum Export beliebiger Recordsets (Link zum Code im Anhang)

Der Aufruf kann bspw. über einen Button stattfinden, der auf einem Hauptformular liegt. Das Hauptformular enthält ein Unterformular, in dem die Daten angezeigt werden.

```
' Aufruf mit Übergabe der Daten des Hauptformulars
Private Sub export_Click()
    XLSExportRS Me.RecordsetClone
End Sub

' Aufruf mit Übergabe der Daten des Unterformulars
Private Sub export2_Click()
    XLSExportRS Me!Unterformular.Form.RecordsetClone
End Sub
```

Abbildung 19: Aufruf der Exportfunktion

Eine kopierfähige Version dieses Codes finden Sie, wenn Sie dem entsprechenden Link im Anhang dieses Handbuchs folgen.

7 Abschluss der Übung und Ausblick

Die letzte Aufgabe lautet nun, das gesammelte Wissen in der Übungs-Datenbank sinnvoll unterzubringen.

Viele der Objekte, die Sie im Verlauf der Übung erstellt haben, bedürfen noch einer Überarbeitung hinsichtlich ihrer Funktionalität und des Layouts.

Auf Grund der Menge an VBA-Code in diesem Projekt, bedarf es nun auch eines sinnvollen Konzepts zur Fehlervermeidung bzw. zum Abfangen von Fehlern, damit der User seine Arbeit auch nach einem Fehler fortsetzen kann. Ein sinnvoll aufgebautes und objektorientiertes Programm lässt Sie diese Aufgabe sehr viel leichter bewältigen als wirrer und zusammenhangloser Spaghetti-Code.

Versuchen Sie, sich in die späteren User der Datenbank hineinzuversetzen. Besser noch, fragen Sie die User nach ihren Empfindungen und Problemen bei der Benutzung Ihres Programms, sofern Sie dazu die Möglichkeit haben.

Programme, die nur der Entwickler selbst getestet hat, werden in der Regel als Alpha-Versionen bezeichnet. Die Beta-Version eines Programms wird dann für einen öffentlichen oder nicht-öffentlichen Benutzertest erstellt. Nach der Bereinigung aller im Beta-Test gefundenen Fehler, darf sich das Programm erst als Version 1.0 bezeichnen. Die Realität in der Softwareentwicklung kann davon jedoch auch schon mal gehörig abweichen.

Anhang: Linksammlung

VBA-Tutorials von Manuela und Stefan Kulpa:

http://www.kulpa-online.com/tutorials.html

Access-FAQ von Karl Donaubauer:

http://www.donkarl.com/FAQ/FAQStart.htm

Verweise auf Haupt- und Unterformulare von Dev Ashish:

http://www.mvps.org/access/forms/frm0031.htm

MS-Office-Forum:

http://www.ms-office-forum.net/forum/

VBA-Code aus Kapitel 6.2, Export beliebiger Recordsets

http://www.ms-office-forum.net/forum/showthread.php?t=260414

Link zu Beispiel mit dynamischen Formularen aus Kapitel 5.2.1, noch unentschieden

www.xyz.de

Artikel „Wie (un)sicher ist Access?“ von Karl Donaubauer

http://www.access-im-unternehmen.de/index1.php?id=300&BeitragID=371

Dieses Handbuch sowie sämtliche Beispiel-Daten, die für die Übungen benötigt werden, finden Sie online unter:

http://yahg.net/kurse/access/vba/