Kapitel 8

Dateien/Verzeichnisse

In diesem Kapitel:

Verzeichnispfade anlegen
Windows- und Office-Dateidialoge einbinden
Dateioperationen durchführen
Dateisystem bestimmen
Datenträger analysieren und formatieren

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

R8.1 Einen Verzeichnispfad anlegen

Aufgabe

Sie kennen zwar die *MkDir*-Funktion, aber Installationsprogramme oder Exportfunktionen nehmen vom Anwender häufig Zielpfade entgegen, die gleich mehrere Unterverzeichnisse enthalten können. Mit der *MkDir*-Funktion aber ist das gleichzeitige Anlegen von mehr als einem Unterverzeichnis nicht möglich.

Lösung

Abhilfe schafft eine »selbst gebastelte« Funktion, in welcher die *MkDir*-Anweisung im Zusammenspiel mit VBA-Zeichenkettenfunktionen wie *Left$*, *Right$*, *Mid$*, *Len* und *InStr* den Verzeichnispfad erstellt.

Oberfläche

Für den Test genügen uns ein *Textfeld* und eine *Befehlsschaltfläche*, die wir auf den Detailbereich eines neuen Formulars setzen.

Quelltext

```
Option Explicit

Sub Mk_Dir(bez1 As String)
On Error Resume Next
Dim verz As String, bez As String
  bez = bez1
  verz = Left$(bez, 3)
  bez = Right$(bez, Len(bez) – 3)
  If Right$(bez, 1) <> "\" Then bez = bez & "\"
  verz = verz & Mid$(bez, 1, InStr(bez, "\") – 1)
  bez = Right$(bez, Len(bez) – InStr(bez, "\"))
  While Right$(verz, 1) <> "\"
    MkDir verz
    If bez <> "" Then
        verz = verz & "\" + Mid$(bez, 1, InStr(bez, "\") – 1)
    Else
        verz = verz & "\"
    End If
    bez = Right$(bez, Len(bez) – InStr(bez, "\"))
  Wend
End Sub
```

Der Aufruf zum Erzeugen eines Verzeichnisses mit beliebig vielen Unterverzeichnissen:

```
Private Sub Befehl0_Click()
  Mk_Dir (Text0.Value)
End Sub
```

Test

Tragen Sie in das Textfeld den gewünschten Verzeichnispfad ein und klicken Sie die Schaltfläche:

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte
© 2011 O'Reilly Verlag GmbH & Co. KG; ISBN 978-3-86645-098-1

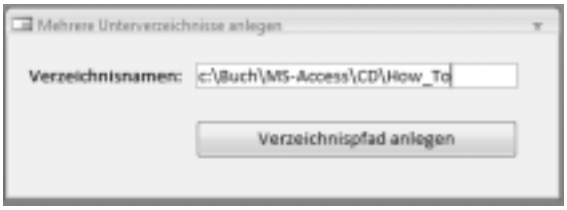

Abbildung 8.1 Laufzeitansicht

Überzeugen Sie sich davon, dass der Verzeichnispfad tatsächlich erstellt wurde.

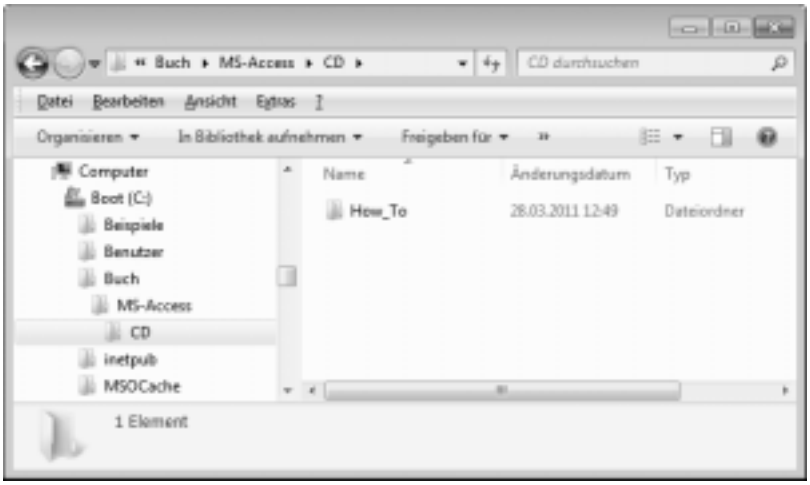

Abbildung 8.2 Der neu angelegte Verzeichnispfad

HINWEIS Wenn aber das Verzeichnis bereits existiert? Kein Problem, die Funktion ignoriert den auftretenden Laufzeitfehler (*On Error Resume Next*).

R8.2 Einen Verzeichnisdialog aufrufen

Aufgabe

Sie wollen einen Verzeichnisdialog aufrufen, ohne dazu erst umständlich ein ActiveX-Steuerelement oder die Office-Library einbinden zu müssen.

Lösung

Die gestellte Aufgabe lässt sich unter Verwendung mehrerer API-Funktionsaufrufe lösen.

Oberfläche

Wir brauchen lediglich ein Formular mit einer *Befehlsschaltfläche* zum Aufruf des Verzeichnisdialogs und einem *Textfeld*, in welchem das ausgewählte Verzeichnis angezeigt wird.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Quelltext

```
Option Explicit
```

Basis des Verzeichnisdialogs ist die folgende Datenstruktur:

```
Private Type BrowseInfo
     hwndOwner As Long
     pIDLRoot As Long
     pszDisplayName As Long
     lpszTitle As Long
     ulFlags As Long
     lpfnCallback As Long
     lParam As Long
     iImage As Long
End Type
```

Die notwendigen API-Funktionen und -Konstanten:

```
Private Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal hMem As Long)

Private Declare Function lstrcat Lib "kernel32" Alias "lstrcatA" _
                                    (ByVal lpString1 As String, ByVal lpString2 As String) As Long

Private Declare Function SHBrowseForFolder Lib "shell32" (lpbi As BrowseInfo) As Long

Private Declare Function SHGetPathFromIDList Lib "shell32" _
                                         (ByVal pidList As Long, ByVal lpBuffer As String) As Long

Private Const BIF_RETURNONLYFSDIRS = 1
Private Const MAX_PATH = 256
```

Die folgende Funktion kapselt obige API-Aufrufe und erzeugt einen Verzeichnisdialog mit Rückgabe eines Pfads:

```
Public Function BrowseForFolder(Beschriftung As String) As String
Dim pidl As Long
Dim path As String
Dim bi As BrowseInfo
  bi.hwndOwner = Screen.ActiveForm.hWnd
  bi.lpszTitle = lstrcat(Beschriftung, "")
  bi.ulFlags = BIF_RETURNONLYFSDIRS
  pidl = SHBrowseForFolder(bi)
  If pidl Then
     path = String(MAX_PATH, 0)
     SHGetPathFromIDList pidl, path
     CoTaskMemFree pidl
     path = Left$(path, InStr(path, vbNullChar) - 1)
  End If
  BrowseForFolder = path
End Function
```

Der Quellcode zum Aufruf der Funktion beschränkt sich auf diesen Ereignishandler:

```
Private Sub Befehl0_Click()
  Text1.Value = BrowseForFolder("Wählen Sie ein Verzeichnis aus ...")
End Sub
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Test

Nach dem Programmstart rufen Sie den Verzeichnisdialog auf und klicken auf die *OK*-Schaltfläche:

Abbildung 8.3 Laufzeitansicht

R8.3 Die Anzeige langer Verzeichnispfade verkürzen

Aufgabe

Im Zeitalter langer Dateinamen kommt auf Sie neben den vielen Vorteilen auch ein Problem zu: Wie soll man einen langen Pfadnamen in der begrenzten Länge eines Formulars anzeigen, ohne den Formularaufbau zu zerstören oder ohne die wesentlichen Informationen wegzulassen?

Lösung

Am sinnvollsten scheint eine Verkürzung derartiger Pfadangaben, indem man die Verzeichnisinformationen durch Platzhalterpunkte (»...«) ersetzt und stattdessen den Dateinamen anzeigt (z.B.: *C:\...\info.dat* anstatt *C:\Programme\Access\Test\Daten\info.dat*).

Oberfläche

Lediglich zwei *Textfelde*r (für die Eingabe eines langen Dateinamens und für die Ausgabe der verkürzten Version) sind notwendig.

Quelltext

```
Option Explicit
```

Der folgenden Funktion übergeben Sie einfach den Pfad sowie die maximal zulässige Länge in Zeichen. Zurückgegeben wird der auf die maximal zulässige Zeichenanzahl verkürzte Pfad:

```
Function MaxDirLen(ByVal pfad As String, ByVal maxSize As Integer) As String
Dim i As Integer, pLen As Long
     pLen = Len(pfad)
     MaxDirLen = pfad
     If Len(Pfad) <= maxSize Then Exit Function
     For i = pLen – maxSize + 6 To pLen
          If Mid$(pfad, i, 1) = "\" Then Exit For
     Next
     MaxDirLen = Left$(pfad, 3) + "..." + Right$(pfad, pLen – (i – 1))
End Function
```

Der Aufruf:

```
Private Sub Text0_Change()
    Text1.Value = MaxDirLen(Text0.Text, 20)
End Sub
```

Test

Sobald Sie den Dateinamen im oberen Textfeld ändern, erscheint im unteren Textfeld die verkürzte Version.

Abbildung 8.4 Das Testformular zur Laufzeit

R8.4 Das Datenbankverzeichnis ermitteln

Aufgabe

Sie brauchen einen Verweis auf das Verzeichnis, in welchem sich die aktuelle Datenbank befindet. Die *CurDir()*-Funktion ist zur Ermittlung des Pfades leider nicht geeignet.

Ein konkreter Anwendungsfall wäre das Zuweisen der *Picture*-Eigenschaft von Steuerelementen, da Sie im Quelltext nicht den kompletten Pfadnamen für die zur Laufzeit zu ladenden Bitmaps angeben möchten. Spätestens dann, wenn die Applikation auf einen anderen PC portiert werden soll, dürfte es aber Probleme geben, da dort die Pfadangaben nicht mehr stimmen.

Lösung

Als Lösung bietet sich die Eigenschaft *Path* des zum *Application*-Objekt gehörigen *CurrentProject*-Objekts an.

Oberfläche

Setzen Sie ein *Bezeichnungsfeld* und eine *Befehlsschaltfläche* auf den Detailbereich eines neuen Formulars.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte
© 2011 O'Reilly Verlag GmbH & Co. KG; ISBN 978-3-86645-098-1

Quelltext

```
Option Explicit
```

Um den Aufruf zu vereinfachen, können Sie zum Beispiel folgende Funktion programmieren:

```
Private Function aktVerz() As String                ' bestimmt Datenbank-Verzeichnis
  aktVerz = Application.CurrentProject.Path
End Function
```

Wir verwenden diese Funktion, um das Datenbankverzeichnis anzuzeigen und um beim Laden des Formulars der *Picture*-Eigenschaft der Befehlsschaltfläche eine kleine Grafik zuzuweisen, die sich im Unterverzeichnis *\Bilder* des Datenbankverzeichnisses befindet:

```
Private Sub Form_Load()
 Bezeichnungsfeld0.Caption = Bezeichnungsfeld0.Caption & " " & aktVerz
 Befehl1.Picture = aktVerz & "\Bilder\Bild1.bmp"
End Sub

Private Sub Befehl1_Click()
 DoCmd.Close
End Sub
```

Test

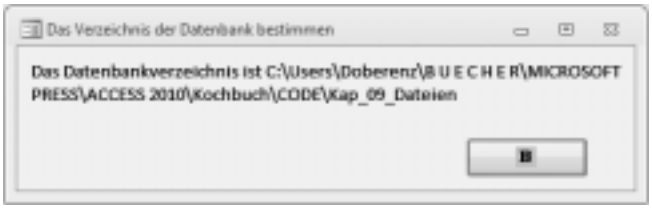

Abbildung 8.5 Ansicht nach dem Laden des Formulars

R8.5 Den Pfad zur zugehörigen EXE-Datei ermitteln

Aufgabe

In welchem Verzeichnis befindet sich das mit einer *.txt*-Datei verknüpfte Programm *Notepad.exe*? Sie möchten die Antwort auf diese oder eine ähnliche Frage gern per VBA-Code ermitteln.

Lösung

Die API-Funktion *FindExecutable* liefert zu jedem Dateityp den dazugehörigen Programmnamen inklusive Pfad.

Der Funktion wird der Name und der komplette Pfad zu einer Datei übergeben. Falls das damit verknüpfte Programm installiert ist, liefert die Funktion einen Wert > 32 zurück und gleichzeitig den Pfad zur entsprechenden EXE-Datei.

Oberfläche

Auf ein Formular setzen Sie zwei *Textfeld*er, eine *Befehlsschaltfläche* und ein für die Ergebnisanzeige besonders hervorgehobenes *Bezeichnungsfeld* (siehe Laufzeitansicht).

Quellcode

```
Option Explicit
```

Zunächst die Deklaration der API-Funktion *FindExecutable* nebst einigen benötigten Konstanten:

```
Private Declare Function FindExecutableA Lib "shell32.dll" _
               (ByVal lpFile As String, ByVal lpDirectory As String, ByVal lpResult As String) As Long

Const ERROR_FILE_NOT_FOUND = 2&
Const ERROR_BAD_FORMAT = 11&
Const ERROR_PATH_NOT_FOUND = 3&
```

Aufruf und Auswertung der Funktion:

```
Private Sub Befehl0_Click()
  Dim pfadExe As String * 255
  Dim ret As Long
  Dim nameDatei As String
  Dim pfadDatei As String
  nameDatei = Text0.Value
  pfadDatei = Text1.Value

  ret = FindExecutableA(nameDatei, pfadDatei, pfadExe)
  Select Case ret
    Case Is > 32
      Bezeichnungsfeld0.Caption = pfadExe
    Case 31
      Bezeichnungsfeld0.Caption = "Keine Zuordnung zu diesem Dateityp!"
    Case 0
      Bezeichnungsfeld0.Caption = "System überlastet!"
    Case ERROR_FILE_NOT_FOUND
      Bezeichnungsfeld0.Caption = "Datei nicht gefunden!"
    Case ERROR_PATH_NOT_FOUND
      Bezeichnungsfeld0.Caption = "Pfad nicht gefunden!"
    Case ERROR_BAD_FORMAT
      Bezeichnungsfeld0.Caption = "Ungültiges EXE-Format!"
  End Select
End Sub
```

Damit Sie sich nicht erst die Mühe machen müssen, Namen und Pfad einer vorhandenen Datei einzugeben, erfolgt die Initialisierung mit einer Datei *Test.txt*, die Sie vorher in den Pfad der aktuellen Datenbank kopiert haben:

```
Private Sub Form_Load()
   Text0.Value = "Test.txt"
   Text1.Value = CurrentProject.path
End Sub
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Test

Nach Öffnen des Formulars und Klick auf die *Start*-Schaltfläche wird der Pfad zu *Notepad.exe* angezeigt.

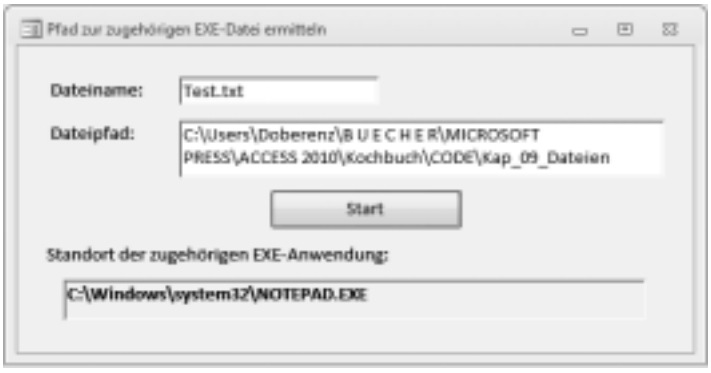

Abbildung 8.6 Laufzeitansicht

Bemerkungen

- Da wir eine universelle Lösung angestrebt haben, funktioniert das Programm auch für andere mit einer bestimmten Anwendung verknüpften Dateitypen. Geben Sie zum Beispiel den Namen und das Verzeichnis einer gültigen Word-Datei ein, so wird als Ergebnis der Pfad zur *Winword.exe* angezeigt.
- Es soll nicht verschwiegen werden, dass die die gezeigte Lösung zwar auch für eine *.mdb*-Datenbank funktioniert, nicht aber für eine *.accdb*-Datenbank. Dies ist aber nicht weiter schlimm, da die *SysCmd*-Funktion eine deutlich einfachere Lösung für den Fall bietet, dass Sie sich nur auf Access-Datenbankdateien beziehen wollen. Die folgende Anweisung zeigt ebenfalls das Verzeichnis zur *msaccess.exe* an:

```
BezeichnungsfeldO.Caption = SysCmd(acSysCmdAccessDir)
```

R8.6 Prüfen, ob eine Datei existiert

Aufgabe

Leider verfügt VBA über keine Funktion, mit der Sie das Vorhandensein einer Datei testen können. Lasst uns Abhilfe schaffen!

Lösung

Wir schreiben eine eigene Funktion *FileExist*, in welcher wir die *Dir*-Standardfunktion verwenden. Letztere gibt eine Zeichenfolge zurück, die dem ersten Dateinamen entspricht, der mit dem übergebenen Argument übereinstimmt.

Oberfläche

Ein *Textfeld*, zwei *Bezeichnungsfeld*er und zwei *Befehlsschaltfläche*n genügen zum Testen der Funktion.

Quellcode

```
Option Explicit

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

Unser Testaufruf:

```
Private Sub Button0_Click()
   If FileExist(Text0.Value) Then
     Bezeichnungsfeld0.Caption = "Die Datei ist vorhanden!"
                               Else
     Bezeichnungsfeld0.Caption = "Die Datei ist nicht vorhanden!"
   End If
End Sub
```

Test

Ohne viele Worte:

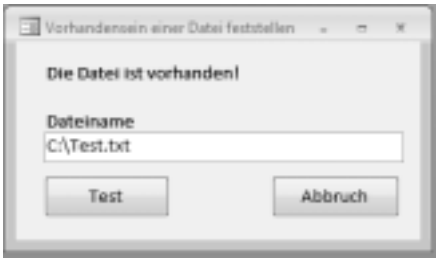

Abbildung 8.7 Funktionstest

Bemerkung

Ein praktisch sinnvolles Beispiel für die Verwendung unserer *FileExist*-Funktion:

```
Dim db As Database
If Not FileExist("C:\TEST.ACCDB") Then
    MsgBox "Datenbank nicht gefunden!", 16, "Problem"
    Exit Sub
Else
    Set db = OpenDatabase("C:\TEST.ACCDB", False, False)
End If
```

HINWEIS Allerdings lässt sich einem zurückgelieferten Wert *True* nicht entnehmen, ob die Datenbank von Ihnen auch exklusiv geöffnet werden kann. Es wird lediglich festgestellt, dass die Datei physisch vorhanden ist.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

R8.7 Die Windows-Dateidialoge einbinden

Aufgabe

Sie wollen die von Windows bereitgestellten Dateidialoge verwenden, verfügen aber nicht über die benötigten ActiveX-Controls bzw. arbeiten mit einer älteren Access-Version.

Lösung

Wir programmieren eine Klasse *Dialog,* welche die entsprechenden API-Aufrufe kapselt. Enthalten sind die jedem Windows-Anwender bekannten Standard-Dialogfelder zum Laden und Speichern von Dateien.

Oberfläche

Zum Testen der Klasse verwenden wir ein Formular mit zwei ungebundenen *Textfeld*ern sowie drei *Befehlsschaltfläche*n (*Datei Laden*, *Datei Sichern*, *Beenden*).

Abbildung 8.8 Startformular (Entwurfsansicht)

Quelltext (Klassendefinition)

Dass der Aufruf der API-Funktionen nicht ganz unkompliziert ist, dürfte aus dem Umfang des folgenden Listings ersichtlich sein. Je universeller eine Funktion ist, umso umfangreicher sind auch die erforderlichen Vorbereitungen und leider auch die Fehlermöglichkeiten.

Der erste Schritt ist das Anlegen eines neuen Klassenmoduls (Hauptregisterkarte *Erstellen*, Befehlsgruppe *Andere*). Übernehmen Sie danach folgende Deklarationen in den Deklarationsabschnitt der Klasse:

```
Option Explicit

Private Const MAX_PATH = 260
```

HINWEIS Achten Sie bei der Übernahme der folgenden Struktur peinlichst genau auf die verwendeten Datentypen. Diese weichen zum Teil von den Definitionen in der Datei *Win32api.txt* ab.

```
Private Type OPENFILENAME
        lStructSize As Long
        hwndOwner As Long
        hInstance As Long
        lpstrFilter As String
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
        lpstrCustomFilter As Long
        nMaxCustFilter As Long
        nFilterIndex As Long
        lpstrFile As String
        nMaxFile As Long
        lpstrFileTitle As Long
        nMaxFileTitle As Long
        lpstrInitialDir As String
        lpstrTitle As Long
        Flags As Long
        nFileOffset As Integer
        nFileExtension As Integer
        lpstrDefExt As Long
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As Long
End Type
```

Die Einbindung der beiden API-Funktionen:

```
Private Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" _
                                            (pOpenfilename As OPENFILENAME) As Long

Private Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA" _
                                            (pOpenfilename As OPENFILENAME) As Long
```

Die Konstanten für die Funktionsaufrufe:

```
Private Const OFN_READONLY = &H1
Private Const OFN_OVERWRITEPROMPT = &H2
Private Const OFN_HIDEREADONLY = &H4
Private Const OFN_NOCHANGEDIR = &H8
Private Const OFN_SHOWHELP = &H10
Private Const OFN_ENABLEHOOK = &H20
Private Const OFN_ENABLETEMPLATE = &H40
Private Const OFN_ENABLETEMPLATEHANDLE = &H80
Private Const OFN_NOVALIDATE = &H100
Private Const OFN_ALLOWMULTISELECT = &H200
Private Const OFN_EXTENSIONDIFFERENT = &H400
Private Const OFN_PATHMUSTEXIST = &H800
Private Const OFN_FILEMUSTEXIST = &H1000
Private Const OFN_CREATEPROMPT = &H2000
Private Const OFN_SHAREAWARE = &H4000
Private Const OFN_NOREADONLYRETURN = &H8000
Private Const OFN_NOTESTFILECREATE = &H10000
Private Const OFN_NONETWORKBUTTON = &H20000
Private Const OFN_NOLONGNAMES = &H40000
Private Const OFN_EXPLORER = &H80000
Private Const OFN_NODEREFERENCELINKS = &H100000
Private Const OFN_LONGNAMES = &H200000
```

Über die Methode *OpenFile* kann der zugehörige Dateidialog angezeigt werden. Um die Einbindung ins Programm so einfach wie möglich zu gestalten, sind beide Übergabewerte optional, als Rückgabewert kann der Dateiname ausgewertet werden.

```
Public Function OpenFile(Optional filterString, Optional verzeichnis) As String
Dim pOpenfilename As OPENFILENAME
```

```
Dim fileName As String, initDir As String, filter As String
Dim l As Long
```

Da die Funktion *GetOpenFileName* nullterminierte Strings bzw. speziell initialisierte String-Puffer erwartet, sind einige Vorarbeiten nötig:

```
    fileName = Chr(0) & Space(MAX_PATH)
```

Falls die optionalen Parameter nicht übergeben wurden, werden Defaultwerte eingesetzt:

```
    If IsMissing(filterString) Then
        filter = "Alle Dateien (*.*)|*.*"
    Else
        filter = filterString
    End If
    If IsMissing(verzeichnis) Then
        initDir = CurDir
    Else
        initDir = verzeichnis
    End If
```

Der Filter-String muss konvertiert werden, zwischen den einzelnen Filterbezeichnern steht eine Null:

```
   If Right(filter, 1) <> "|" Then filter = filter & "|"
    For l = 1 To Len(filter)
        If Mid(filter, l, 1) = "|" Then Mid(filter, l, 1) = Chr(0)
    Next l
    filter = filter & Chr(0) & Chr(0)
```

Füllen der Struktur, an dieser Stelle könnten Sie noch Anpassungen vornehmen (*Flags*):

```
   With pOpenfilename
     .lStructSize = Len(pOpenfilename)
     .hwndOwner = Screen.ActiveForm.hwnd
     .lpstrFile = fileName
     .nMaxFile = MAX_PATH
     .lpstrFilter = filter
     .nFilterIndex = 1
     .lpstrInitialDir = initDir
     .Flags = OFN_EXPLORER Or OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST
    End With
```

Aufruf der Funktion und nachfolgende Auswertung der Rückgabewerte:

```
    If GetOpenFileName(pOpenfilename) <> 0 Then
        fileName = pOpenfilename.lpstrFile
        OpenFile = Left(filename, InStr(filename, Chr(0)) – 1)
    Else
        OpenFile = ""
    End If
End Function
```

Die Funktion *SaveFile* hat fast den gleichen internen Aufbau, optional kann jedoch ein Dateiname angegeben werden.

```
Public Function SaveFile(Optional dateiname, Optional filterstring, Optional verzeichnis) As String
Dim pOpenfilename As OPENFILENAME
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
Dim filename As String, initdir As String, filter As String
Dim l As Long

    If IsMissing(dateiname) Then
        filename = Chr(0) & Space(MAX_PATH)
    Else
        filename = dateiname & Chr(0) & Space(MAX_PATH)
    End If
    If IsMissing(filterstring) Then
        filter = "Alle Dateien (*.*)|*.*"
    Else
        filter = filterstring
    End If
    If IsMissing(verzeichnis) Then
        initdir = CurDir
    Else
        initdir = verzeichnis
    End If
    If Right(filter, 1) <> "|" Then filter = filter & "|"
    For l = 1 To Len(filter)
        If Mid$(filter, l, 1) = "|" Then Mid(filter, l, 1) = Chr(0)
    Next l
    filter = filter & Chr(0) & Chr(0)

    With pOpenfilename
        .lStructSize = Len(pOpenfilename)
        .hwndOwner = Screen.ActiveForm.hwnd
        .lpstrFile = filename
        .nMaxFile = MAX_PATH
        .lpstrFilter = filter
        .nFilterIndex = 1
        .lpstrInitialDir = initdir
        .Flags = OFN_EXPLORER
    End With

    If GetSaveFileName(pOpenfilename) <> 0 Then
        filename = pOpenfilename.lpstrFile
        SaveFile = Left(filename, InStr(filename, Chr(0)) - 1)
    Else
        SaveFile = ""
    End If
End Function
```

Quelltext (Formular)

Die Verwendung der Klasse *Dialog* ist denkbar einfach.

```
Option Explicit
```

Zunächst der *Öffnen*-Dialog:

```
Private Sub Befehl1_Click()
 Text1.Value = Dialog.OpenFile("Access (*.mdb)|*.mdb;*.mda", "c:\")
```

oder

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
 Text1.Value = Dialog.OpenFile("Access (*.mdb)|*.mdb;*.mda")
```

oder

```
 Text1.Value = Dialog.OpenFile()
End Sub
```

Die Varianten beim *Speichern unter*-Dialog:

```
Private Sub Befehl2_Click()
 Text2.Value = Dialog.SaveFile("c:\test.dat", "*.dat|*.dat", "c:\")
 ' oder
 Text2.Value = Dialog.SaveFile("c:\test.dat", "*.dat|*.dat")
 ' oder
 Text2.Value = Dialog.SaveFile("c:\test.dat")
 ' oder
 Text2.Value = Dialog.SaveFile()
End Sub
```

Test

Starten Sie das Programm und überprüfen Sie Funktion und Aussehen der Dateidialoge:

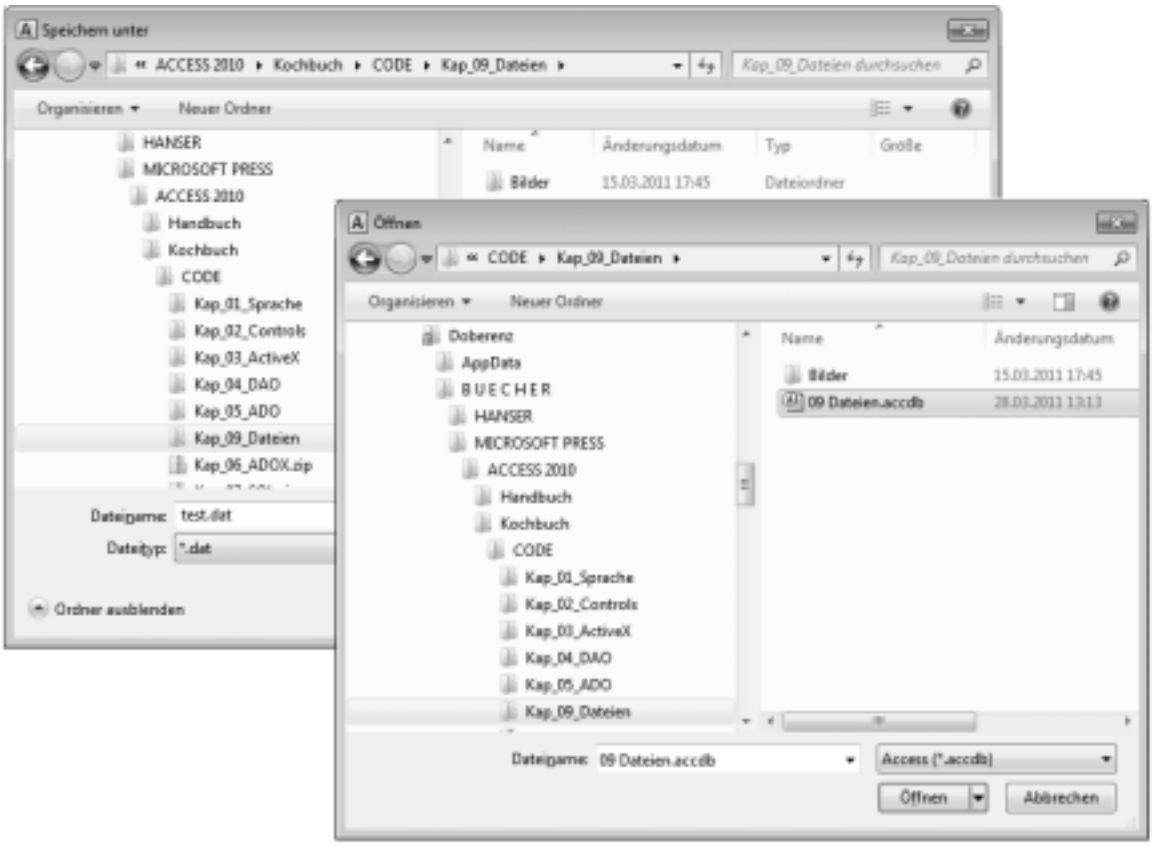

Abbildung 8.9 Testen der Dateidialoge

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

R8.8 Die Office-Dateidialoge verwenden

Aufgabe

Sie benötigen einen Dateidialog, verfügen aber weder über ein passendes ActiveX-Steuerelement (*CommonDialog*), noch wollen Sie den doch recht erheblichen Aufwand mit Windows-API-Funktionen (siehe Vorgängerrezept R8.7) betreiben.

Lösung

Benutzen Sie die in der Objektbibliothek von Microsoft Office zur Verfügung gestellten Dateidialoge, um Dateien auszuwählen, zu öffnen und zu speichern!

Oberfläche

Auf ein Formular setzen Sie zwei *Befehlsschaltfläche*n (*Datei öffnen*, *Datei speichern*).

Quelltext

HINWEIS Binden Sie über das Menü *Extras/Verweise...* die *Microsoft Office 14.0 Object Library* ein!

```
Option Explicit
```

Per Klick auf die erste Befehlsschaltfläche wird der Dialog zum Öffnen einer Datei aufgerufen:

```
Private Sub Befehl0_Click()
Dim dlg As FileDialog
Dim si As Variant
```

Objekt erzeugen:

```
   Set dlg = Application.FileDialog(msoFileDialogFilePicker)
   With dlg
```

Mehrfachauswahl zulässig und Beschriftung der Schaltfläche ändern:

```
      .AllowMultiSelect = True
      .ButtonName = "Backup"
```

Dateifilter definieren und den ersten aktivieren:

```
      .Filters.Add "Alle", "*.*"
      .Filters.Add "Texte", "*.txt; *.rtf; *.doc"
      .Filters.Add "Grafiken", "*.gif; *.jpg; *.jpeg"
      .FilterIndex = 0
```

Den Ausgangspfad einstellen:

```
      .InitialFileName = Application.CurrentProject.Path
      .InitialView = msoFileDialogViewDetails
      .Title = "Datei-Backup"
   End With
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Auswerten:

```
   If dlg.Show Then
      For Each si In dlg.SelectedItems
         MsgBox si
      Next
   End If
End Sub
```

Ähnlich funktioniert der Dialog zum Speichern einer Datei:

```
Private Sub Befehl1_Click()
Dim dlg As FileDialog
   Set dlg = Application.FileDialog(msoFileDialogFilePicker)
   With dlg
      .Title = "Speichern als"
      .ButtonName = "Speichern"
      .AllowMultiSelect = False
      .InitialFileName = "c:\"
      If .Show Then MsgBox dlg.SelectedItems(1)
   End With
End Sub
```

Test

Starten Sie das Programm und testen Sie die Dialoge auf »Herz und Nieren«.

HINWEIS Halten Sie die Shift-Taste gedrückt, um mehrere Dateien auszuwählen!

Bemerkungen

- Durch Übergabe einer anderen Konstanten (z.B. *msoFileDialogFolderPicker*) an die Funktion *FileDialog* können Sie auch einen anderen Dialogtyp erzeugen
- Über zahlreiche andere Eigenschaften (z.B. *InitialView*, womit die Ansicht des Dialogfelds eingestellt wird) informieren Sie sich am besten in der Online-Hilfe
- Leider zeigen nicht alle Eigenschaften unter Access 2010 bzw. Windows 7/Vista die erhoffte Wirkung, sodass Sie um einiges Experimentieren nicht herumkommen werden

R8.9 Dateien rekursiv suchen

Aufgabe

Sie wollen in einem Verzeichnis nach Dateien suchen, für die Sie ein bestimmtes Suchkriterium vorgeben. Zum Beispiel sollen aus dem Verzeichnis *C:\Programme\Microsoft Office\Office14* alle **.exe*-Dateien angezeigt werden.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Lösung

Für die Dateisuche bieten einige API-Funktionen eine schnelle und relativ einfach zu realisierende Möglichkeit, die auch rekursive Aufrufe unterstützt.

Oberfläche

Erforderlich sind drei *Textfeld*er (zwei kleinere und ein großes) sowie eine *Befehlsschaltfläche*. Die Verwendung von *Bezeichnungsfeld*ern bleibt Ihnen überlassen (siehe Laufzeitabbildung am Schluss des Beispiels).

Quelltext

Wie üblich, müssen zunächst einige API-Deklarationen eingefügt werden:

```
Private Const MAX_PATH = 259

Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
End Type

Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * MAX_PATH
        cAlternate As String * 14
End Type

Private Declare Function FindFirstFile Lib "kernel32" Alias "FindFirstFileA" _
                        (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindNextFile Lib "kernel32" Alias "FindNextFileA" _
                        (ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Private Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As Long

Private Const FILE_ATTRIBUTE_DIRECTORY = &H10
```

HINWEIS Lassen Sie sich nicht abschrecken, das eigentliche Programm ist wesentlich kürzer, als es obige Funktionen erwarten lassen!

Der Prozedur *GetAllFiles* übergeben Sie einfach den Suchpfad sowie das Suchkriterium (z.B. *.* für alle Dateien):

```
Sub GetAllFiles(directory As String, mask As String)
Dim rec As WIN32_FIND_DATA
Dim filename As String
Dim fh As Long
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
  DoEvents
  If Right$(directory, 1) <> "\" Then directory = directory & "\"
  fh = FindFirstFile(directory & mask, rec)
  If fh = 0 Then Exit Sub
  Do
   filename = Left$(rec.cFileName, InStr(rec.cFileName, Chr$(0)) - 1)
   If (rec.dwFileAttributes And FILE_ATTRIBUTE_DIRECTORY)=FILE_ATTRIBUTE_DIRECTORY _
   Then
    If (filename <> ".") And (filename <> "..") Then GetAllFiles directory & filename, mask
   Else
     dateien = dateien & directory & filename & Chr$(13) & Chr$(10)
   End If
  Loop While FindNextFile(fh, rec)
  FindClose fh
End Sub
```

Die Verwendung von *GetAllFiles* in unserem Testprogramm:

```
Private dateien As String
...
Private Sub Befehl0_Click()
  dateien = ""
  GetAllFiles Text0.Value, Text1.Value
  Text2.Value = dateien
End Sub
```

Damit gleich nach dem Start ein erster Test möglich wird, initialisieren wir die beiden oberen Textfelder mit Erfolg versprechenden Vorgaben:

```
Private Sub Form_Load()
  Text0.Value = "C:\Programme\Microsoft Office\Office12"
  Text1.Value = "*.exe"
End Sub
```

Test

Nach Eingabe von Suchpfad und Suchkriterium kann es losgehen:

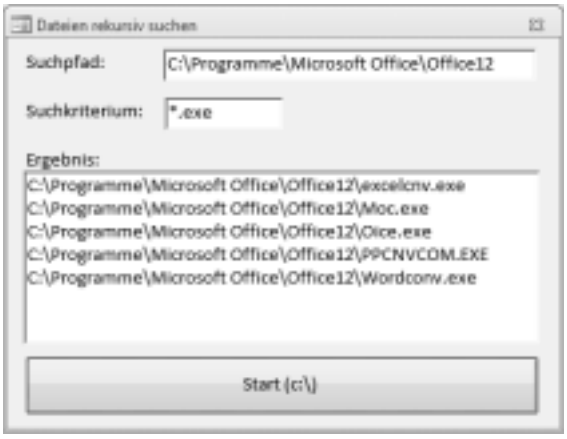

Abbildung 8.10 Beispiel für die Dateisuche (alle Office-Anwendungen werden angezeigt)

Bemerkungen

- Sollen weitere Selektionen über die Dateiattribute vorgenommen werden, müssen Sie diese mit den FILE_...-Konstanten durchführen (*If (rec.dwFileAttributes And FILE_ATTRIBUTE ...)*)
- Falls es Sie interessiert, können Sie über die Struktur *WIN32_FIND_DATA* weitere Informationen über die Datei einholen (Größe, Datum, DOS-Name usw.)

R8.10 Name, Pfad und Extension einer Datei ermitteln

Aufgabe

Sie haben schon einmal mit der Programmiersprache Pascal bzw. Delphi gearbeitet hat und kennen die Funktion *FileSplit*, mit der man eine vollständige Pfadangabe in die Komponenten Verzeichnis, Dateiname und Extension zerlegen kann?

VBA bietet leider keine solch komfortable Funktion, obwohl sie häufig gebraucht wird, beispielsweise für die SELECT-Klausel in SQL-Strings:

```
SELECT * INTO " & file & " IN '" & path & "' 'dBase IV;' FROM [ADRESSE]
```

Woher die beiden Bezeichner für *file* und *path* nehmen, wenn nur ein kompletter Dateiname, wie zum Beispiel *C:\Daten\DBase\Adressen\Adresse.dbf*, vorliegt?

Lösung

Wir entwickeln eine Routine *FileSplit*, die den übergebenen Dateipfad unter Verwendung diverser Zeichenkettenfunktionen (*Mid$, Left$, Right$, Len, InStr, ...*) in seine Bestandteile »zerpflückt«.

Oberfläche

Ein *Textfeld*, eine *Befehlsschaltfläche* und mehrere *Bezeichnungsfeld*er genügen für den Funktionstest (siehe Abbildung am Schluss).

Quelltext

```
Option Explicit
```

Die folgende *FileSplit*-Funktion prüft zuerst auf das Vorhandensein eines Punktes. Falls dieser vorhanden ist, werden alle rechts davon befindlichen Zeichen als Extension interpretiert (bei mehr als drei Zeichen handelt es sich wahrscheinlich nicht um eine Extension). Bei der Suche nach dem Punkt müssen wir unbedingt am Ende des Strings anfangen, da ein Dateiname auch mehrere Punkte enthalten kann. Danach wird der am weitesten rechts stehende Backslash (\) gesucht. An dieser Stelle ist der String zu teilen.

```
Private Sub FileSplit(ByVal s As String, path As String, file As String, ext As String)
```

HINWEIS Die zusätzliche Angabe von *ByVal* für den ersten Parameter von *FileSplit* ist unbedingt erforderlich, da sonst die übergebene Variable durch die Prozedur verändert werden würde.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
Dim i As Integer
 For i = Len(s) To 1 Step -1
    If Mid$(s, i, 1) = "\" Then           ' keine Extension vorhanden
        ext = ""
        Exit For
    End If
    If Mid$(s, i, 1) = "." Then
        ext = Right$(s, Len(s) – i)
        s = Left$(s, i – 1)
        Exit For
    End If
 Next i
 i = Len(s)
 If InStr(s, "\") <> 0 Then
    While Mid$(s, i, 1) <> "\"
        i = i – 1
    Wend
 End If
 path = Left$(s, i)
 file = Right$(s, Len(s) – i)
End Sub
```

Übergeben Sie der Prozedur die Werte in folgender Reihenfolge:

- Dateibezeichnung
- Variable für das Verzeichnis
- Variable für den Dateinamen
- Variable für die Extension

Für unsere Testoberfläche gestalten sich Aufruf und Ergebnisübergabe wie folgt:

```
Private Sub Button0_Click()
Dim pfad As String, file As String, ext As String

    FileSplit Text0.Value, pfad, file, ext
    Bezeichnungsfeld1.Caption = pfad
    Bezeichnungsfeld2.Caption = file
    Bezeichnungsfeld3.Caption = ext
End Sub
```

Damit gleich zu Beginn ein sinnvoller Dateipfad vorliegt, initialisieren wir das Textfeld mit dem Pfad der aktuellen Datenbank:

```
Private Sub Form_Load()
  Text0.Value = Application.CurrentDb.Name
End Sub
```

Test

Geben Sie einen gültigen Dateipfad ein und lassen Sie ihn in seine Bestandteile zerlegen (Abbildung 8.11)

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte
© 2011 O'Reilly Verlag GmbH & Co. KG; ISBN 978-3-86645-098-1

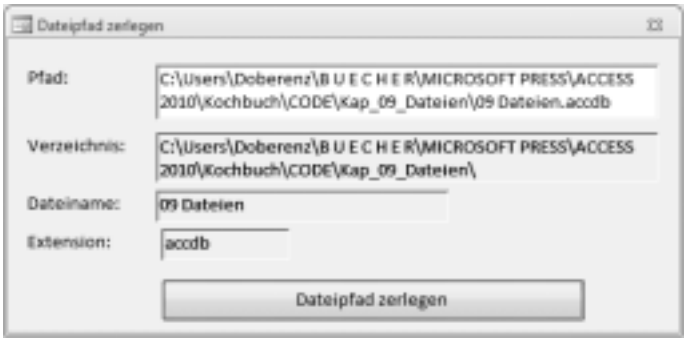

Abbildung 8.11 Beispiel für das Zerlegen eines Dateipfads

HINWEIS Beachten Sie, dass die Verzeichnisausgabe immer mit einem Backslash (\) abgeschlossen wird!

R8.11 Einen Temp-Dateinamen erzeugen

Aufgabe

In manchen Anwendungen stehen Sie vor dem Problem, Daten während des Programmablaufs in einer Datei sichern zu müssen, die bei Programmende wieder gelöscht wird. Aber welchen Namen sollen Sie dieser Datei geben, damit andere gleichnamige Dateien nicht überschrieben werden?

Lösung

Sie verwenden die beiden API-Funktionen *GetTempFileName* und *GetTempPath*, die den Dateinamen bzw. das temporäre Verzeichnis des Computers ermitteln. Ganz nebenbei zeigen wir Ihnen noch, wie Sie mit Hilfe der *Dir*-Standardfunktion einen kurzen in einen langen Dateinamen verwandeln und wie Sie mittels *Kill*-Methode eine Datei löschen.

Oberfläche

Zwei ungebundene *Textfeld*er und zwei *Befehlsschaltfläche*n genügen zum Testen (siehe Laufzeitansicht).

Quelltext

Zu einem neuen Modul (oder einem bereits vorhandenen) fügen Sie zunächst zwei API-Deklarationen hinzu:

```
Option Explicit

Private Declare Function GetTempFileNameA Lib "kernel32" (ByVal lpszPath As String, _
                                    ByVal lpPrefixString As String, _
                                    ByVal wUnique As Long, ByVal lpTempFileName As String) As Long

Private Declare Function GetTempPathA Lib "kernel32" (ByVal nBufferLength As Long, _
                                                      ByVal lpBuffer As String) As Long
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte
© 2011 O'Reilly Verlag GmbH & Co. KG; ISBN 978-3-86645-098-1

Die folgende Funktion kapselt obige API-Aufrufe:

```
Public Function GetTempName() As String
  Dim p As String, d As String
  p = Space(260)
  d = Space(260)
  GetTempPathA 255, d
  GetTempFileNameA d, "$", 0, p
  GetTempName = p
End Function
```

Eine weitere Funktion verwandelt die von *GetTempName* erzeugten kurzen (max. acht Zeichen langen) Verzeichnisse wieder in lange Dateipfade:

```
Public Function GetLongPath(ByVal shortPath As String) As String
  Dim folder() As String
  Dim tmp1 As String, tmp2 As String
  Dim longPath As String
  Dim i As Integer
  If Right$(shortPath, 1) = "\" Then shortPath = Left$(shortPath, Len(shortPath) - 1)
  folder = Split(shortPath, "\")
  tmp1 = folder(0)
  tmp2 = tmp1
  For i = 1 To UBound(folder)
    tmp1 = tmp1 & "\" & folder(i)
    longPath = Dir$(tmp1, vbNormal + vbHidden + vbSystem + vbDirectory)  ' gibt Dateinamen entsprechend
                                                                         ' Suchmuster zurück
    If longPath <> "" Then tmp2 = tmp2 & "\" & longPath
  Next i
  If longPath <> "" Then GetLongPath = tmp2 Else GetLongPath = ""
End Function
```

Wechseln Sie nun in das Codefenster des Formulars und besetzen Sie die *Click*_Eventhandler der beiden Schaltflächen:

```
Option Explicit
```

Temp-Datei erzeugen und Pfad anzeigen:

```
Private Sub Befehl0_Click()
  Text0.Value = GetTempName               ' kurzer Dateiname
  Text1.value = GetLongPath(Text0.Value)  ' langer Dateiname
End Sub
```

Temp-Datei löschen:

```
Private Sub Befehl1_Click()
On Error GoTo fehler
  If MsgBox("Temporäre Datei löschen?", 36, "Frage") = 6 Then
     Kill Text0.Value
  End If
  Exit Sub
fehler:   MsgBox "Datei nicht vorhanden!"
End Sub
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Test

Nach Aufruf der *GetTempName*-Funktion befindet sich die TEMP-Datei bereits im angezeigten *\Temp*-Verzeichnis, Sie sollten sie anschließend sofort wieder löschen.

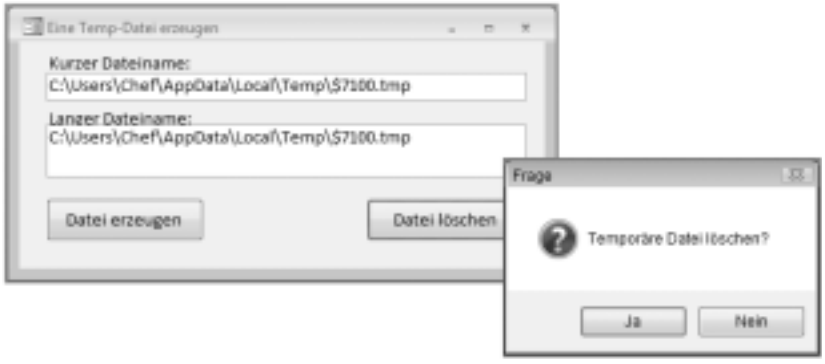

Abbildung 8.12 Laufzeitansicht

HINWEIS Da in unserem Beispiel das temporäre Verzeichnis des PCs keine langen Verzeichnispfade (>8 Zeichen) enthält, sind kurzer und langer Dateiname identisch (auf Ihrem PC kann das durchaus anders aussehen).

R8.12 Die Shellfunktionen für Dateioperationen nutzen

Aufgabe

Sie wollen Ihr Programm um eine Kopier-, Verschiebe- und Löschfunktion für Dateien bereichern, möchten dazu aber nicht unbedingt das Rad neu erfinden.

Lösung

Die Windows-Shell bzw. der Explorer stellen über die API-Schnittstelle die wohl jedem bekannten Dialoge zum Kopieren, Verschieben und Löschen von Dateien und Verzeichnissen zur Verfügung.

All diese Aufgaben werden über die Funktion *SHFileOperation* abgewickelt. Die Optionen, Dateinamen etc. übergeben Sie in einem Record mit folgendem Aufbau:

```
Private Type SHFILEOPSTRUCT
        hwnd As Long
        wFunc As Long
        pFrom As String
        pTo As String
        fFlags As Integer
        fAnyOperationsAborted As Boolean
        hNameMappings As Long
        lpszProgressTitle As String
End Type
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte
© 2011 O'Reilly Verlag GmbH & Co. KG; ISBN 978-3-86645-098-1

Die Parameter im Einzelnen:

- Handle des übergeordneten Fensters (Ihre Anwendung)
- Konstante für die Funktion (*FO_MOVE, FO_COPY, FO_DELETE, FO_RENAME*)
- Quelle (ein oder mehrere nullterminierte Strings, die mit einem Leerstring abgeschlossen werden)
- Ziel (siehe Quelle)
- Spezielle Optionen (siehe Online-Hilfe)
- Rückgabe von *True* bei vorzeitigem Abbruch der Operation
- Zeiger auf ein Array, mit dem überprüft werden kann, welche Dateien kopiert oder verschoben wurden
- Titel für ein Dialogfeld mit Fortschrittsbalken

Um die Funktionen möglichst einfach in ein Programm einbinden zu können, integrieren wir diese in ein Klassenmodul, das entsprechende Methoden bereitstellt.

Oberfläche

Für den Test der Dateioperationen verwenden wir ein Formular mit zwei *Textfeld*ern und drei *Befehlsschaltfläche*n (siehe Laufzeitansicht am Schluss des Beispiels).

Quelltext (Klassenmodul)

Über die Hauptregisterkarte *Erstellen/Andere* erzeugen wir ein neues Klassenmodul mit dem Namen *FileOperation*. Neben der eingangs schon erwähnten Struktur *SHFILEOPSTRUCT* nehmen wir noch folgende Konstanten und die API-Funktion *SHFileOperation* in den Deklarationsabschnitt auf:

```
Private Const FO_MOVE = &H1
Private Const FO_COPY = &H2
Private Const FO_DELETE = &H3
Private Const FO_RENAME = &H4
Private Const FOF_MULTIDESTFILES = &H1
Private Const FOF_CONFIRMMOUSE = &H2
Private Const FOF_SILENT = &H4
Private Const FOF_RENAMEONCOLLISION = &H8
Private Const FOF_NOCONFIRMATION = &H10
Private Const FOF_WANTMAPPINGHANDLE = &H20
Private Const FOF_ALLOWUNDO = &H40
Private Const FOF_FILESONLY = &H80
Private Const FOF_SIMPLEPROGRESS = &H100
Private Const FOF_NOCONFIRMMKDIR = &H200
Private Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" ( _
                                                              lpFileOp As SHFILEOPSTRUCT) As Long
```

Die Übergabewerte für die Kopierfunktion sind ein Array mit den Dateinamen, das Zielverzeichnis sowie eine optionale boolesche Variable, die bestimmt, ob auch Unterverzeichnisse kopiert werden sollen.

```
Public Sub Copy(dateiNamen() As String, zielVerzeichnis As String, _
                                                    Optional inklusiveUnterverzeichnisse)
 Dim fileNames As String
 Dim i As Integer
 Dim shellInfo As SHFILEOPSTRUCT
```

Die Funktion *SHFileOperation* kann mit einem Stringarray nichts anfangen, wir müssen einen String erzeugen, in dem die einzelnen Einträge mit einem Null-Zeichen voneinander getrennt sind:

```
 For i = 0 To UBound(dateiNamen)
    fileNames = fileNames & dateiNamen(i) + Chr$(0)
 Next i
 fileNames = fileNames + Chr$(0)

With shellInfo
  .hwnd = Screen.ActiveForm.hwnd
  .wFunc = FO_COPY
  .pFrom = fileNames
  .pTo = zielVerzeichnis
  If Not IsMissing(inklusiveUnterverzeichnisse) Then
    If Not inklusiveUnterverzeichnisse Then .fFlags = FOF_FILESONLY
  End If
 End With
 SHFileOperation shellInfo
End Sub
```

Das Verschieben von Dateien funktioniert analog:

```
Public Function Move(dateiNamen() As String, zielVerzeichnis As String, _
                                    Optional inklusiveUnterverzeichnisse)
 Dim fileNames As String
 Dim i As Integer
 Dim shellInfo As SHFILEOPSTRUCT

 For i = 0 To UBound(dateinamen)
    filenames = fileNames & dateiNamen(i) & Chr$(0)
 Next i
 fileNames = fileNames & Chr$(0)

 With shellInfo
  .hwnd = Screen.ActiveForm.hwnd
  .wFunc = FO_MOVE
  .pFrom = fileNames
  .pTo = zielVerzeichnis
  If Not IsMissing(inklusiveUnterverzeichnisse) Then
    If Not inklusiveUnterverzeichnisse Then .fFlags = FOF_FILESONLY
  End If
 End With
 SHFileOperation shellInfo
End Function
```

Beim Löschen von Dateien brauchen wir ein Zielverzeichnis nicht anzugeben:

```
Public Function Delete(dateiNamen() As String, Optional inklusiveUnterverzeichnisse)
 Dim fileNames As String
 Dim i As Integer
 Dim shellInfo As SHFILEOPSTRUCT

 For i = 0 To UBound(dateiNamen)
   fileNames = fileNames & dateiNamen(i) & Chr$(0)
 Next i
 fileNames = fileNames + Chr$(0)
 With shellInfo
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
  .hwnd = Screen.ActiveForm.hwnd
  .wFunc = FO_DELETE
  .pFrom = fileNames
  .pTo = "" + Chr$(0)
  If Not IsMissing(inklusiveUnterverzeichnisse) Then
    If Not inklusiveUnterverzeichnisse Then .fFlags = FOF_FILESONLY
  End If
 End With
 SHFileOperation shellInfo
End Function
```

Quelltext (Formular)

```
Option Explicit

Private s(0) As String

Private Sub Befehl0_Click()
    s(0) = Text0.Value
    FileOperation.Copy s, Text1.Value
End Sub

Private Sub Befehl1_Click()
   s(0) = Text0.Value
   FileOperation.Move s, Text1.Value
End Sub

Private Sub Befehl2_Click()
    s(0) = Text0.Value
    FileOperation.Delete s, False
End Sub
```

Wir sorgen dafür, dass gleich zu Beginn etwas Sinnvolles in den Textfeldern steht (in unserem Fall befindet sich eine Datei *Test.txt* im Projektverzeichnis):

```
Private Sub Form_Load()
  Text0.Value = Application.CurrentProject.path & "\Test.txt"
  Text1.Value = "C:\"
End Sub
```

Test

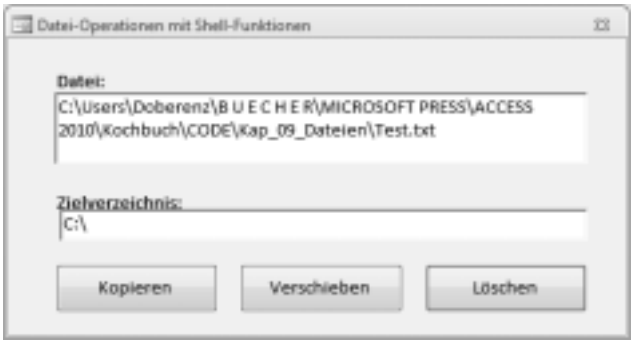

Abbildung 8.13 Die Testoberfläche (Laufzeitansicht)

Bevor Sie loslegen sollten Sie sich vergewissern, ob die Datei und das Zielverzeichnis tatsächlich vorhanden sind, andernfalls erhalten Sie eine entsprechende Fehlermeldung.

Bemerkungen

Einer der wesentlichsten Vorteile der Funktion *SHFileOperation* ist die automatische Anzeige von Zusatzdialogfeldern (z.B. beim Löschen oder Überschreiben von Dateien, Erzeugen von Verzeichnissen etc.).

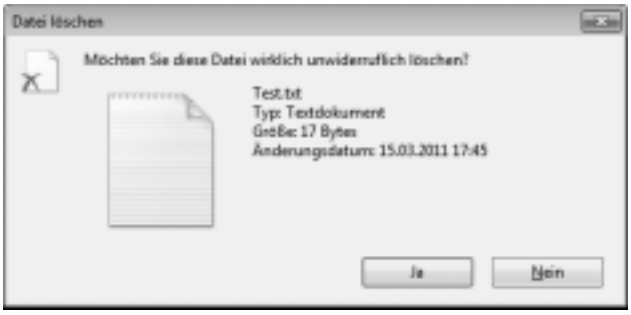

Abbildung 8.14 Beispiel für Dialogfeld der Funktion *SHFileOperation*

R8.13 Die verfügbaren Laufwerke feststellen

Aufgabe

Sie wollen ein automatisches Backup der Datenbank durchführen und müssen sich unter anderem auch für den Typ des Ziellaufwerks interessieren. Beispielsweise dürfte ein CD-ROM-Laufwerk für die Datensicherung ein gänzlich ungeeignetes Medium sein.

Lösung

Mit *GetDriveType* steht dem Programmierer eine API-Funktion zur Verfügung, die alle bekannten Laufwerkstypen problemlos unterscheiden kann.

Oberfläche

Ein *Listenfeld* (*Herkunftstyp* = *Wertliste*) und eine *Befehlsschaltfläche* genügen (siehe Laufzeitansicht).

Quelltext

Bevor Sie die API-Funktion einsetzen können, müssen Sie diese vorschriftsmäßig deklarieren:

```
Private Declare Function GetDriveType Lib "kernel32" Alias "GetDriveTypeA" _
                                                          (ByVal nDrive As String) As Long
```

Zusätzlich sind einige vordefinierte Konstanten einzubinden:

```
Const DRIVE_REMOVABLE = 2         ' Diskette
Const DRIVE_FIXED = 3             ' Festplatte
Const DRIVE_REMOTE = 4            ' Netzlaufwerk
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
Const DRIVE_CDROM = 5             ' CD-Rom
Const DRIVE_RAMDISK = 6           ' RAM-Disk
```

Im *Form_Load*-Ereignis wird das Listenfeld gefüllt:

```
Private Sub Form_Load()
Dim Typ As Integer, i As Integer, s As String
 s = ""
 For i = 0 To 25
  Typ = GetDriveType(Chr$(i + 65) + ":\")
    If Typ <> 0 Then
      Select Case Typ
          Case DRIVE_REMOVABLE
              If s <> "" Then
                 s = s & ";""" & Chr$(i + 65) & ": (Diskette)"""
              Else
                 s = s & """" & Chr$(i + 65) & ": (Diskette)"""
              End If
          Case DRIVE_FIXED
              If s <> "" Then
                  s = s & ";""" & Chr$(i + 65) & ": (Festplatte)"""
              Else
                  s = s & """" & Chr$(i + 65) & ": (Festplatte)"""
              End If
          Case DRIVE_CDROM
              If s <> "" Then
                  s = s & ";""" & Chr$(i + 65) & ": (CDROM)"""
              Else
                  s = s & """" & Chr$(i + 65) & ": (CDROM)"""
              End If
          Case DRIVE_RAMDISK
              If s <> "" Then
                  s = s & ";""" & Chr$(i + 65) & ": (RAM-Disk)"""
              Else
                  s = s & """" & Chr$(i + 65) & ": (RAM-Disk)"""
              End If
          Case DRIVE_REMOTE
              If s <> "" Then
                s = s & ";""" & Chr$(i + 65) & ": (NETZ-Laufwerk)"""
              Else
                s = s & """" & Chr$(i + 65) & ": (NETZ-Laufwerk)"""
             End If
        End Select
    End If
 Next i
 Liste1.RowSource = s             ' Listenfeld anbinden
End Sub
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Test

Abbildung 8.15 Die Laufwerkstypen werden angezeigt

R8.14 Den freien Festplattenspeicher ermitteln

Aufgabe

Da VBA keine geeignete Funktion für die Bestimmung des freien Festplattenspeichers anbietet, ist wieder einmal Selbsthilfe angesagt.

Lösung

Die Win32-Schnittstelle hält auch hier eine Lösung parat. Der Aufruf der entsprechenden API-Funktion ist allerdings nicht ganz trivial, Sie müssen sogar etwas mit Bits und Bytes rechnen. Die folgende Abbildung zeigt das Grundprinzip der Festplattenverwaltung.

Ein Cluster verwaltet mehrere Sektoren, diese wiederum bestehen aus einer bestimmten Anzahl von Bytes (in der Regel 512). Die Anzahl der Sektoren pro Cluster ist abhängig vom Dateisystem und der Festplatten- bzw. Partitionsgröße. Eine Festplatte, die mit dem FAT-Dateisystem formatiert ist, verwaltet 64 Sektoren pro Cluster zu je 512 Byte. Da jede Datei mindestens einen Cluster belegt, ergibt sich aus obigen Beziehungen ein minimaler Platzbedarf von 32.768 Bytes. Dies gilt auch, wenn die Datei leer ist (eine ungeheure Platzverschwendung!). Wer seine Festplatte effektiver verwalten will, sollte eine kleinere Partitionsgröße wählen oder die Festplatte mit dem NTFS- oder dem FAT32-Dateisystem formatieren (weniger Sektoren pro Cluster).

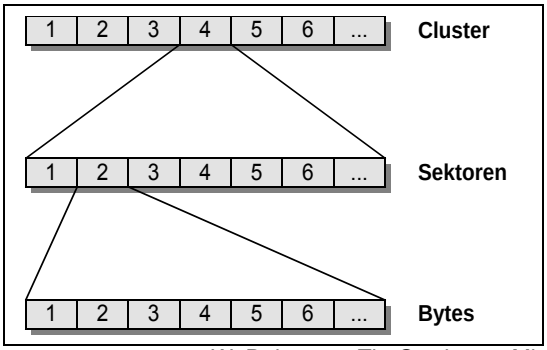

Abbildung 8.16 Prinzip der Festplattenverwaltung

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Oberfläche

Für die grafische Anzeige verwenden wir zwei sich überlappende *Rechteck*-Steuerelemente (*bar, barx*), wobei die endgültige Breite von *barx* erst zur Laufzeit festgelegt wird. Damit haben wir eine einfache Möglichkeit, ein Balkendiagramm zu erstellen. Weiterhin werden noch einige *Bezeichnungsfeld*er und eine *Befehlsschaltfläche* benötigt (siehe Laufzeitabbildung).

Quelltext

Nach dem Einbinden der API-Funktion

```
Public Declare Function GetDiskFreeSpace Lib "kernel32" Alias "GetDiskFreeSpaceA" _
        (ByVal lpRootPathName As String, lpSectorsPerCluster As Long, lpBytesPerSector As Long, _
         lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters As Long) As Long
```

definieren Sie zwei neue Funktionen:

```
Public Function GetDriveFree(lw As String) As Double
Dim SectorsPerCluster As Long, BytesPerSector As Long, NumberOfFreeClusters As Long, _
                                                                    TotalNumberOfClusters As Long

   GetDiskFreeSpace Left(lw, 1) & ":\", SectorsPerCluster, BytesPerSector, NumberOfFreeClusters, _
                                                                    TotalNumberOfClusters
   GetDriveFree = 1# * SectorsPerCluster * BytesPerSector * NumberOfFreeClusters
End Function

Public Function GetDriveSize(lw As String) As Double
Dim SectorsPerCluster As Long, BytesPerSector As Long, NumberOfFreeClusters As Long
Dim TotalNumberOfClusters As Long

   GetDiskFreeSpace Left(lw, 1) & ":\", SectorsPerCluster, BytesPerSector, NumberOfFreeClusters, _
                                                                    TotalNumberOfClusters
   GetDriveSize = 1# * SectorsPerCluster * BytesPerSector * TotalNumberOfClusters
End Function
```

Übergabeparameter ist in beiden Fällen der Laufwerksbuchstabe.

HINWEIS Vergessen Sie nicht die Multiplikation mit einem *Double*-Wert (*1#*), andernfalls kann es bei großen Festplatten zu einem Werteüberlauf kommen!

Der Aufruf der beiden neuen Funktionen am Beispiel des Laufwerks C:

```
Private Sub Befehl0_Click()
Dim länge As Long
Dim free As Double
Dim size As Double

Const gb = 1073741824               ' Umrechnungsfaktor Byte => GigaByte

    free = GetDriveFree("C")
    size = GetDriveSize("C")
    länge = barx.Width
    bar.Width = länge * free / size
```

```
    BezeichnungsfeldO.Caption = Format$(free / gb, "#.O") & " GByte"
    Bezeichnungsfeld1.Caption = Format$(size / gb, "#.O") & " GByte"
End Sub
```

Test

Die Abbildung zeigt die Werte für die Festplatte des Autors. Rechts daneben ist zu Vergleichszwecken der entsprechende Eigenschaftendialog von Windows dargestellt, der die Ergebnisse bestätigt.

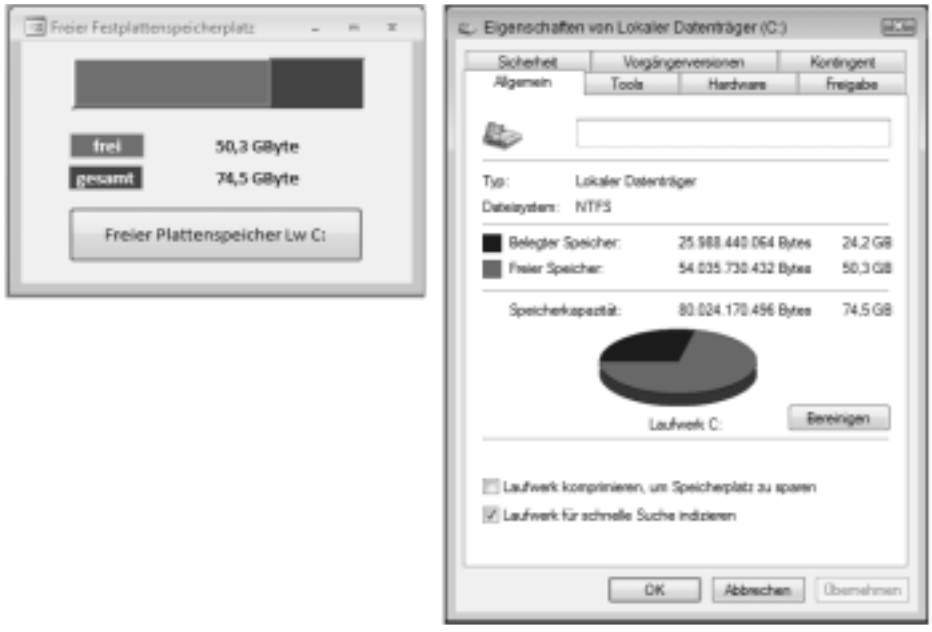

Abbildung 8.17 Laufzeitabbildung (links) und Vergleich mit dem Windows-Dialog (rechts)

R8.15 Seriennummer der Festplatte ermitteln

Aufgabe

Sie wollen einen möglichst einfachen Kopierschutz in Ihre Programme einbauen.

Lösung

Einen einfachen Kopierschutz, der allerdings für einen Profi nicht unüberwindbar ist, können Sie mit Hilfe der Festplattenseriennummer realisieren. Diese lesen Sie mit Hilfe der API-Funktion *GetVolumeInformation* aus.

Oberfläche

Zum Austesten genügt ein Formular mit einer *Befehlsschaltfläche*.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Quelltext

Binden Sie zunächst die folgende API-Funktion ein:

```
Private Declare Function GetVolumeInformation Lib "kernel32" Alias "GetVolumeInformationA" _
        (ByVal lpRootPathName As String, ByVal pVolumeNameBuffer As String, _
             ByVal nVolumeNameSize As Long, lpVolumeSerialNumber As Long, _
                  lpMaximumComponentLength As Long, lpFileSystemFlags As Long, _
                    ByVal lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long
```

Unsere Funktion vereinfacht den Aufruf von *GetVolumeInformation*:

```
Public Function GetHDNumber(Drive As String) As Long
Dim serId As Long
Dim dummy1 As String, dummy2 As String, dummy3 As Long, dummy4 As Long
   dummy1 = Space(260)
   dummy2 = Space(260)
   GetVolumeInformation Left$(Drive, 1) + ":\" & Chr$(0), dummy1, 260, serid, dummy3, _
                        dummy4, dummy2, 260
   GetHDNumber = serId
End Function
```

Mit obiger Funktion genügt folgender Code zum Auslesen der Seriennummer von Laufwerk »C:«:

```
Private Sub Befehl0_Click()
    MsgBox GetHDNumber("c")
End Sub
```

Test

Abbildung 8.18 Das Programm in Aktion

R8.16 Das Volume-Label lesen/schreiben

Aufgabe

Sie möchten aus Ihrem Programm heraus auf den Laufwerksbezeichner (Label) zugreifen.

Lösung

Auch hier bleibt Ihnen nur der mühsame Weg über die API-Funktionen.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Oberfläche

In den Detailbereich eines Formulars setzen Sie ein *Textfeld* und eine *Befehlsschaltfläche*.

Quelltext

Einbinden der API-Funktionen:

```
Private Declare Function SetVolumeLabel Lib "kernel32" Alias "SetVolumeLabelA" _
        (ByVal lpRootPathName As String, ByVal lpVolumeName As String) As Long
Private Declare Function GetVolumeInformation Lib "kernel32" Alias _
   "GetVolumeInformationA" (ByVal lpRootPathName$, ByVal pVolumeNameBuffer$, ByVal nVolumeNameSize&, _
        lpVolumeSerialNumber&, lpMaximumComponentLength&, lpFileSystemFlags&, _
            ByVal lpFileSystemNameBuffer$, ByVal nFileSystemNameSize&) As Long
```

Die folgende Funktion vereinfacht Ihnen den Aufruf von *GetVolumeInformation*:

```
Public Function GetVolName(Drive As String) As String
Dim Labelname As String, dummy2 As String, dummy3 As Long, dummy4 As Long, dummy5 As Long

   Labelname = Space(260)
   dummy2 = Space(260)
   GetVolumeInformation Left$(Drive, 1) & ":\", Labelname, 260, dummy5, dummy3, dummy4, dummy2, 260
   GetVolName = Left$(Labelname, InStr(Labelname, Chr$(0)) – 1)
End Function
```

Zum Setzen des Laufwerksbezeichners verwenden Sie die folgende Funktion:

```
Public Sub SetVolName(Drive As String, name As String)
  SetVolumeLabel Left$(Drive, 1) & ":\", name
End Sub
```

So verwenden Sie die beiden Funktionen:

```
Private Sub Befehl0_Click()
  SetVolName "C", Text1.Text
End Sub

Private Sub Form_Load()
  Text1.Value = GetVolName("c")
End Sub
```

Test

Unmittelbar nach dem Start wird das aktuelle Volume-Label des Laufwerks *C:* angezeigt, Sie können den Namen ändern und über die *Change*-Schaltfläche neu zuweisen.

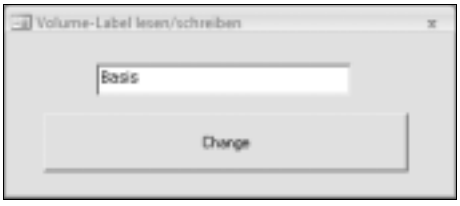

Abbildung 8.19 Laufzeitansicht

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

R8.17 Das Dateisystem bestimmen

Aufgabe

Sie möchten wissen, mit welchem Dateisystem (NTFS, FAT, ...) ein bestimmter Datenträger Ihres Computers formatiert ist.

Lösung

Auch hier hilft die API-Funktion *GetVolumeInformation* weiter. Um ihre Anwendung zu vereinfachen, wird der Aufruf in der Funktion *GetFileSystemType* gekapselt. Als Parameter ist der Pfad zu übergeben, von dem der Name des Dateisystems zu ermitteln ist (lokaler Pfad oder UNC Pfad). Misslingt der Funktionsaufruf, wird ein abfangbarer Fehler ausgelöst.

Oberfläche

Auf ein Formular setzen Sie ein *Textfeld* für die Eingabe des Laufwerkpfades, eine *Befehlsschaltfläche* und einige *Bezeichnungsfeld*er (siehe Laufzeitansichten).

Quelltext

HINWEIS Die Deklaration der API-Funktion *GetVolumeInformation* entspricht der Deklaration in den Rezepten R8.15 bzw. R9.16 und braucht deshalb hier nicht nochmals wiederholt zu werden.

Lediglich drei Konstanten für die Fehlerbehandlung sind dem API-Modul hinzuzufügen:

```
Public Const ERROR_PATH_NOT_FOUND As Long = 3&
Public Const ERROR_NOT_READY      As Long = 21&
Public Const ERROR_BAD_NETPATH    As Long = 53&
```

Der Formularcode:

```
Option Explicit

Public Function FileSystemType(ByVal RootPath As String) As String
  Dim lRet        As Long
  Dim lFileSystem As String
  Dim lSlashCount As Integer
  Dim lUNCRoot    As Long

  If Left$(RootPath, 2) <> "\\" Then
    RootPath = Left$(RootPath, 1) & ":\"
  Else
    If Right$(RootPath, 1) <> "\" Then
      RootPath = RootPath & "\"
    End If
    lUNCRoot = 2
    For lSlashCount = 1 To 2
      lUNCRoot = InStr(lUNCRoot + 1, RootPath, "\")
    Next
    If CBool(lUNCRoot) Then
      RootPath = Left$(RootPath, lUNCRoot)
```

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

```
    End If
  End If
  lFileSystem = Space$(16)
  lRet = GetVolumeInformation(RootPath, vbNullString, 0, 0, 0, 0, lFileSystem, Len(lFileSystem))
  If CBool(lRet) Then
    FileSystemType = Left$(lFileSystem, InStr(1, lFileSystem, vbNullChar) - 1)
  Else
    Select Case Err.LastDllError
      Case ERROR_BAD_NETPATH
        Err.Raise 76               ' Pfad nicht gefunden
      Case ERROR_PATH_NOT_FOUND
        Err.Raise 68               ' Gerät nicht verfügbar
      Case ERROR_NOT_READY
        Err.Raise 71               ' Datenträger nicht bereit
      Case Else
        Err.Raise 5                ' Ungültiger Prozeduraufruf/Argument
    End Select
  End If
End Function
```

Der Funktionsaufruf:

```
Private Sub Befehl0_Click()
      Bezeichnungsfeld0.Caption = FileSystemType(Text0.Value)
End Sub
```

Test

Ohne viele Worte:

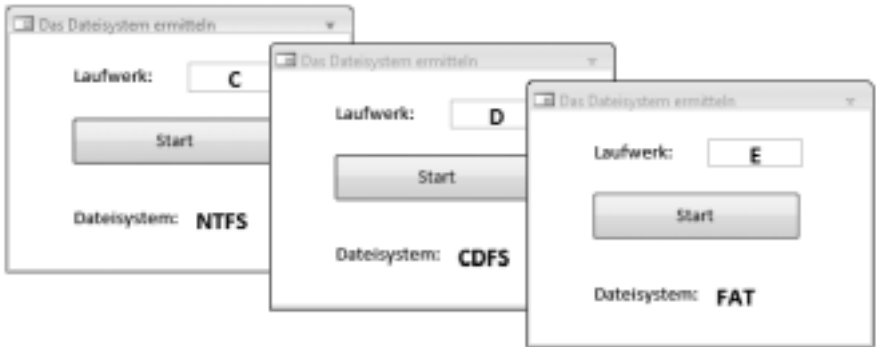

Abbildung 8.20 Laufzeitansichten

R8.18 Den UNC-Namen eines Netzlaufwerks ermitteln

Aufgabe

Sie möchten den Freigabenamen – auch als UNC (*Universal Naming Convention*) bezeichnet – eines Laufwerks ermitteln.

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte

Lösung

Die API-Funktion *WNetGetConnection* hilft Ihnen weiter.

Oberfläche

Für den Test brauchen wir lediglich ein *Bezeichnungsfeld* und eine *Befehlsschaltfläche*.

Quelltext

Wir binden zunächst die benötigte API-Funktion ein:

```
Option Explicit

Private Declare Function WNetGetConnection Lib "mpr.dll" Alias "WNetGetConnectionA" _
        (ByVal lpszLocalName As String, ByVal lpszRemoteName As String, cbRemoteName As Long) As Long
```

Die eigentliche Funktion zum Ermitteln des UNC:

```
Public Function GetUNC(Drive As String) As String
Dim RemoteName As String, RemoteNameSize As Long

    RemoteNameSize = 255
    RemoteName = Space(256)
    If WNetGetConnection(Left$(Drive, 1) & ":", RemoteName, RemoteNameSize) <> 0 Then
        GetUNC = ""
    Else
        GetUNC = Left$(RemoteName, RemoteNameSize – 1)
    End If
End Function
```

Aufruf der Funktion:

```
Private Sub Befehl0_Click()
  Bezeichnungsfeld0.Caption = GetUNC("E:")
End Sub
```

HINWEIS Bitte ändern Sie an dieser Stelle Ihren Code und geben Sie einen gültigen Netzlaufwerksbuchstaben an.

Test

Abbildung 8.21 Laufzeitansicht

W. Doberenz, Th. Gewinnus: Microsoft Access - Programmier-Rezepte