

Suchen und Kopieren im Windows-Netzwerk

# Pfadfinder

Über ein Netzwerk miteinander verbundene Rechner können freigegebene Ressourcen gemeinsam nutzen. Für netzwerkübergreifende Suchen- und Kopierfunktionen reichen die Dateisystemsteuerelemente von VB 6 und VB.NET aber nicht aus. dotnetpro zeigt, wie Sie ein benutzerdefiniertes Netzwerksteuerelement mit den Dateisystemsteuerelementen kombinieren.

Unter Windows gibt es kaum ein Programm, das nicht auf spezielle Dateien zurückgreift, in denen programmspezifische Daten abgelegt werden. Ist der Zugriff auf lokale Laufwerke, Verzeichnisse und Dateien mithilfe der Laufwerk-, Verzeichnis- und Dateilistenfelder auch sehr einfach, können Sie auf freigegebene Netzwerkressourcen ohne Verwendung spezieller API-Funktionen ausschließlich über den gemeinsamen Windows-Dialog zum Öffnen und Speichern von Dateien zugreifen. Dieser Dialog erlaubt es jedoch nicht, Netzwerkressourcen in benutzerdefinierten Dialogen einzubinden, die ein eigenes Design aufweisen.

Aus diesem Grunde hat dotnetpro das Netzwerksteuerelement für Visual Basic 6.0 und Visual Basic .NET entwickelt (siehe [1]). Mit dem Netzwerksteuerelement

## Auf einen Blick

### Autor



Dipl.-Ing. **Andreas Masto** leitet das Ingenieurbüro IngES, das sich mit der Erstellung von EDV-Publikationen und mit Software-Entwicklung befasst. Er ist ferner als freier Journalist, EDV-Berater und Fachbuchautor tätig. Sie erreichen ihn unter der E-Mail-Adresse amf@ing-es.de.

**dotnetpro code**  
A0312Netzwerkpraxis

**Sprachen** VB 6 und VB.NET

**Technik** Windows-Netzwerke

**Voraussetzungen** Das benutzerdefinierte Netzwerksteuerelement aus dotnetpro 6.2003

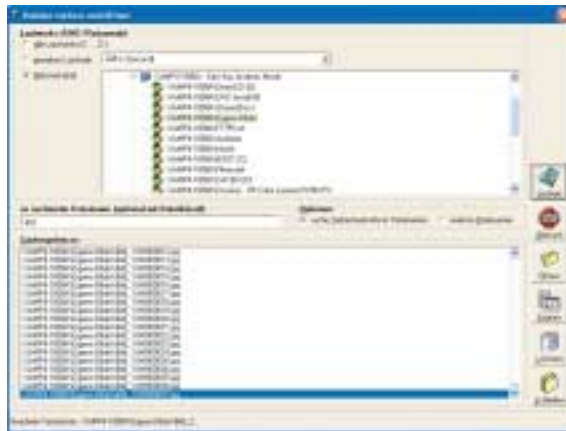


Abbildung 1 Der Suchen-Dialog in der Programmversion von Visual Basic 6.0.

können freigegebene Netzwerkressourcen in einem Windows-Netzwerk abgefragt und in hierarchischer Struktur dargestellt werden. Über das Netzwerksteuerelement können Sie Netzwerkressourcen gezielt auswählen.

### Netzwerkpfade auswählen

An dieser Stelle soll gezeigt werden, wie Sie das benutzerdefinierte Netzwerksteuerelement in Visual Basic 6.0 und Visual Basic .NET sinnvoll mit den Dateisystemsteuerelementen verbinden. Dazu müssen Sie nur wissen, dass das Netzwerksteuerelement entsprechend einem Laufwerklistenfeld genutzt werden kann. Selektierte Netzwerkpfade brauchen entsprechend einem ausgewählten Laufwerk lediglich in die *Path*-Eigenschaft eines Verzeichnislistenfeldes übergeben zu werden, um alle untergeordneten Ver-

zeichnisse einer Netzwerkressource angezeigt zu bekommen.

Wählen Sie über das Verzeichnislistenfeld einen untergeordneten Pfad einer Netzwerkressource an, so wird dieser Pfad in der Eigenschaft *Path* des Verzeichnislistenfeldes gesichert. In Verbindung mit einem Dateilistenfeld können Sie zu einem bestimmten Netzwerkpfad die darin enthaltenen Dateien auflisten lassen, indem Sie diesen Pfad in die Eigenschaft *Path* des Dateilistenfeldes übernehmen. VB.NET stellt die Dateisystemsteuerelemente ausschließlich als VB-6-Kompatibilitätsobjekte zur Verfügung. Netzwerkpfade können jedoch auch direkt an Methoden aus dem Namensraum *System.IO* übergeben werden.

An dieser Stelle soll zunächst ein Programm mit Visual Basic 6.0 definiert werden, über das Dateien netzwerkübergreifend gesucht und kopiert werden können.

Das Programm nutzt das *NetAX*-Steuerelement, dessen Quelltext Sie auch auf der Heft-CD finden.

### Netzwerkpraxis mit Visual Basic 6.0

Das Beispielprogramm zu Visual Basic 6.0 setzt sich aus drei Dialogen zusammen. Im Hauptdialog wählen Sie über Befehlschaltflächen die untergeordneten Dialoge zum Suchen und Kopieren.

Der Suchen-Dialog ist im Formular *frmFind* beziehungsweise in der Datei *Find.frm* definiert. Im Dialog legen Sie per Optionsfeld fest, ob sich die Dateisuche auf ein bestimmtes Laufwerk beschränken, auf sämtliche lokalen Laufwerke von C: bis Z: oder auf einen gewählten Netzwerkpfad beziehen soll. Es wird darauf verzichtet, sämtliche freigegebenen Netzwerkressourcen beziehungsweise alle lokalen und Netzwerkressourcen in einem einzigen Arbeitsgang auszuwerten. Bei Bedarf können Sie diese Funktionen selbst nachrüsten.

Haben Sie sich für die Durchsuchung eines bestimmten Laufwerks entschieden, so wählen Sie das Laufwerk über das nebenstehende Laufwerklistenfeld aus. Für das Durchsuchen eines Netzwerkverzeichnisses brauchen Sie den freigegebenen Netzwerkpfad lediglich per Netzwerksteuerelement zu markieren (vergleiche Abbildung 1). Die Suche selbst erfolgt über eine Zeichenkette, bei der nicht zwischen Groß- und Kleinschreibung unterschieden und die über ein Textfeld eingegeben wird. Ob es sich bei einer zu suchenden Zeichenkette um einen exakten Dateinamen oder eine Teilzeichenkette handelt, können Sie über gesonderte Optionsfelder bestimmen. Haben Sie eine Suche nach Teilzeichenketten durchgeführt, werden alle Dateieinträge zurückgeliefert, in denen die Teilzeichenkette innerhalb des Hierarchiepfades einer Datei enthalten ist. Die Treffer werden im Listenfeld *lst* der Suchergebnisse zusammengestellt.

Im rechten Bereich des Dialogfeldes sind über Befehlsschaltflächen diverse Funktionen abrufbar. Mit *Suchen* starten Sie einen Suchlauf entsprechend den Suchparametern, und mit *Abbruch* können Sie einen laufenden Suchvorgang vorzeitig unterbrechen. Da die Suchfunktion rekursiv arbeitet und ausgehend vom gewählten Startlaufwerk beziehungsweise -verzeichnis sämtliche Unterverzeichnisse berücksichtigt, ist ein Suchlauf mitunter sehr zeitaufwändig. Eine im Listenfeld der Suchergebnisse se-

lektierte Datei können Sie mit *Öffnen* zur Bearbeitung mit der verknüpften Anwendung öffnen. *Explore* öffnet das Dateiverzeichnis zu einer gefundenen Datei im Windows Explorer. Mit *Löschen* können Sie die Suchergebnisse löschen und mit *Schließen* die Suchfunktion beenden. Nähere Hinweise zu diesen Funktionen finden Sie im Quelltext auf der Heft-CD.

Der Dialog zum Kopieren von Dateien ist so aufgebaut, wie ihn Abbildung 2 in der Variante mit Visual Basic .NET zeigt. Über gesonderte Registerseiten können Sie Quell- und Zielverzeichnis für das Kopieren wählen. Beim Quellverzeichnis können Sie sich zwischen einem lokalen und einem freigegebenen Netzwerkpfad entscheiden. Die Laufwerksauswahl erfolgt erneut nach Markierung der entsprechenden Option über ein Laufwerklistenfeld. Über ein Netzwerksteuerelement wird der Netzwerkpfad selektiert. Anschließend können Sie per Verzeichnislistenfeld das zu kopierende Verzeichnis bestimmen. Die darin enthaltenen Dateien werden in der aktuellen Programmfassung zwar einzeln angezeigt, aber in jedem Fall in der Gesamtheit kopiert. Nachdem Sie das Quellverzeichnis festgelegt haben, bestimmen Sie das Zielverzeichnis über die gleichnamige Registerseite. Auch hier können Sie wahlweise ein lokales Laufwerk oder einen Netzwerkpfad festlegen, um anschließend das Zielverzeichnis zu bestimmen. Mit der Schaltfläche *neues Verzeichnis* können Sie im Zielverzeichnis gegebenenfalls zu nächst ein leeres Unterverzeichnis einrichten. Sie starten das Kopieren mit *OK* und können es vorzeitig über die Schaltfläche *Abbruch* beenden.

In der Regel werden Dateien innerhalb des Beispielprogramms nur dann kopiert, wenn die Dateien im gewählten Zielverzeichnis nicht bereits vorhanden sind. Sie können dies jedoch ändern, indem Sie auf der Registerseite *Optionen* die Option *bereits vorhandene Zielfdateien ohne Sicherheitsabfrage beenden* aktivieren. In diesem Fall werden vorhan-

dene Dateien ohne Sicherheitsabfrage überschrieben. Unter Umständen können Sie später eine weitere Option ergänzen, die dafür sorgt, dass vor jedem Überschreiben eine Sicherheitsabfrage eingeblendet wird.

### Schnelle Dateisuche

Nachdem damit die Programmfunktionalität beschrieben ist, können Sie sich der Lösung unter Visual Basic 6.0 zuwenden. Visual Basic 6.0 stellt die *Dir*-Anweisung zur Verzeichnisanalyse bereit, die für die Suchfunktion genutzt werden könnte. Die *Dir*-Funktion hat jedoch erhebliche Nachteile, sodass auf ihren Einsatz hier verzichtet wird. Zum einen ist die *Dir*-Anweisung zu langsam in der Verarbeitung, und zum anderen arbeitet sie fehlerhaft bei rekursiven Operationen. Aus diesem Grunde sollen hier die Verzeichnisinformationen unter Einsatz der Windows-API-Funktionen *FindFirstFile* und *FindNextFile* ermittelt werden. Über diese Funktionen können Dateien über eine Suchmaske ermittelt werden. *FindFirstFile* legt die Suchmaske fest und ermittelt die erste entsprechende Datei, und mit *FindNextFile* werden alle weiteren der Suchmaske entsprechenden Dateien abgefragt. Anders als bei der *Dir*-Anweisung gehen Zwischenwerte beim rekursiven Aufruf nicht verloren. Dementsprechend brauchen keine Arbeitsgänge zwischenzeitlich gesichert und wiederhergestellt zu werden. Die logische Folge: Die hierarchischen Verzeichnisoperationen können erheblich schneller ausgeführt werden.

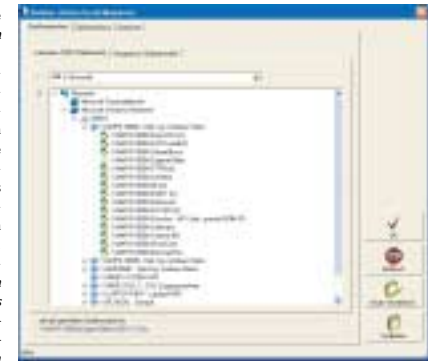


Abbildung 2 Zum Kopieren wählen Sie zuerst das Quelllaufwerk oder den UNC-Quellpfad und dann ein untergeordnetes Verzeichnis.

### Listing 1

#### Schnelle Dateisuche per Windows-API in Visual Basic 6.0.

```

Declare Function FindFirstFile Lib "kernel32" Alias _
    "FindFirstFileA" (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long
Declare Function FindNextFile Lib "kernel32" Alias _
    "FindNextFileA" (ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long
Public Const MAX_PATH = 260
Public Const FILE_ATTRIBUTE_DIRECTORY = &H10

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

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
    
```

Die API-Funktionen sind im Modul *GlbModul.bas* deklariert. Beide API-Funktionen liefern die Dateiinformationen über eine Variable der Struktur *WIN32\_FIND\_DATA* zurück. Diese enthält die hier benötigten Verzeichnis- und Dateinamen im Strukturelement *cFileName*. Die maximale Pfadlänge von 260 Zeichen wird über die Konstante *MAX\_PATH*, das Attribut für Verzeichniseinträge über die Konstante *FILE\_ATTRIBUTE\_DIRECTORY* definiert. Innerhalb der Struktur *WIN32\_FIND\_DATA* werden Datums- und Zeitangaben über eine Verbundvariable vom Typ *FILETIME* definiert. Obgleich diese Informationen hier nicht benötigt werden, muss auch diese Struktur in das Quellmodul aufgenommen werden (vergleiche Listing 1).

#### Rekursive Dateioperationen

Innerhalb des Moduls *GlbModul.bas* ist auch die benutzerdefinierte Prozedur *RecursiveFileOperation* enthalten, die für die rekursiven Dateioperationen verantwortlich ist (siehe Listing 2). Um rekursive Verzeichnisoperationen vorzeitig abbrechen zu können, wird im Modul die Wahrheitsvariable *Cancel* global verwaltet. Immer dann, wenn diese Variable durch das Betätigen der Schaltfläche *Abbruch* auf den Wert *True* gesetzt wird,

wird die rekursive Verzeichnisanalyse vorzeitig beendet.

Der Prozedur wird der Ausgangspfad der Verzeichnisoperation über den Parameter *Ausgangspfad* und die Dateisuchmaske über den Parameter *Suchmaske* übergeben. Der Parameter *Funktion* legt die gewünschte Verzeichnisoperation fest. Übergeben Sie hier die Zeichenkette *find* für die Dateisuche und die Zeichenkette *backup* für ein Kopieren.

Über den Parameter *Param1*, der ausschließlich für die Kopierfunktion benötigt wird, kann der Zielpfad für das Kopieren angegeben werden. Das Listenfeld für die Aufnahme der Suchergebnisse können Sie über den Parameter *lst* übergeben. Im Parameter *opt* wird das Optionsfeld übergeben, über das die Überschreibungsoption bestimmt wurde. Innerhalb der Prozedur wird zunächst die Variable *fStruct* vom Typ *WIN32\_FIND\_DATA* eingeführt. Mittels *FindFirstFile* wird dann die Suchmaske festgelegt und ein erster übereinstimmender Dateieintrag über die Variable *fStruct* zurückgeliefert. Ferner liefert die Funktion *FindFirstFile* eine eindeutige Kennung *hFile* zurück, mit der der jeweils nächste Dateieintrag via *FindNextFile* abgefragt werden kann.

Der ermittelte Dateiname wird aus *fStruct.cFileName* ausgelesen und an die Zeichenkettenvariable *Eintrag* überge-

ben. Ob es sich bei dem aktuell ermittelten Eintrag um einen Verzeichnis- oder Dateieintrag handelt, können Sie über das Element *dwFileAttributes* der Strukturvariablen *fStruct* ermitteln, indem Sie es auf den Wert der Konstanten *FILE\_ATTRIBUTE\_DIRECTORY* überprüfen. Handelt es sich um einen Verzeichniseintrag, kann die Routine *RecursiveFileOperation* rekursiv aufgerufen werden.

Handelt es sich bei dem aktuellen Eintrag um eine Datei, kann diese je nach gewählter Funktion für die Suche oder das Kopieren berücksichtigt werden. Gefundene Dateien werden mit dem Ausgangspfad verkettet und in das Ergebnislistenfeld übernommen. Für das Kopieren wird die Standardanweisung *FileCopy* verwendet.

Mithilfe der Funktion *FindNextFile* wird jeweils der nächste Verzeichniseintrag ermittelt. Der Funktion übergeben Sie die durch *FindFirstFile* ermittelte Kennung *hFile*. Liefert die Funktion einen Wert ungleich *Null* zurück, wurde ein weiterer Eintrag gefunden, der über das Element *fStruct.cFileName* an die Variable *Eintrag* übergeben wird. Liefert *FindNextFile* den Wert *Null* zurück, wird der Wert von Eintrag auf eine leere Zeichenkette gesetzt und einhergehend damit die Abarbeitung der *While-Wend*-Schleife beendet. Die *While-Wend*-Schleife wird auch dann beendet, wenn *Cancel* zwischenzeitlich durch entsprechende Schaltflächenanwahl auf *True* gesetzt wurde (siehe oben).

Innerhalb des Moduls ist auch die Funktion *IsNetworkPath* definiert. Diese überprüft, ob es sich bei einer übergebenen Netzwerkressource, die über das Netzwerksteuerprogramm gewählt wurde, tatsächlich um einen freigegebenen UNC-Netzwerkpfad handelt. Die Funktion liefert immer dann den Wert *True* an das aufrufende Programm zurück, wenn in einem übergebenen Pfad mindestens drei Schrägstriche vorhanden sind. UNC-Pfadnamen werden durch zwei Schrägstriche eingeleitet, worauf der Rechnername folgt. Nachfolgende Laufwerk- und Verzeichnisangaben sind durch einen weiteren Schrägstrich getrennt, womit sich die Mindestzahl von drei Schrägstrichen ergibt.

#### Netzwerkübergreifend suchen

Nachdem die rekursiven Verzeichnisoperationen codiert sind, können sie auch über die Suchen- und Kopieren-Dialoge bereitgestellt werden. Um die Suche zu starten, klicken Sie im Suchen-Dialog *Find.frm* auf die Schaltfläche *Suchen*. Dadurch wird

innerhalb des Formulars die Ereignisprozedur *btnFind\_Click* ausgeführt. Darin wird zunächst das Listenfeld *LstFind*, in das die gefundenen Übereinstimmungen eingetragen werden, gelöscht.

Ist das Optionsfeld *alle Laufwerke* (*C...Z*) markiert beziehungsweise liefert die *Value*-Eigenschaft des Optionsfelds *optDrivePath(0)* den Wert *True* zurück, werden sämtliche Laufwerke nach dem angegebenen Dateinamen beziehungsweise Teildateinamen durchsucht. Prozedurintern wer-

den die verfügbaren Laufwerkbuchstaben aus dem Laufwerklistenfeld *DriveCil* ermittelt und nacheinander abgearbeitet. Diskettenlaufwerke werden bei der Suche nicht berücksichtigt.

Der eigentliche Suchvorgang wird an die benutzerdefinierte Routine *RecursiveFileOperation* delegiert, die für sämtliche rekursiven Dateioperationen verantwortlich ist. Der Routine übergeben Sie das zu durchsuchende Laufwerk im Parameter *Drv*, den zu suchenden Dateinamen be-

ziehungsweise die zu suchende Teilzeichenkette im Parameter *Datei*, das Listenfeld zur Aufnahme der gefundenen Dateieinträge über den Parameter *LstFind* und das Optionsfeld, über das die Suchart bestimmt wird, im Parameter *optFind2*. Damit die Suchfunktion ausgeführt wird, wird im dritten Parameter das Schlüsselwort *find* übergeben.

Ist das Optionsfeld *einzelnes Laufwerk* beziehungsweise *optDrivePath(1)* markiert, wird das jeweils im Laufwerklisten-

### Listing 2

#### Rekursive Dateisuche mit Netzwerkunterstützung unter Visual Basic 6.0.

```

Public Cancel As Boolean

Sub RecursiveFileOperation(Ausgangspfad As String, _
    Suchmaske As String, Funktion As String, Param1 As String, _
    lst As ListBox, opt As OptionButton)
    On Error Resume Next
    Cancel = False
    Dim fStruct As WIN32_FIND_DATA : Dim Eintrag As String
    Dim hFile As Long : Dim Dummy As Long : Dim zPfad As String
    Dim ZielPfad As String
    While Right(Ausgangspfad, 1) = "\"
        Ausgangspfad = Left(Ausgangspfad, Len(Ausgangspfad) - 1)
    Wend
    hFile = FindFirstFile(Ausgangspfad + "*.*", fStruct)
    Eintrag = Left(fStruct.cFileName, InStr(fStruct.cFileName, Chr(0)) - 1)
    DoEvents
    If Eintrag <> "" And Eintrag <> "." Then
        If (fStruct.dwFileAttributes And _
            FILE_ATTRIBUTE_DIRECTORY) = FILE_ATTRIBUTE_DIRECTORY Then
            If Not (lst Is Nothing) Then lst.ListIndex = lst.ListCount - 1
            frmFind.stBar.SimpleText = "Bearbeite Verzeichnis: " & _
                & Ausgangspfad & "\" & Eintrag & "..."
            RecursiveFileOperation Ausgangspfad & "\" + _
                Eintrag, Suchmaske, Funktion, Param1, lst, opt
        Else
            If LCase(Funktion) = "find" Then
                If opt.Value = True Then
                    If LCase(Eintrag) = LCase(Suchmaske) Then
                        lst.AddItem Ausgangspfad + "\" + Eintrag
                        lst.Refresh
                    End If
                Else
                    If InStr(LCase(Eintrag), LCase(Suchmaske)) > 0 Then
                        lst.AddItem Ausgangspfad + "\" + Eintrag
                        lst.Refresh
                    End If
                End If
            ElseIf LCase(Funktion) = "backup" Then
                If Len(Ausgangspfad) > Len(Suchmaske) Then
                    zPfad = Right(Ausgangspfad, _
                        Len(Ausgangspfad) - Len(Suchmaske) - 1)
                Else
                    zPfad = ""
                End If
                If zPfad <> "" Then zPfad = zPfad + "\"
                ZielPfad = Param1 + "\" + zPfad
                If Not PathExist(ZielPfad) Then
                    Mkdir Filter(ZielPfad)
                End If
                If opt.Value = True Then
                    If Not FileExist(ZielPfad + Eintrag) Then
                        frmBackRes.stBar.SimpleText = "Kopiere " & _
                            & Ausgangspfad & "\" & Eintrag & " " & _
                                "nach " & ZielPfad & "..."
                        FileCopy Ausgangspfad & "\" & Eintrag, _
                            ZielPfad + Eintrag
                    End If
                Else
                    frmBackRes.stBar.SimpleText = "Kopiere " & _
                        Ausgangspfad & "\" & Eintrag & "nach " & _
                            & ZielPfad & "..."
                        FileCopy Ausgangspfad & "\" & Eintrag, _
                            ZielPfad + Eintrag
                End If
            End If
        End If
    End If
    Dummy = FindNextFile(hFile, fStruct)
    If Dummy <> 0 Then
        Eintrag = Left(fStruct.cFileName, _
            InStr(fStruct.cFileName, Chr(0)) - 1)
        Else
            Eintrag = ""
        End If
    Wend
End Sub

Function IsNetworkPath(Pfad As String) As Boolean
    Dim x As Integer
    Dim Counter As Integer
    Dim Z As String
    IsNetworkPath = False
    For x = 1 To Len(Pfad)
        If Mid(Pfad, x, 1) = "\" Then
            Counter = Counter + 1
            If Counter >= 3 Then
                IsNetworkPath = True
                Exit For
            End If
        End If
    Next x
End Function
    
```

### Listing 3

#### Schnelle Dateisuche unter Visual Basic 6.0.

```
Private Sub btnFind_Click()
    On Error Resume Next
    Dim Datei As String
    Dim Drv As String
    Dim x As Integer
    1stFind.Clear
    1stFind.Refresh
    If optDrivePath(0).Value = True Then
        Me.MousePointer = 11
        Datei = Trim(LCase(txtFind.Text))
        If Datei = "" Then
            Me.MousePointer = 0
            Exit Sub
        End If
        For x = 0 To DriveCt1.ListCount
            Drv = Left(DriveCt1.List(x), 2)
            If Drv <> "a:" And Drv <> "b:" And Drv <> "" Then
                stBar.SimpleText = "Laufwerk " & UCase(Drv) + " " & _
                    " wird durchsucht. Einen Moment Geduld ..."
                stBar.Refresh
                RecursiveFileOperation Drv, Datei, "find", "", _
                    1stFind, optFind2
            End If
        Next x
        Me.MousePointer = 0
    ElseIf optDrivePath(1).Value = True Then
        Me.MousePointer = 11
        Drv = Left(DriveCt1.Drive, 2)
        Datei = Trim(LCase(txtFind.Text))
        If Datei = "" Then
            Me.MousePointer = 0
            Exit Sub
        End If
        stBar.SimpleText = "Netzwerkpfad " & UCase(SuchPfad) & _
            " & wird durchsucht. Einen Moment Geduld ..."
        stBar.Refresh
        RecursiveFileOperation SuchPfad, Datei, "find", "", _
            1stFind, optFind2
        Me.MousePointer = 0
    End If
    stBar = "Wählen Sie eine Datei ..."
    stBar.Refresh
End Sub

Exit Sub
End If
stBar.SimpleText = "Laufwerk " & UCase(Drv) & _
    " wird durchsucht. Einen Moment Geduld ..."
stBar.Refresh
RecursiveFileOperation Drv, Datei, "find", "", _
    1stFind, optFind2
ElseIf optDrivePath(2).Value = True Then
    Dim UNCPath As String
    Dim SuchPfad As String
    Me.MousePointer = 11
    UNCPath = NetCtl1.NodeText
    If Left(UNCPath, 2) = "\\\" And IsNetworkPath(UNCPath) _
        Then
        SuchPfad = UNCPath
    End If
    Datei = Trim(LCase(txtFind.Text))
    If Datei = "" Then
        Me.MousePointer = 0
        Exit Sub
    End If
    stBar.SimpleText = "Netzwerkpfad " & UCase(SuchPfad) & _
        " & wird durchsucht. Einen Moment Geduld ..."
    stBar.Refresh
    RecursiveFileOperation SuchPfad, Datei, "find", "", _
        1stFind, optFind2
    Me.MousePointer = 0
End If
stBar = "Wählen Sie eine Datei ..."
stBar.Refresh
End Sub
```

### Listing 4

#### Dateien unter Visual Basic 6.0 netzwerkübergreifend kopieren.

```
Private Sub btnOK_Click()
    Dim Quell As String
    Dim Ziel As String
    Me.MousePointer = 11
    If LCase(lbZiel.Caption) & "\\" <> LCase$(DbPfad) Then
        If lbQuell1.Caption <> lbZiel.Caption Then
            If InStr(lbZiel, lbQuell1) = 0 Then
                Quell = lbQuell1.Caption
                Ziel = lbZiel.Caption
                RecursiveFileOperation Quell, Quell, "backup", _
                    Ziel, Nothing, optBackup2
            MsgBox "Das Kopieren wurde abgeschlossen!", _
                vbExclamation, "Kopieren beendet!"
            Me.stBar.SimpleText = "Wählen Sie ..."
            Me.MousePointer = 0
        Else
            Me.MousePointer = 0
            MsgBox "Das Zielverzeichnis ...", _
                vbExclamation, "Falsche Pfadwahl"
            Exit Sub
        End If
    Else
        Me.MousePointer = 0
    End If
Else
    Me.MousePointer = 0
End Sub

MsgBox "Quell- und Zielpfad ...", _
    vbExclamation, "Falsche Pfadwahl"
Exit Sub
End If
Else
    Me.MousePointer = 0
MsgBox "Das global bestimmte Datenbankverzeichnis ...", _
    vbExclamation, "Falsche Pfadwahl"
Exit Sub
End If
Private Sub NetCtl1_Click()
    On Error Resume Next
    Dim UNCPath As String
    If resOptNet.Value Then
        UNCPath = NetCtl1.NodeText
        If Left(UNCPath, 2) = "\\\" And IsNetworkPath(UNCPath) Then
            Dir1.Path = UNCPath
            lbQuell1.Caption = UNCPath
        End If
    End If
End Sub
```

feld *DriveCt1* angewählte Laufwerk durchsucht. Beim aktuell gewählten Laufwerk kann es sich auch um ein Diskettenlaufwerk handeln. Die Laufwerkbezeichnung wird aus dem Laufwerklistenfeld ermittelt und entsprechend der Berücksichtigung mehrerer Laufwerke an die Routine *RecursiveFileOperation* weitergeleitet.

Innerhalb des Suchen-Dialogs steht zusätzlich die Option *Netzwerkpfad* zur Verfügung. Wählen Sie diese an, und ist damit auch das Optionsfeld *optDrivePath(1)* auf *True* gesetzt, können Sie einen Netzwerkpfad über das *NetCtl*-Steuerelement wählen. Der Netzwerkpfad wird über die Eigenschaft *NodeText* aus dem aktuell markierten Knoten ermittelt, der Variablen *UNCPath* übergeben und mithilfe der Funktion *IsNetworkPath* auf Zulässigkeit überprüft. Der Text muss mit *\\* eingeleitet sein und mindestens drei Schrägstriche enthalten. Ein zulässiger Pfad wird in die Variable *SuchPfad* übernommen und an die Routine *RecursiveFileOperation* übergeben. Sämtliche gefundenen Dateien werden mithilfe der Routine *RecursiveFileOperation* samt komplettem Suchpfad in das Listenfeld *1stFind* übernommen (vergleiche Listing 3).

#### Netzwerkübergreifendes Kopieren

Über die Formulardatei *frmBackRes* beziehungsweise *BackRes.frm* können Sie Dateien durch Angabe von Quell- und Zielverzeichnis beliebig zwischen Laufwerken und Verzeichnissen kopieren. Sind Quelllaufwerk und -verzeichnis sowie Ziellaufwerk und -verzeichnis bestimmt, können Sie das gewählte Verzeichnis in der Gesamtheit durch Betätigen der Schaltfläche *OK* kopieren.

Durch Auswertung der Bezeichnungsfelder *lbQuell* (Quellpfad) und *lbZiel* (Zielpfad) wird in der Ereignisprozedur *btnOK\_Click* zunächst überprüft, ob sich Ziel- und Quellpfad voneinander unterscheiden (vergleiche Listing 4). Ist dies nicht der Fall, wird ein entsprechender Fehlerhinweis ausgegeben. Des Weiteren wird überprüft, ob das Zielverzeichnis ein Unterverzeichnis des Quellpfads ist. Auch in diesem Fall wird der Kopiervorgang vorzeitig mit einer Fehlermeldung abgebrochen. Unterscheiden sich die Pfadangaben, wird der Kopiervorgang über die Routine *RecursiveFileOperation* gestartet.

Der Routine übergeben Sie in den ersten beiden Parametern den Quellpfad, im dritten Parameter das Kennwort *backup* sowie im vierten Parameter das Zielverzeichnis. Da die Funktion keine Ergebnisse zurückliefert, wird im fünften Parameter der Wert *Nothing* übergeben. Der letzte Parameter wird erneut zur Übergabe einer Programmoption in Form eines Optionsfelds genutzt.

Bleibt darauf hinzuweisen, dass Netzwerkpfade an dieser Stelle nur dann genutzt werden, wenn die Optionsfelder zu den Netzwerksteuerelementen markiert sind. Dies gilt gleichermaßen für die Wahl des Quell- als auch des Zielverzeichnisses.

Die Netzwerkpfadzuweisung erfolgt, sofern das zugeordnete Optionsfeld markiert ist, durch Pfadauswahl im Netzwerksteuerelement. Für das Netzwerkquellverzeichnis wird dafür die Ereignisprozedur *NetCtl1\_Click* und für das Netzwerkzielverzeichnis die Ereignisprozedur *NetCtl2\_Click* ausgelöst. Innerhalb der Ereignisprozeduren wird der aktuelle Netzwerkpfad auf Zulässigkeit überprüft und in das Bezeichnungsfeld *lbQuell* für das Quellverzeichnis beziehungsweise *lbZiel* für das Zielverzeichnis übernommen. Ferner wird der Pfad in die *Path*-Eigenschaft des Verzeichnislistenfelds *Dir1* (Quellunterverzeichnis) beziehungsweise *Dir2* (Zielunterverzeichnis) übernommen.

Ansonsten gilt: Der Laufwerkwechsel des Laufwerklistenfelds *Drive1* (Quellaufwerk) wird an das Verzeichnislistenfeld *Dir1* (Quellverzeichnis) und der Verzeichniswechsel von *Dir1* an das Dateilistensteuerelement *File1* (Quelldateien) über die entsprechenden *Change*-Ereignisse weitergeleitet. Entsprechend wird der Laufwerkwechsel des Laufwerklistenfelds *Drive2* (Ziellaufwerk) an das Verzeichnislistenfeld *Dir2* (Zielverzeichnis) und der Verzeichniswechsel von *Dir2* an das Dateilistensteuerelement *File2* (Dateien im Zielverzeichnis) über die zugehörigen *Change*-Ereignisse gemeldet. Auf die Wiedergabe der entsprechenden Ereignisprozeduren wird an dieser Stelle verzichtet.

#### Migration auf Visual Basic .NET

Nachdem die Lösung damit für Visual Basic 6.0 codiert ist, soll diese nun in Visual Basic .NET realisiert werden. Da das Netzwerksteuerelement bereits in einer .NET-Programmfassung vorliegt, kann es direkt ersetzt werden. Die Quellen selbst können migriert werden, indem das VB-6-Projekt mit Visual Studio .NET geöffnet wird. Dabei gibt es jedoch einige Dinge zu beachten:

- Visual Studio .NET 2002 ist nur bedingt für die Portierung bestehender VB-6-Quellen geeignet, da weder Ressourcen korrekt übernommen noch Steuerelemente korrekt portiert werden. Steuerelementbibliotheken können zudem nicht übernommen werden. Portierte Anwendungen müssen daher umfassend auch im Windows Form Designer nachbearbeitet werden. Änderungen an Quelltexten werden über To-do-Listen zusammengestellt.
- Visual Studio .NET 2003 bietet Verbesserungen bei der Portierung von Formularbeschreibungen und Steuerelementeigenschaften und erlaubt zudem die Übernahme bestehender Steuerelementbibliotheken. Obgleich Formularbeschreibungen nur geringfügig nachzubearbeiten sind, ergeben sich für die anzupassenden Quelltexte erneut umfangreiche To-do-Listen, die abgearbeitet werden müssen.

Anzeige  
1/4 hoch

Auf der Heft-CD finden Sie Beispiele, die sich durch eine Portierung mit Visual Studio .NET 2002 beziehungsweise 2003 ergeben. Nur die mit Visual Studio .NET 2003 erzeugte Programmfassung wurde jedoch nachbearbeitet und in eine lauffähige Variante umgewandelt. Dabei wurde das ActiveX-Netzwerksteuerelement nachträglich gelöscht und gegen das benutzerdefinierte .NET-Netzwerksteuerelement ausgetauscht. Alle ActiveX-Steuerelemente werden standardmäßig in .NET-Projekte übernommen. Dabei werden die erforderlichen Interoperabilitäts-

DLLs automatisch generiert und als Verweis in das neue .NET-Projekt integriert. Durch Austausch des Netzwerksteuerelements gegen die entsprechende .NET-Variante können Sie auch die Verweise auf die Interoperabilitäts-DLLs entfernen.

### Rekursive Dateioperationen unter Visual Basic .NET

Bei der Nachbearbeitung eines Quelltextes, der mit Visual Basic 6.0 erzeugt wurde, sollten Sie sich zunächst an die generierten To-Do-Listen halten. Diese geben eine

Übersicht über notwendige oder sinnvolle Umwandlungen. Außerdem sollten Sie die API-Anweisungen, die über das .NET Framework verfügbar sind, ersetzen. Im vorliegenden Beispiel wird auf den Einsatz der API-Funktionen für die Dateisuche verzichtet. Das .NET Framework bietet eine Vielzahl an Methoden an, die der Verzeichnis- und Dateianalyse dienen und sich allesamt im Namensraum *System.IO* befinden. Diese verhalten sich so wie die bereits behandelten API-Funktionen.

In der .NET-Programmvariante zum netzwerkübergreifenden Suchen und Ko-

### Listing 5

#### Rekursive Dateioperationen mit Netzwerkunterstützung unter Visual Basic .NET.

```
Option Strict Off
Option Explicit On
Imports System.IO

Module GLOBMODUL
    Public Cancel As Boolean

    Sub RecursiveFileOperation(ByVal Ausgangspfad As String, _
        ByVal Suchmaske As String, _
        ByVal Funktion As String, ByVal Param1 As String, _
        ByVal lst As System.Windows.Forms.ListBox, _
        ByVal opt As System.Windows.Forms.RadioButton, _
        ByVal stBar As System.Windows.Forms.StatusBar)
        On Error Resume Next
        Dim zPfad As String
        Dim ZielPfad As String
        If Ausgangspfad.EndsWith("\") = False Then
            Ausgangspfad = Ausgangspfad & "\"
        End If
        If Ausgangspfad.StartsWith("\") = False _
            And Mid(Ausgangspfad, 2, 1) <> ":" Then
            Exit Sub
        End If
        Dim FileArray As System.Array
        Dim FileInfo As System.IO.FileInfo
        FileArray = Directory.GetFiles(Ausgangspfad, "*.*)")
        For Each Eintrag As String In FileArray
            FileInfo = New FileInfo(Eintrag)
            If Cancel = True Then Exit Sub
            System.Windows.Forms.Application.DoEvents()
            If LCase(Funktion) = "find" Then
                If opt.Checked = True Then
                    If LCase(Eintrag) = LCase(Suchmaske) Then
                        lst.Items.Add(Eintrag)
                        lst.Refresh()
                    End If
                Else
                    If InStr(LCase(Eintrag), LCase(Suchmaske)) > 0 _
                        Then
                        lst.Items.Add(Eintrag)
                        lst.Refresh()
                    End If
                End If
            End If
            ElseIf LCase(Funktion) = "backup" Then
                If Len(Ausgangspfad) > Len(Suchmaske) Then
                    zPfad = Right(Ausgangspfad, Len(Ausgangspfad) -
                        Len(Suchmaske) - 1)
                Else
                    zPfad = ""
                End If
                If zPfad <> "" Then zPfad = zPfad & "\"
                ZielPfad = Param1 & "\" & zPfad
                If Not Directory.Exists(ZielPfad) Then
                    MkDir(Filtern(ZielPfad))
                End If
                If opt.Checked = True Then
                    If Not File.Exists(ZielPfad & Eintrag) Then
                        stBar.Text = "Kopiere ..."
                        FileCopy(Eintrag, ZielPfad & _
                            GetFileByPath(Eintrag))
                    End If
                Else
                    stBar.Text = "Kopiere ..."
                    FileCopy(Eintrag, ZielPfad & _
                        GetFileByPath(Eintrag))
                End If
            End If
        Next
        Dim DirArray As System.Array
        DirArray = Directory.GetDirectories(Ausgangspfad)
        For Each Pfad As String In DirArray
            System.Windows.Forms.Application.DoEvents()
            If Cancel = True Then Exit Sub
            If Not (lst Is Nothing) Then lst.SelectedIndex = _
                lst.Items.Count - 1
            stBar.Text = "Bearbeite Verzeichnis ..."
            RecursiveFileOperation(Pfad, Suchmaske, Funktion, _
                Param1, lst, opt, stBar)
        Next
    End Sub

    Function GetFileByPath(ByVal Path As String) As String
        Dim Position As Short = RInStr(Path, "\")
        If Position = 0 Then
            Return Path
        Else
            Return Right(Path, Len(Path) - Position)
        End If
    End Function
End Module
```

### Listing 6

#### Kopieren unter Visual Basic .NET.

```
Private Sub btnOK_Click(ByVal sender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles btnOK.Click
    Dim DbPfad As String
    Dim Quell As String
    Dim Ziel As String
    Cancel = False
    btnCancel.Focus()
    Me.Cursor = System.Windows.Forms.Cursors.WaitCursor
    If LCase(lbZiel.Text) & "\" <> LCase(DbPfad) Then
        If lbQuell.Text <> lbZiel.Text Then
            If InStr(lbZiel.Text, lbQuell.Text) = 0 Then
                Quell = lbQuell.Text : Ziel = lbZiel.Text
            RecursiveFileOperation(Quell, Quell, "backup", Ziel, _
                Nothing, optBackup2, Me.stBar)
            MsgBox("Das Kopieren ... wurde abgeschlossen!", _
                MsgBoxStyle.Exclamation, "Kopieren beendet!")
            Me.stBar.Text = "Wählen Sie ..."
            Me.Cursor = System.Windows.Forms.Cursors.Default
        Else
            Me.Cursor = System.Windows.Forms.Cursors.Default
            MsgBox("Das Zielverzeichnis darf ...!", _
                MsgBoxStyle.Exclamation, "Falsche Pfadwahl")
            Exit Sub
        End If
    Else
        Me.Cursor = System.Windows.Forms.Cursors.Default
        MsgBox("Quelle- und Zielpfad sind identisch!", _
            MsgBoxStyle.Exclamation, "Falsche Pfadwahl")
        Exit Sub
    End If
End Sub

Private Sub NetCtl1_AfterSelect(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.TreeViewEventArgs) _
    Handles NetCtl1.AfterSelect
    On Error Resume Next
    Dim UNCPath As String
    If resOptNet.Checked Then
        UNCPath = NetCtl1.NodeText
        If VB.Left(UNCPath, 2) = "\\\" And _
            IsNetworkPath(UNCPath) Then
            Dir1.Path = UNCPath
            lbQuell.Text = UNCPath
        End If
    End If
End Sub

Private Sub NetCtl1_AfterSelect(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.TreeViewEventArgs) _
    Handles NetCtl1.AfterSelect
    On Error Resume Next
    Dim UNCPath As String
    If resOptNet.Checked Then
        UNCPath = NetCtl1.NodeText
        If VB.Left(UNCPath, 2) = "\\\" And _
            IsNetworkPath(UNCPath) Then
            Dir1.Path = UNCPath
            lbQuell.Text = UNCPath
        End If
    End If
End Sub
```

pieren liegt demnach die Hauptänderung in der Methode *RecursiveFileOperation* (vergleiche Listing 5). Die Methode ist auch hier im Modul *GlbModul* definiert. Um über die Prozedur einen Bezug zur Statusleiste der Suchen- und Kopieren-Dialoge herstellen zu können, wird hier das jeweilige Statuszeilenobjekt über den Parameter *stBar* übergeben. Alle weiteren Parameter entsprechen denen der Visual-Basic-6.0-Programmfassung.

Die Verzeichnisanalyse ist jedoch anders aufgebaut als in VB 6.0. Zuerst werden zum Ausgangsverzeichnis über die Methode *Directory.GetFiles* die enthaltenen Dateien abgefragt und mittels eines *For-Each*-Konstruktes durchlaufen. In Abhängigkeit von der übergebenen Funktion werden Dateien gesucht (*Find*) oder kopiert (*Backup*). Anders als bei VB 6.0 werden Eintragsinformationen zu Dateien bereits standardmäßig mitsamt dem Verzeichnis zurückgeliefert. Mit der Funktion *GetFileByPath* kann aus einer Kombination aus Dateiname und Pfad der Dateiname selbst abgespalten werden. Für Kopiervorgänge muss der Dateiname mit dem neuen Zielverzeichnis verkettet werden.

Nachdem sämtliche Dateieinträge verarbeitet sind, werden zum Ausgangsverzeichnis über die Methode *Directory.GetDi-*

*rectories* die darin enthaltenen Unterverzeichniseinträge ermittelt. Sämtliche Unterverzeichniseinträge werden ebenfalls über ein *For-Each*-Konstrukt durchlaufen. Zu jedem Unterverzeichnis wird dann die Funktion *RecursiveFileOperation* erneut rekursiv aufgerufen. Nachdem damit die Hauptfunktion für das .NET-Programm verfügbar ist, können Sie die Dialoge zum Suchen und Kopieren einbinden. Im Suchen-Dialog nutzen Sie erneut die Ereignisprozedur *btnFind\_Click* und im Kopieren-Dialog die Ereignisprozedur *btnOK\_Click*.

Die Ereignisprozedur zum Kopieren mit .NET ist in Listing 6 enthalten und ermöglicht den Vergleich mit der VB-6-Variante.

Die Lösungssätze bei VB 6.0 und VB.NET unterscheiden sich mitunter ganz erheblich voneinander. In .NET steht Ihnen eine umfassende Klassenbibliothek zur Verfügung, die fast für alle Aufgaben

das erforderliche Werkzeug bereitstellt und viele Programmieraufgaben vereinfacht. Bei der Migration von Projekten, die ursprünglich mit VB 6.0 angelegt wurden, ist es sinnvoll, die erweiterten Funktionalitäten des .NET Framework einzusetzen. Bei einer Portierung von VB 6.0 auf VB.NET ist die Zeit für die Nachbearbeitung nicht zu unterschätzen. |||||

[1] Andreas Maslo, Netzwerkreisourcen mit VB 6 und VB.NET analysieren, dotnetpro 6/2003, Seite 108 ff.

Anzeige 1/8 quer